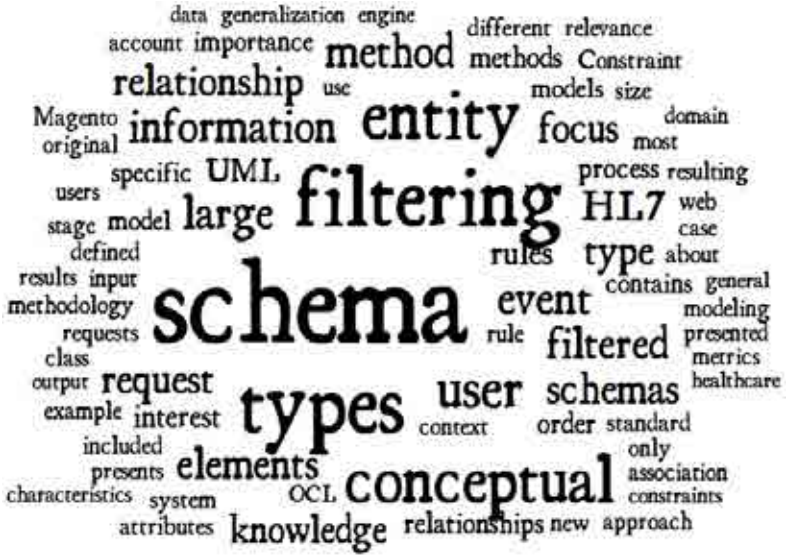
A FILTERING ENGINE FOR LARGE CONCEPTUAL SCHEMAS



Author: Antonio Villegas Niño
Address: Department of Service and Information System Engineering
Building Omega, S206
C. Jordi Girona 29, 1-3
08034 Barcelona, Spain
Email: avillegas@essi.upc.edu
Telephone: +34 93 413 71 74
Fax: +34 93 413 78 33

DOCTORAL STUDIES

Doctoral Thesis

– December 11, 2012 –

A FILTERING ENGINE FOR LARGE CONCEPTUAL SCHEMAS

Antonio Villegas Niño

Advisors

Antoni Olivé Ramon

Maria-Ribera Sancho Samsó

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

This thesis was written in LaTeX.
Figures were drawn using Inkscape.

A thesis presented by Antonio Villegas Niño in partial fulfillment of the requirements
for the degree of *Doctor per la Universitat Politècnica de Catalunya*

In loving memory of my sister,
Laura

Acknowledgements

At the end of my thesis I would like to thank all those people who made this thesis possible and an unforgettable experience for me. I would like to express my sincere gratitude to all who have supported and contributed the achievement of this goal.

At this moment of accomplishment, first of all I would like to thank my advisors, Dr. Antoni Olivé and Dr. Maria-Ribera Sancho. Working with them has indeed been a great pleasure. They have taught me how to enjoy researching. This work would not have been possible without their guidance, support, and encouragement.

Thanks to my colleagues in the Grup de Recerca en Modelització Conceptual and the Department of Service and Information System Engineering for their interesting and useful comments on my research work and for giving me their utmost support. I would also thank to all the reviewers that have contributed with their comments in different stages of this research.

I am particularly grateful to Josep Vilalta, owner of Vico Open Modeling and my mentor in the HL7 world, for providing necessary infrastructure and resources to accomplish my research work in the healthcare domain. Under his guidance I successfully overcame many difficulties and learned a lot. I owe gratitude to Diego Kaminker and Alberto Saez, HL7 technical experts, who willingly devoted so much time in giving guidance to me.

I expand my thanks to David Ortiz and Jose Maria Gomez, final year students who helped me in making really interesting research ideas come true through their final career projects. I wish them a great future.

I am indebted to my many colleagues for providing a stimulating and fun filled working environment at the office. My special appreciation to David Aguilera, Nikolaos Galanis, Jordi Piguillem, and David Sancho for being there all these years. I take this opportunity to say heartful thanks to Raimon Lapuente, Miquel Camprodon, Marc Rodriguez, and Oscar Martinez, with whom I had the opportunity to share great moments during this journey. I would like to offer my special thanks to Noël and Laura. And I also want to thank Ruth for her truly support. Thank you doesn't seem sufficient but it is said with appreciation and respect to all of them for their support, encouragement, care, understanding, and precious friendship.

It's my fortune to gratefully acknowledge the support of Montse. During the inevitable ups and downs of conducting my research she often reminded me life's true priorities. She was always beside me during the happy and hard moments to push me and motivate me. I doubt that I will ever be able to convey my appreciation fully, but I owe her my eternal gratitude.

My special acknowledgements go to my parents Antonio and Victorina for their sincere encouragement and inspiration throughout my research work and lifting me uphill this phase of life. Together we have experienced difficult times during the last years but they were always there to make me look further and find the motivation to continue. Words are short to express my deep sense of gratitude towards them.

This work has been partly supported by the *Ministerio de Ciencia y Tecnología* under TIN2008-00444 project, *Grupo Consolidado*.

This work has been partly supported by the *Universitat Politècnica de Catalunya*, under *FPI-UPC* program.

Abstract

Nowadays, the need for representation and conceptualization of real world information has dramatically increased. Organizations evolution and diversification require the management and maintenance of large amounts of knowledge from their domains of interest. That growth also has an impact in the size of conceptual schemas of information systems, making them larger. The sheer size of those schemas transforms them into very useful artifacts for the communities and organizations for which they are developed. However, the size of the schemas and their overall structure and organization make it difficult to manually extract knowledge from them, to understand their characteristics, and to change them.

There are many information system development activities in which people needs to get a piece of the knowledge contained in the conceptual schema. For example, a conceptual modeler needs to check with a domain expert that the knowledge is correct, a database designer needs to implement that knowledge into a relational database, a software tester needs to write tests checking that the knowledge has been correctly implemented in the system components, or a member of the maintenance team needs to change that knowledge.

Dealing with large conceptual schemas is one of the most challenging and long-standing goals in conceptual modeling. The purpose of this thesis is to formally define a new information filtering methodology to help users of very large conceptual schemas to understand the characteristics and knowledge these schemas contain. This thesis analyzes and describes the different phases of an information filtering engine, identifies and studies several properties of relevance for elements of large conceptual schemas, provides a catalog of specific filtering requests to explore a schema in several filtering scenarios, and implements and evaluates the efficiency and effectiveness of a filtering engine prototype with several real case studies.

The filtering methodology studies the characteristics of the knowledge contained within a large conceptual schema, and proposes ways to select and represent the user interest in order to specialize the results of a filtering engine. Consequently, a user focus on a fragment of the large schema of interest to her and our methodology automatically obtains a reduced conceptual schema extracted from the large schema and focused on the knowledge that has a closer relation with the focus of the user. Such filtered conceptual schema is a subset of the original one, and because of its reduced size it is more comprehensible to the user.

The filtering approach provides knowledge extraction techniques aligned with the user interest representation, and presents such knowledge in an appropriate way to simplify its understanding. Furthermore, using this filtering approach the large conceptual schemas are navigated more quickly, increasing their usability and reducing the user effort.

Contents

List of Figures

List of Tables

List of Algorithms

Anyone who has never made a mistake
has never tried something new.

Albert Einstein (1879-1955)

1 Introduction

This thesis provides a filtering methodology and engine to automatically extract relevant information from a very large conceptual schema in order to help users to understand and deal with such kind of schemas. Through the different chapters, we formally construct and describe the components that conform our filtering methodology and the artifacts of a filtering engine prototype that proves the effectiveness and efficiency of our research contributions. This chapter introduces our work, details our research approaches, and presents the structure of our dissertation.

The chapter starts with a brief explanation about the motivation of this thesis and the antecedents in the history of conceptual modeling and software engineering in Sect. 1.1. Along the development of this thesis we have followed the design-science research methodology. Section 1.2 presents the fundamental characteristics of this research paradigm and associates them with the research stages in this thesis. Section 1.3 enumerates and describes the main research contributions of the thesis and presents our main objective of providing a filtering engine for very large conceptual schemas. Finally, the overview of the structure of this document, including a description of its chapters and a reading guide is presented in Sect. 1.4.

1.1 Motivation and antecedents

Software engineering is constantly evolving. In the last years, the different processes and methodologies that take part in this field have become more flexible and easy to change and reuse. The development of software artifacts has begun to focus its activities on the usage of conceptual schemas and the abstraction and conceptualization of information domains, in the same way as other branches of engineering do.

A conceptual schema provides an abstraction layer between the real-world knowledge and the portion of that knowledge that is really useful in the development of an information system. This abstraction provides simplification to describe real concepts as general ones without taking into account the development technology of the final stages in the software engineering process.

Conceptual modeling can be defined as the software engineering activity that must be done to obtain the conceptual schema of an information system [87]. We define information system as a designed system that collects, stores, processes and distributes information about the state of a domain. The conceptual schema of an information system contains a representation of the knowledge that results from the process of gathering, classifying, structuring and maintaining all relevant characteristics appearing in a domain that are useful about the information system.

The development process of information systems always includes a conceptual schema. Sometimes, such schema can be explicitly reproduced as a piece of documentation and, sometimes, the schema is shared in the minds of the stakeholders. In any case, the conceptual schema of an information system exists, although obviously to have the schema explicitly is always the best choice. Note that if shared in the stakeholders minds it may take differences due to the inherent differences of thought we have as human beings.

Traditionally, conceptual modeling has been seen as a supporting activity in the software engineering process. Conceptual schemas have been defined in the initial stages of such process as documentation artifacts, rarely maintained up to date after the design phase. Fortunately, that situation is really changing. The emergence of model-driven approaches [101, 4, 68] increases the importance of conceptual schemas and their participation as key artifacts in the software development activities. Model-driven software engineering aims to generate software (including both code and documentation) from a specified information model, which may be a conceptual schema of an organization domain. This new role given to conceptual schemas implies that the specification and comprehension of conceptual schemas are main tasks for all the stakeholders of an information system.

There are many information system development activities in which people needs to get a piece of the knowledge contained in the conceptual schema. For example, a conceptual modeler needs to check with a domain expert that the knowledge is correct, a database designer needs to implement that knowledge into a relational database, a software tester needs to write tests checking that the knowledge has been correctly implemented in the system components, or a member of the maintenance team needs to change that knowledge.

One of the most challenging and long-standing goals in conceptual modeling, and therefore in software engineering, is to understand, comprehend and work with very large conceptual

schemas [88, 72]. Nowadays, the need for representation and conceptualization of real world information has dramatically increased. Organizations evolution and diversification require the management and maintenance of large amounts of knowledge from their domains of interest. Furthermore, data mining and knowledge extraction are becoming trending topics in business processes.

That growth also has an impact in the size of conceptual schemas of information systems, making them larger. The sheer size of those schemas transforms them into very useful artifacts for the communities and organizations for which they are developed. They are not only becoming a key artifact in software engineering, but also a conceptual representation of the whole *know-how* of the organization and information system they represent. However, the size of the schemas and their overall structure and organization make it difficult to manually extract knowledge from them, to understand their characteristics, and to change them. It is clear that to be useful, large containers of information, as large conceptual schemas are, need tools that can extract the portion of knowledge of interest to a particular user at a time.

That situation already happened with the World Wide Web. Until the uprising of web search engines [21], the knowledge that the web contained was only partially accessible due to its huge size and difficulty of finding what was sought. After them, the web has become into something useful and really easy to use due to the existing tool support that allows automatic information filtering. Nowadays not only experts can search the web, but also everybody with a computer and an information need is capable of that.

The aim of information filtering is to expose users to only information that is relevant to them. There are many filtering systems of widely varying philosophies [55], but all share the goal of automatically directing the most valuable information to users in accordance with their needs, and of helping them using their limited time and information processing capacity most optimally.

The purpose of this thesis is to define a new information filtering methodology to help users of very large conceptual schemas to understand the characteristics and knowledge these schemas contain. This thesis analyzes and defines the different phases of an information filtering engine. The method studies the characteristics of the knowledge contained within a large conceptual schema, and proposes techniques to select and represent the user interest in order to specialize the results of a filtering engine. Our approach provides knowledge extraction techniques aligned with the user interest representation, and presents such knowledge in an appropriate way to simplify its understanding. Furthermore, using this filtering engine the conceptual schemas are navigated more quickly, increasing its usability and reducing the user effort.

Therefore, the general problems addressed in this thesis are:

- Automating the extraction of knowledge of interest to the user from a large conceptual schema.
- Using inherent properties of large conceptual schemas to analyze the importance and interest of the elements in the schema.
- Introducing a request/response flow to the user interaction with a large conceptual schema
- Reducing the time and effort a user requires to understand a large conceptual schema.

1.2 Research approach

Along the development of this thesis we have followed the design-science research methodology. The design-science paradigm has its roots in engineering and the sciences of the artificial. It is fundamentally a problem-solving paradigm that consists of activities aimed at constructing and evaluating artifacts addressed to fulfill the requirements of organizations as well as developing their associated research theories. Table 1.1 summarizes the seven guidelines of design-research.

According to Hevner et al. in [59] the fundamental principle of design-science research is that knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact. That is, design-science research requires the creation of an innovative, purposeful artifact for a specified problem domain. Because the artifact is purposeful, it must yield utility for the specified problem. Hence, thorough evaluation of the artifact is crucial. Novelty is similarly crucial since the artifact must be innovative, solving a heretofore unsolved problem or solving a known problem in a more effective or efficient manner.

Guideline	**Description**
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires the usage of available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 1.1. Design-Science Research Guidelines [59].

Thus, design-science research is differentiated from the practice of design. The produced artifact itself must be rigorously defined, formally represented, coherent, and internally consistent. The process by which it is created, and often the artifact itself, incorporates or enables a search process whereby a problem space is constructed and a mechanism posed or enacted

to find an effective solution. The results of the design-science research must be communicated effectively both to a technical audience (researchers who will extend them and practitioners who will implement them) and to a managerial audience (researchers who will study them in context and practitioners who will decide if they should be implemented within their organizations).

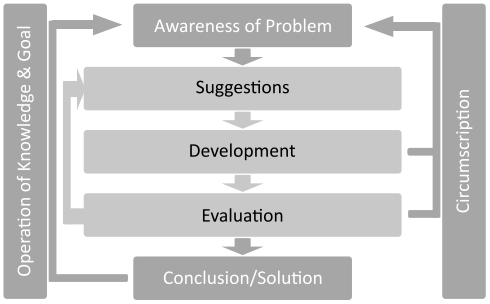


Figure 1.1. Reasoning on Design Cycle.

Fig. 1.1 illustrates the course of a general design cycle, as Takeda et al. analyzed in [114]. In this model all design begins with *Awareness of a problem*. *Suggestions* for a problem solution are abductively drawn from the existing knowledge/theory base for the problem area. An attempt at implementing an artifact according to the suggested solution is performed next. This stage is shown as *Development* in the diagram. Partially or fully successful implementations are then under *Evaluation* (according to the functional specification implicit or explicit in the suggestion). *Development*, *Evaluation* and further *Suggestion* are frequently iteratively performed in the course of the research (design) effort. The basis of the iteration, the flow from partial completion of the cycle back to *Awareness of the Problem*, is indicated by the *Circumscription* arrow. *Conclusion* indicates termination of a specific design project or solution.

Output	**Description**
Constructs	The conceptual vocabulary of a domain
Models	A set of propositions or statements expressing relationships between constructs
Methods	A set of steps used to perform a task
Instantiations	The operationalization of constructs, models and methods
Better theories	Artifact construction as analogous to experimental natural science

Table 1.2. The Outputs of Design Research.

Table 1.2 summarizes the outputs that can be obtained from a design research effort [122]. March and Smith propose in [74] four general outputs for design research: constructs, models, methods, and instantiations. Constructs arise during the conceptualization of the problem and are refined throughout the design cycle. Models are proposals for how things are. Methods are goal directed plans for manipulating constructs so that the solution statement model is realized. Instantiations are the realization of the artifact in an environment. Finally, the overall process of design-science research can contribute to have better theories as output.

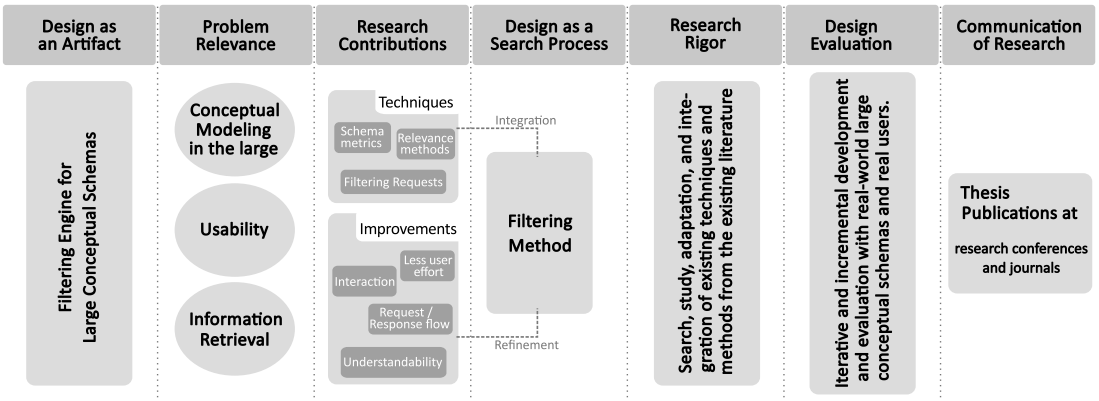


Figure 1.2. Design Research in this Thesis.

The characteristics of our research have a clear relationship with the guidelines of design-science research. The first stage in design research is the *awareness of the problem*. In our case, we found that there is a lack of a real filtering method for large conceptual schemas. As stated in Sec. 1.1, nowadays large schemas are difficult to understand. Existing techniques to reduce the complexity of large schemas provide static solutions, but do not include a possible request/response interaction flow between the user and the schema based on the user's information needs. We followed the *suggestions-development-evaluation* cycle until the achievement of a complete information filtering engine that fulfilled the goals of the thesis. During the research iterations, we continually searched and studied the proposals from the existing literature on conceptual modeling in the large in order to be aware of techniques and methods to deal with large conceptual schemas that could be adapted or integrated on our work.

Figure 1.2 shows how design research was applied to this thesis following the guidelines of Tab. 1.1. The problem of extracting knowledge from large conceptual schemas is qualified as relevant from the perspectives of conceptual modeling, usability and information retrieval. The importance of providing a solution to this problem justifies the design of a filtering engine for large conceptual schemas and, consequently, the existence of this thesis.

1.3 Research contributions

The main objective of this thesis is to provide a **filtering engine of very large conceptual schemas** to help users to easily **extract** from them the most **relevant knowledge** for a particular purpose.

As we mentioned in the previous sections, the usability of large schemas improves due to the emergence of specific tools and methods that make easier the tasks of querying, modifying, updating and constructing such schemas by simplifying their required comprehension effort. What is needed is to study and define the different phases of an information filtering engine, in the context of very large conceptual schemas. Also, it is mandatory to study the characteristics of the knowledge contained within them, and propose methods to select and represent the user interest in order to specialize the results of the filtering engine. Furthermore, it is mandatory to provide knowledge extraction techniques aligned with the user interest representation, and to present such knowledge in an appropriate way to simplify its understanding.

The information filtering methodology for large conceptual schemas presented in this thesis can be outlined in the following sub-goals:

- **Identify, study and describe several properties of relevance for elements of large conceptual schemas.** The properties that we address are structural properties, based on the elements of the schema and the characteristics defined about them.
- **Present useful feedback to the user.** Most of the existing methods in the literature do not take advantage of the available graphical possibilities when presenting conceptual schemas. The use of different colors, sizes and shapes to highlight the elements of a schema will reduce the understanding effort of the user.
- **Evaluate the usability of the filtering engine.** A goal and a mandatory requirement for a filtering engine is to study and adapt existing techniques of evaluation in order to reinforce and clarify its contribution. Experimentation with real conceptual schemas is also a related goal to provide quality assurance.

These goals are the principal requirements that our thesis must accomplish. To this end, the main contributions of this thesis are:

- A general filtering method for large conceptual schemas.
- A catalog of specific filtering requests to explore large conceptual schemas.
- A web-based filtering engine for large conceptual schemas.
- A study of metrics of relevance for elements of large conceptual schemas.
- An adaptation of the filtering methodology to complex schemas from the healthcare domain.

All these contributions are introduced in the following subsections.

1.3.1 General Filtering Method for Large Conceptual Schemas

As aforementioned, this thesis presents a method that filters large conceptual schemas to extract the knowledge of interest to the user according to the user's information need. The problem appears in many information systems development activities in which people needs to operate for some purpose with a piece of the knowledge contained in that schema. The method automates the process and reduces the time and effort a user requires to understand a large conceptual schema. A user focus on a fragment of the large schema of interest to her and the method automatically obtains a reduced conceptual schema extracted from the large schema and focused on the knowledge that has a closer relation with the focus of the user. Such filtered conceptual schema is a subset of the original one, and because of its reduced size it is more comprehensible to the user. The user may then start another interaction with a different focus, until she has obtained all knowledge of interest. The different stages of the filtering method are:

1. **Metrics Processing** The first stage applies relevance metrics to the elements of the original large schema in order to discover which are the most relevant ones for the user.
2. **Entity and Event Types Processing** The second stage selects the entity and event types from the large schema that will appear in the resulting filtered schema.
3. **Relationship Types Processing** This stage selects the relationship types from the large schema that will appear in the resulting filtered schema. This process makes use of projection and redefinition to align the relationships with the user's interest.
4. **Generalizations Processing** This stage selects the generalization relationships whose members belong to the filtered schema, processes them to avoid redundancies, and includes them in the output.
5. **Schema Rules Processing** This stage processes the schema rules of the original schema in order to include into the output those that affect elements of interest to the user.
6. **Data Types Processing** This stage selects those data types referenced by elements of the filtered schema in order to add them into it.
7. **Presentation** The last stage deals with the representation of the filtered schema to the user in order to maximize its understandability.

The filtering method is general and can be directly adopted for any tool or technique that deals with large schemas or ontologies specified on several modeling languages. Despite that, we centered our explanation on large conceptual schemas written in UML/OCL [84, 85]. Nowadays, this pair of languages are the *de-facto* standard in conceptual modeling and most of the users of conceptual schemas are familiar with them.

1.3.2 Catalog of Specific Filtering Requests for Large Conceptual Schemas

Based on the characteristics introduced in our general method to filter large conceptual schemas, we define a set of specific filtering requests a user is capable of employ to interact with a target schema. Each of these filtering requests are instantiations of the general method with specific characteristics that provide concrete value to the user. We distinguish between filtering requests according to their input parameters, which represent the user's information need and modifies the kind of obtained output. Our catalog contains the specification of the following filtering requests:

- $\mathcal{F}_1$: Filtering request for entity and relationship types, described in Sect 6.3.1 of Ch. 6.
- $\mathcal{F}_2$: Filtering request for schema rules, described in Sect 6.3.2 of Ch. 6.
- $\mathcal{F}_3$: Filtering request for event types, described in Sect 6.3.3 of Ch. 6.
- $\mathcal{F}_4$: Filtering request for a conceptual schema, described in Sect 6.3.4 of Ch. 6.
- $\mathcal{F}_5$: Filtering request for behavioral entity types, described in Sect 6.3.5 of Ch. 6.
- $\mathcal{F}_6$: Filtering request for contextualized types, described in Sect 6.3.6 of Ch. 6.

The usage of specific filtering requests allows the user to navigate through a large schema following and iterative and dynamic request/response cycle. To our knowledge, none of the existing approaches in the literature allow dynamism, or at least little interaction with their produced output, which mainly consists of a static reorganization of the structure of the large schema.

1.3.3 Filtering Engine for Large Conceptual Schemas

Following the guidelines of the design-science research discussed in Sect. 1.2, we have developed an artifact to evaluate the benefits of our research approach. Our artifact consists of a filtering engine that implements the specific filtering requests in a web-based environment.

Our engine contains a core that is responsible for maintaining and access the characteristics of a large conceptual schema. In addition, the core is under control of the filtering requests that require querying the schema in order to serve the information needs of the user. The conjunction of the specific information filtering requests and the core of the engine is implemented as a web service. This architectural decision allows an easy interaction with web clients and increase the technology independence, and therefore, the usability of the overall system.

1.3.4 Relevance Metrics for Large Conceptual Schemas

There are several techniques and associated tools for the visualization and comprehension of large conceptual schemas or ontologies. The group of techniques we see more appropriate for our purposes is the one called focus+context. In this techniques, the user focus on a single element (node), which becomes the central one, and the rest of the nodes are presented around it, reduced in size until they reach a point that they are no longer visible. However, the techniques do not distinguish between the nodes presented around the central one: all nodes and edges are assumed to be equally relevant to the user.

The general method for filtering large schemas requires to know the structure and relevance of the schema elements that conform a large conceptual schema to align them with the user's information need. The thesis formalizes several metrics over the schema elements with relation to their relevance in the schema to be used in the filtering process. There are several contributions in the literature about schema metrics that have been adapted and extended in this thesis to cover a larger amount of knowledge about schema elements. The thesis also include some comparisons between relevance metrics that indicate in which situations it is preferable to use ones among others.

1.3.5 Adaptation of the Filtering Methodology to HL7 V3 schemas

The Health Level Seven International (HL7) is an standards developing organization dedicated to providing a comprehensive framework and related standards for the exchange, integration, sharing, and retrieval of electronic health information that supports clinical practice and the management, delivery and evaluation of health services. HL7 V3 [9, 18] is one of the most widely used HL7 standards that enables disparate healthcare applications to exchange clinical and administrative data through messages or documents whose structure is based on a large set of conceptual schemas. The amount of knowledge represented in the HL7 V3 schemas is very large and the sheer size of those models makes them very useful to the healthcare communities for which they were developed. However, the size of HL7 V3 schemas and their organization make it very difficult for those communities to manually extract knowledge from them.

The HL7 V3 consists of a set of conceptual schemas that together conform a large conceptual schema of the healthcare domain. Those schemas are interconnected, and the knowledge of some the concepts defined within is spread through different schemas. Furthermore, some of the HL7 V3 schemas are refinements of other HL7 V3 schemas, which indicates the existence of some kind of hierarchy between schemas. Our information filtering methodology is able to directly work with the union of the HL7 V3 schemas, but it is possible to take advantage of their particular structure and different characteristics with some adaptations in our filtering engine. This thesis shows how to deal with HL7 V3 schemas to improve the quality of the output obtained by our filtering methodology.

1.4 Overview of the thesis

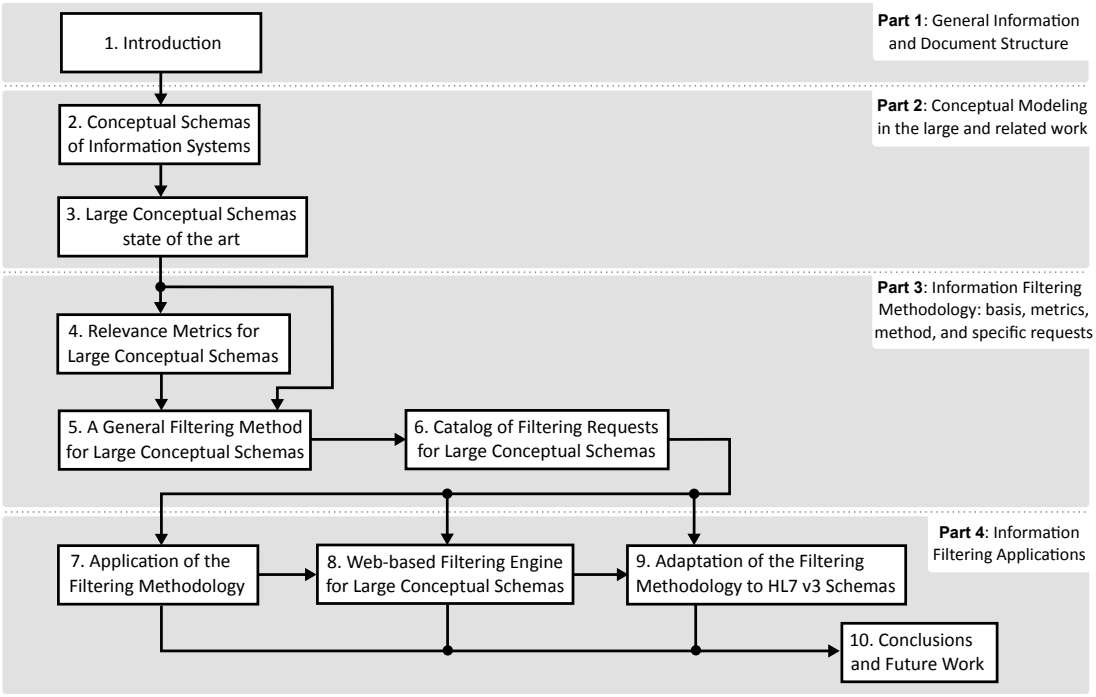


Figure 1.3. Structure of the thesis.

The thesis is structured in 10 chapters. This chapter introduces the structure of the document, describes its contents and the topics it addresses, and helps the user in the reading of the thesis. The document is structured in four parts well differentiated as we can see in Fig. 1.3:

Part 1. Structure of the thesis. It is composed of Ch. 1, which helps the reader in the identification of parts of the thesis that may be relevant to him/her, including the research approach, thesis goals and main contributions.

Part 2. Conceptual modeling. It is composed of Chapters 2 and 3. It formally describes the structure and characteristics of conceptual schemas, presents the problem of working with large schemas, and explains a general overview of different approaches on this field with the same purpose that ours: simplify the knowledge extraction from large schemas and increase their understandability.

Part 3. Information filtering methodology. It is composed of Chapters 4, 5, and 6. It deals with the description of a general information filtering methodology to apply to large conceptual schemas, why this method is necessary, and which are the most important components of this method. These chapters show the set of relevance metrics of schema elements that are the core of the methodology, and a catalog of specific filtering requests to extract knowledge from a schema.

Part 4. Information filtering application. It is composed of Chapters 7, 8, 9 and 10. These chapters show the application of the filtering methodology introduced in the third part to a real-case example consisting in a large schema from the e-commerce domain, and a web-based implementation of the methodology as a filtering engine. In addition, Ch. 9 presents the characteristics of a very different large schema from the healthcare domain. This schema is composed of a set of interrelated subschemas with repeated elements. We describe the adaptation of our methodology to deal with this kind of schemas, and the corresponding results. Finally, we expose our conclusions and the future work related to the activity of information filtering of large conceptual schemas.

The reader may go directly to part 3 if he/she is familiarized with conceptual modeling and conceptual schemas, their use in software engineering, and the difficulties of knowledge extraction when the schemas are of large size.

In the following we comment the content covered in each chapter of this thesis.

Chapter 2: Conceptual Schemas of Information Systems

This chapter defines the concept of conceptual schema and explains its key role in the field of conceptual modeling of information systems. We formally present the different components and schema elements that are included within a conceptual schema defined in UML and OCL. This formalization is referenced along the chapters of this thesis. We use it in the definition of several concepts within the information filtering methodology such as relevance metrics or specific filtering requests.

Chapter 3: Conceptual Modeling in the Large

This chapter presents the problem of extracting knowledge from large conceptual schemas. We show some examples that illustrate the need for methods to simplify the required effort a user has to dedicate to entirely understand a large schema. The chapter includes an overview of existing tools, methods and techniques in the literature that address the problem of large conceptual schemas in several ways. Finally, we discuss the necessity of new methods based on information filtering to contribute to this working area.

Chapter 4: Relevance Metrics for Large Conceptual Schemas

This chapter formally presents metrics over large conceptual schemas specified in UML/OCL to compute the importance of schema elements and the closeness between them, among other relevance metrics. The metrics described in the chapter are a core element of the general filtering method. They are the power engine that allows to discover the schema elements that are of interest to the user according to his/her information needs at a given moment. We compare the metrics with different schemas in order to explore different behaviors and propose some guidelines of use depending on the structure of the schema the user wants to explore.

Chapter 5: A General Filtering Method for Large Conceptual Schemas

This chapter formalizes our general method for filtering large conceptual schemas and presents its requirements. In particular, we describe the different components that conform the filtering methodology and its expected input, with the relationship with the user's information needs. Finally, the chapter presents the concept of filtered conceptual schema and its characteristics as the output of our proposed filtering method.

Chapter 6: Catalog of Filtering Requests for Large Conceptual Schemas

This chapter formally introduces the different specific filtering requests a user can manipulate to iteratively extract knowledge from a large conceptual schema. A filtering request is an instantiation of the general information filtering method with specific behavior and particular requirements. These filtering requests are the main component of the user interaction with a large schema due to their request/response flow. The chapter explains the particularities of each of these specific filtering requests, including their different input parameters, their similarities and differences and the changing characteristics of their produced output.

Chapter 7: Application of the Filtering Methodology

This chapter describes the application of our filtering methodology to four different real-case large conceptual schemas from several domains. The chapter presents the main characteristics of the four schemas, the problems a user faces when working with them, and the facilities provided by our method. We experimentally evaluate the effectiveness and efficiency of our filtering approach with respect to these conceptual schemas and report the main results from the experimentation in order to demonstrate the benefits of using information filtering.

Chapter 8: Web-based Filtering Engine for Large Conceptual Schemas

This chapter explains the details of the filtering engine we have implemented in a web-based environment. We present the main components of the architecture of the filtering engine together with their intended functionality. The chapter describes the client side and server side of the filtering engine as a web service and demonstrates the requirements and benefits of this architectural decision. Finally, the chapter clarifies the web-based interaction with users by presenting the expected way of using the filtering engine itself.

Chapter 9: Adaptation of the Filtering Methodology to HL7 V3 Schemas

This chapter describes the application of our filtering methodology to a real-case large conceptual schema from the healthcare domain: the HL7 V3. Such schema is defined in a non-standard modeling language and is conformed by several interconnected subschemas that contain re-

peated concepts and special constructions that are worth explaining. We translate the HL7 V3 schemas into UML/OCL and propose modifications to our general information filtering methodology in order to take advantage of the particular aspects of HL7 V3 to improve the quality of the produced results.

Chapter 10: Conclusions and Future work

The final chapter presents the conclusions of the information filtering methodology and points out future work that may continue our research line to improve the usability of large conceptual schemas.

First, solve the problem.
Then, write the code.

John Johnson

2

Conceptual Schemas of Information Systems

The main objective of this thesis is to provide an information filtering engine and methodology to help users of very large conceptual schemas to understand the characteristics and knowledge these schemas contain. What is needed is an automatic extraction methodology following the characteristics of the specific information request a user indicates. To this end, our first task consist of introducing the basic concepts about conceptual schemas in order to clarify our vision and present general aspects about conceptual modeling that are used through several chapters of this document. We formally indicate which are the components of a conceptual schema, and present examples of conceptualization to understand these components and their function in the different stages of the software engineering and software development processes.

This chapter starts with a description about conceptual modeling in Sect. 2.1. Then, Sect. 2.2 presents the contents of complete conceptual schemas taking into account both structural and behavioral (sub)schemas. The second part of the chapter continues in Section 2.3 with an introduction to the modeling languages used to represent conceptual schemas. Concretely the aim of such section is centered in the UML and OCL as the modeling languages selected in this thesis. In addition to it, we include a mention to the entity-relationship model, which is the precursor of modern object-oriented approaches to model data. Finally, Section 2.4 summarizes the chapter.

2.1 Conceptual Modeling

Conceptual modeling can be defined as the activity that must be done to obtain the conceptual schema of an information system. We define information system as a designed system that collects, stores, processes and distributes information about the state of a domain of interest to an organization. Thus, conceptual modeling is one of the initial activities in the software engineering process that gathers, classifies, structures and maintains knowledge about a real-world domain. The main goal of conceptual modeling is to construct a conceptual model from the knowledge about the domain.

Conceptual schemas are the central unit of knowledge in the development process of information systems. A conceptual schema must include the definition of all relevant characteristics appearing in an organization that are useful in the task of representation (also know as conceptualization) of the information system. Such task consists of abstracting real-world knowledge into categories of concepts that may be related to other concepts.

Usually, conceptual schemas have been seen as documentation items in the software engineering process. However, since the initial stages of model-driven approaches [68] came out many years ago, conceptual schemas have increased its importance and participation as key artifacts in the software development activities. Model-driven approaches try to generate final software (including both code and documentation) from a specified model, which may be a conceptual schema of an organization domain. This new role given to conceptual schemas implies that the specification and comprehension of conceptual schemas are main tasks for the stakeholders of an information system.

A conceptual schema provides an abstraction layer between the real-world knowledge and the portion of that knowledge that is really useful in the development of an information system. This abstraction provides simplification to describe real concepts as general ones without taking into account the development technology of the final stages in the software engineering process. In early stages of the software development activity, having a conceptualized vision of the domain of interest for a particular information system by means of its conceptual schema helps stakeholders in order to successfully understand the structure and behavior of such domain of knowledge.

The development process of information systems always include a conceptual schema. Sometimes, such schema can be explicitly reproduced as a piece of documentation and, sometimes, the schema is shared in the minds of the stakeholders. In any case, the conceptual schema of an information system exists, although obviously to explicitly have the schema as an existing artifact available for all the stakeholders is always the best choice. Note that if it only exists scattered over the stakeholders minds it may produce development problems due to the inherent differences of thought we have as human beings.

Comprehension and understandability of conceptual schemas and their components are the main object of research of this thesis. To this end, we formally define the characteristics of complete conceptual schemas in the following sections of the chapter.

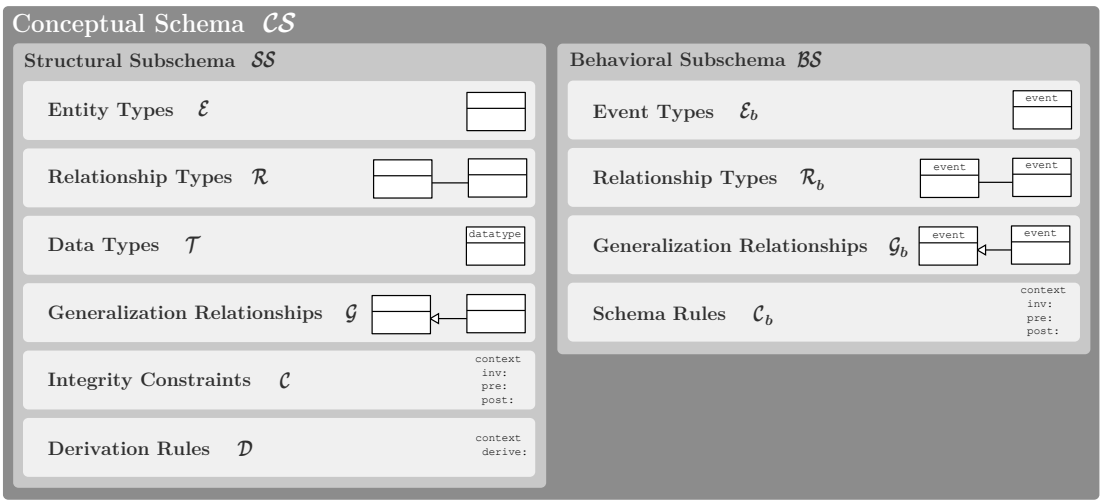


Figure 2.1. Structure of a conceptual schema.

2.2 Conceptual Schema

As aforementioned, conceptual schemas are one of the key artifacts in the software engineering and software development processes. To be complete, a conceptual schema should specify two subschemas: the structural schema, which deals with the static scope of the information system, and the behavioral schema, which indicates the dynamic component of the same system.

The structural schema is the part of the conceptual schema that consists on the set of entity and relationship types, as well as other elements that will be mentioned in the following section, used to observe the state of a domain in an specific moment. This part is also known as the static component of the whole knowledge of the information system because defines the concepts of interest to an information system plus the specific information about them, including associations, inheritance, and structural attributes.

On the other hand, the behavioral schema represents the valid changes in the domain state, as well as the actions that the system can perform. Changes in the domain state are domain events, and a request to perform an action is an action request event. We represent such events as special entity types following the same approach as in [89]. In the UML, we use for this purpose a new stereotype, that we call ≪event≫. A type with this stereotype defines an event type. The characteristics of events should be modeled like those of ordinary entities. We define a particular operation in each event type, whose purpose is to specify the event effect. To this end, we use the operation *effect*. The pre- and postconditions of this operation will be exactly the pre- and postconditions of the corresponding event.

Formally, we define a conceptual schema as a tuple $\mathcal{CS} = \langle \mathcal{SS}, \mathcal{BS} \rangle$, where $\mathcal{SS} = \langle \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ is the structural subschema, and $\mathcal{BS} = \langle \mathcal{E}_b, \mathcal{R}_b, \mathcal{G}_b, \mathcal{C}_b \rangle$ is the behavioral subschema. Figure 2.1 depicts the structure of a conceptual schema with all the components of the structural and behavioral subschemas. Next sections present the details about the components that conform the structural and behavioral subschemas.

2.2.1 Structural Subschema

The structural subschema of a conceptual schema contains the concepts of a concrete real-world domain. More precisely, each concept that has an important role in an organization and it is worth to be included in the information system is represented as an entity type. Imagine we need to conceptualize the domain of an organization that manages information about customers. This way, the concept *customer* which is important for the organization is represented in the information system by the entity type *Customer*. Real-world customers are represented by instances of the *Customer* entity type in the information system of the organization. Figure 2.2 presents this example.

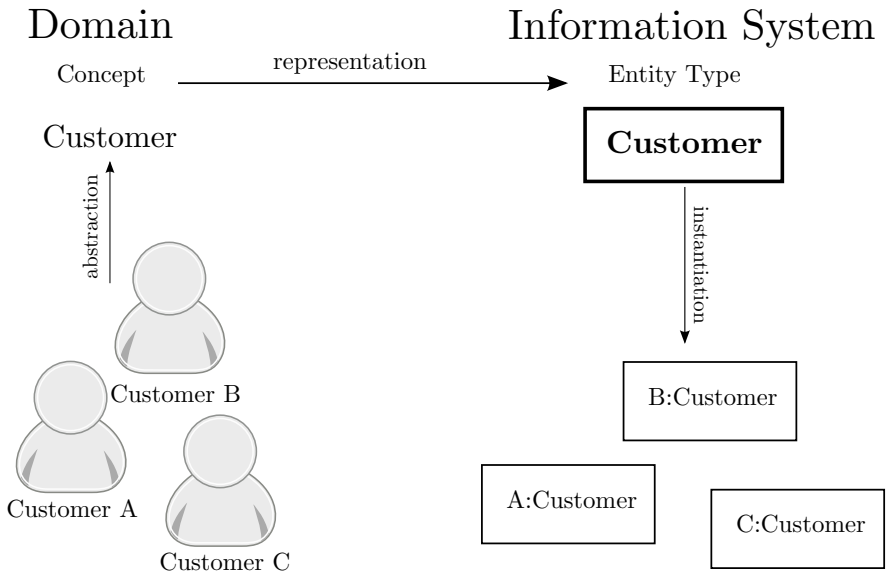


Figure 2.2. Representation of domain concepts as entity types.

Entity types can be interconnected. It is possible to have generalization/specialization relationships between any pair of them. In the previous example, the organization may have a special kind of customers, say gold customers. To denote this situation in the conceptual schema, it is possible to indicate that exists another entity type called *GoldCustomer* such that *Customer* IsA *GoldCustomer*. It means that GoldCustomer is an specialization of the entity type Customer, and that Customer is a generalization of GoldCustomer. Each information about Customer in the information system is inherited by the GoldCustomer entity type.

Furthermore, relationship types are basic elements in conceptual schemas. Each relationship type represents a connection or association between two or more entity types. In the previous example, imagine that it is important to maintain information about the parents of a customer to offer them special discounts. The conceptual schema should contain a new relationship type *IsParentOf(parent:Customer, child:Customer)*. It means that exists a reflexive relationship in Customer to denote the parents/children of an instance of Customer. The tags before the names of the entity types in the relationship represent the role names of such entity types as participants in the relationship.

The relationship types also have cardinalities. In the relationship *IsParentOf*, the cardinalities should be something like *Card(IsParentOf; parent, child) = (0,∞)* and *Card(IsParentOf; child, parent) = (0,2)*. This way, an instance of Customer can have zero or more instances of Customer as its children; and an instance of Customer can have at most two instances of Customer as its parents.

Another important characteristic to explain are attributes. An attribute is a property of an entity type that contains information about it. Concretely, an attribute can be seen as a special relationship with two participants: an entity type and a data type. If the organization of our example wants to maintain the name of the customers in its information system, it is possible to declare an attribute in the form of $HasName(Customer, name : String)$. It means that an instance of Customer is related to an instance of the data type String (e.g. the word `'John Smith'`) that is the name of the customer. Data types are basic types and there exists some predefined like String, Integer, Real or Boolean.

A structural schema may also contain schema rules. A schema rule is a property about a subset of the conceptual schema that must be always satisfied. In our example about the management of customers, an instance of the entity type Customer could be related to itself through the association IsParentOf. This way, we could have that a customer is one of its parents. Obviously, this behavior is not allowed and to denote it, we can specify a schema rule like:

$$Rule(NotParentOfItself : IsParentOf(c_1, c_2) \Rightarrow c_1 \neq c_2)$$

In following sections we will present the formal specification of schema rules by means of the OCL textual modeling language, which is the `de-facto` standard to define object-oriented constraints.

The structural subschema $\mathcal{SS} = \langle \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ of a conceptual schema $\mathcal{CS}$ is formally defined as a tuple that contains the following components:

- $\mathcal{E}$ is a set of **entity types**. Entity types may define a set of owned **attributes** $\mathcal{A}$.
- $\mathcal{R}$ is a set of **relationship types**. We denote by $r(p_1{:}e_1, ..., p_n{:}e_n)$ a relationship type r with participant entity types $e_1, ..., e_n \in \mathcal{E}$ playing roles $p_1, ..., p_n$ respectively. Note that the number of participants k in $r \in \mathcal{R}$ is the degree of r and $k > 1$. We see attributes as binary relationship types. We denote by *attr(e,t)* an attribute owned by an entity type $t \in \mathcal{E}$, named *attr*, and whose type is $t \in \mathcal{T}$.
- $\mathcal{T}$ is a set of **data types** and **enumerations**.
- $\mathcal{G}$ is a set of **generalization relationships**. Each $g \in \mathcal{G}$ represents a directed relationship between a pair of entity types $(e_i \rightarrow e_j)$ where $e_i, e_j \in \mathcal{E}$ indicating that e_i is a direct descendant of e_j and e_j is a direct ascendant of e_i.
- $\mathcal{C}$ is a set of **integrity constraints**.
- $\mathcal{D}$ is a set of **derivation rules**.

Redefinition of Relationship Types

Relationships are central structural elements in UML. The concept of redefinition allows enhancing the definition of a relationship by means of another relationship that defines it more specifically in a particular context [84].

The concrete syntax {`redefines end_name`} placed near an association end (the redefining end) indicates that this end redefines the one named `end_name` (the redefined end) [32, 83]. Figure 2.3 depicts a binary association *Participates* with an end *project* that is redefined by a redefining end *projectOfJunior*. In Fig 2.3(left) the redefining end *projectOfJunior* is connected to the same class as the redefined end *project* whereas in Fig 2.3(right) the redefining end *projectOfJunior* is connected to one of the descendants of that class.

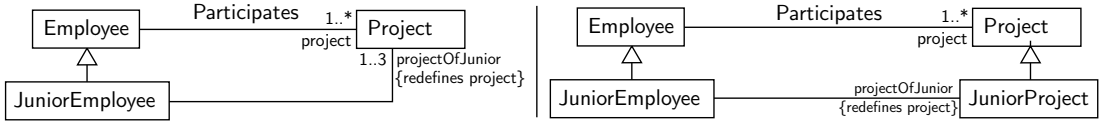


Figure 2.3. Redefinition notation in UML.

A redefinition is a *name redefinition* when the redefining end has a name different from that of the redefined end. The effect is to give a new name to the property at the redefined end for the affected instances of the redefinition. Figure 2.3 shows a name redefinition. The name *project* is redefined by the end *projectOfJunior*.

Also, a redefinition is a *type redefinition* when the redefining end is connected to a descendant of the class at the redefined end. Figure 2.3(right) shows a type redefinition. The effect is that junior employees can only participate in junior projects.

Finally, a redefinition is a multiplicity redefinition when a multiplicity is specified at the redefining end and it is more restrictive than that of the redefined end. Figure 2.3(left) shows a multiplicity redefinition. The end *projectOfJunior* redefines the multiplicity of the end *project*. The effect is that junior employees can not participate in more than three projects.

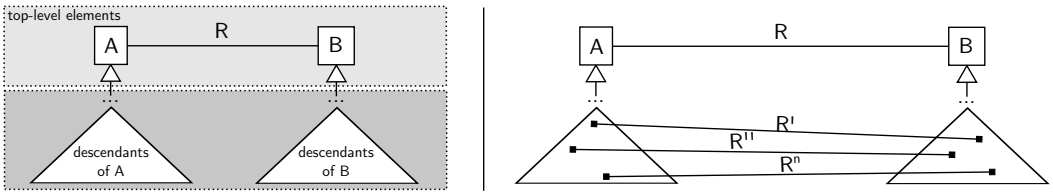


Figure 2.4. Structure of a large hierarchical schema with redefinitions.

Large conceptual schemas or ontologies may have the structure depicted in Fig. 2.4(left) where the relationships are defined between a subset of top-level elements. The rest of relationships are redefinitions of the core relationships of the top, as shown in Fig. 2.4(right). Not providing support for redefinitions may lead to poor results when extracting portions of a large schema in a filtering environment. Redefinitions include semantics that must be considered. In this thesis we include redefinitions in the set $\mathcal{R}$ of the structural subschema $\mathcal{SS}$.

2.2.2 Behavioral Subschema

The behavioral subschema of a conceptual schema contains the abstraction of the allowed changes of the state represented by the structural subschema through the definition of event types. Those events are the conceptualization of the common actions and functions of the information system represented by the complete conceptual schema.

An example of action request event type can be the action of sending a mail to customers once a new product is included in the information system. Also, an example of domain event type can be the registration of a new customer in the knowledge base of the information system as a new instance of the entity type Customer. Figure 2.5 presents the conceptualization of these entity types.

The effect of an event type must be defined with pre- and postconditions included in the schema rules component of the behavioral schema. At bottom part of Fig. 2.5 there is an example of definition in natural language of the effect and postcondition for the domain event type NewCustomer.

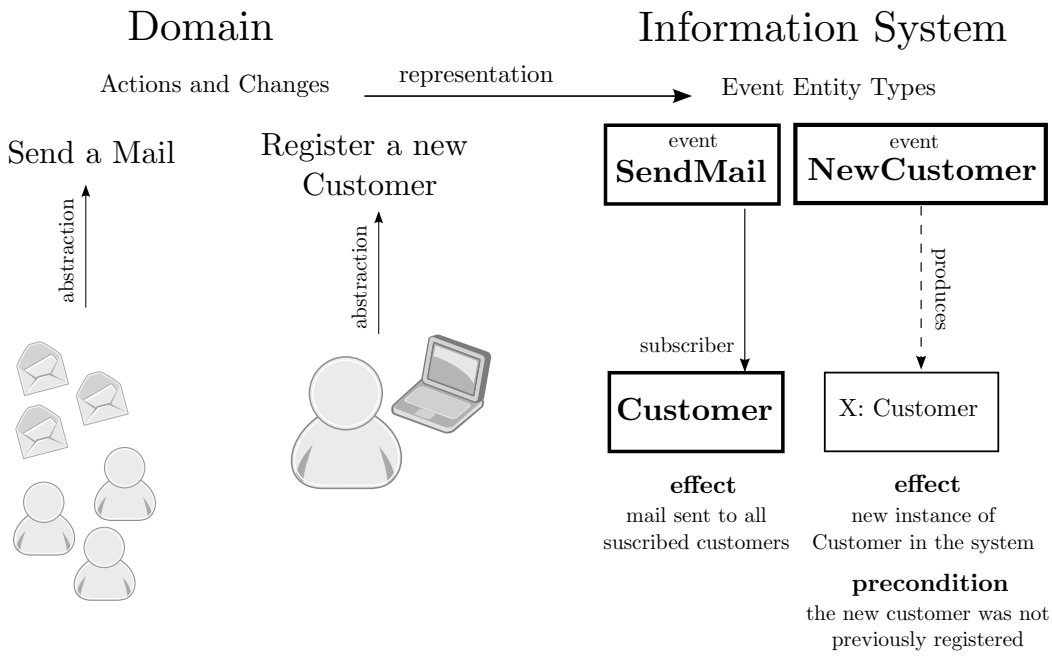


Figure 2.5. Example of event types.

The behavioral schema can contain relationship types whose participants include entity types from the structural schema related with event types from the behavioral schema. Nevertheless, definition of entity types and relationship types between entity types are only placed in the structural schema. Inheritance is also supported by event types by the definition of generalization relationships that specify binary specializations between pairs of event types —a subtype and a supertype.

The behavioral subschema $\mathcal{BS} = \langle \mathcal{E}_b, \mathcal{R}_b, \mathcal{G}_b, \mathcal{C}_b \rangle$ of a conceptual schema $\mathcal{CS}$ is formally defined as a tuple that contains the following components:

- $\mathcal{E}_b$ is a set of **event types**. Event types may define a set of owned **attributes** $\mathcal{A}_b$.
- $\mathcal{R}_b$ is a set of **relationship types**. We denote by $r(p_1{:}s_1, ..., p_n{:}s_n)$ a relationship type r with participant entity or event types $s_1, ..., s_n \in \mathcal{E} \cup \mathcal{E}_b$ playing roles $p_1, ..., p_n$ respectively. Note that the number of participants k in $r \in \mathcal{R}_b$ is the degree of r and $k > 1$. We see attributes as binary relationship types. We denote by *attr(e,b)* an attribute owned by an event type $b \in \mathcal{E}_b$, named *attr*, and whose type is $t \in \mathcal{T}$.
- $\mathcal{G}_b$ is a set of **generalization relationships**. Each $g \in \mathcal{G}_b$ represents a directed relationship between a pair of event types $(b_i \rightarrow b_j)$ where $b_i, b_j \in \mathcal{E}_b$ indicating that b_i is a direct descendant of b_j and b_j is a direct ascendant of b_i.
- $\mathcal{C}_b$ is a set of **schema rules** including invariants, and pre- and postconditions of the effect of event types of $\mathcal{E}_b$.

2.3 Modeling Languages

A modeling language is a formal language used to express information or knowledge about a domain. Modeling languages can be graphical or textual. Usually, graphical modeling languages are used to define the structure of concepts and its relationships using symbols and lines, while textual modeling languages are used to express what is not possible to express graphically like schema rules.

In the following subsections we briefly describe the de-facto standard modeling languages: the Unified Modeling Language (UML) [84] —which is a graphical modeling language— and the Object Constraint Language (OCL) [85] —which is a textual modeling language. Such languages are used to describe the conceptual schemas in the rest of this thesis.

Before the introduction of such languages, we make a short description about the entity-relationship model, which is the precursor of modern object-oriented approaches to model data and is broadly used in the literature about the topic of this thesis.

2.3.1 The Entity-Relationship Model

The Entity-Relationship (ER) model was firstly introduced by Chen in [29]. It defines a conceptual representation of data, formerly used for graphical database modeling, and introduces the concept of entity as an abstraction of some aspect of the real world that can be distinguished from other aspects of the real world. Furthermore it represents entities as rectangles and the relationships between them as diamonds, connected by lines to each of the entities in the relationship. Entities can be thought as nouns while relationships can be thought as verbs connecting two or more nouns. Finally, attributes are represented as ellipses connected to the entity or relationship that owns them.

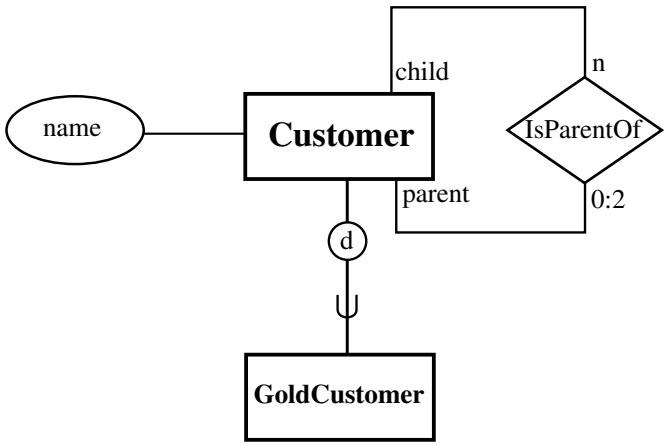


Figure 2.6. Entity-Relationship diagram about customers.

The graphical diagram containing entities, relationships and attributes is known as Entity-Relationship Diagram (or simply ERD). An example of ERD about customers explained in previous sections of the chapter can be found at Fig. 2.6.

The ER diagram notation has many variants and has evolved in the course of time. As we will see in the next chapter, many contributions in the literature about construct reduced, focused or filtered conceptual schemas are based on the ER notation and work with ER diagrams. Although our thesis focus on UML/OCL schemas, it is important to note that in the basis, both UML/OCL and ER schemas follow the same ideas and, therefore, the solutions to the problem of dealing with large and complex schemas for one of these modeling languages are valid solutions to the other type of modeling language.

2.3.2 The Unified Modeling Language

The Unified Modeling Language (UML) is a standardized general-purpose modeling language maintained by the Object Management Group(OMG).

The UML is a graphical language that contains several diagrams to specify a conceptual schema. In this thesis we will only use the class diagram of the UML, which in fact is the most used diagram, to define both the structural and behavioral subschemas of the conceptual schema. Entity and relationship types can be defined as UML classes and associations. In the case of the event types of the behavioral schema, since we model them as events it is no necessary to use a special diagram other than the class diagram to specify them.

Modeling entity types as boxes and relationship types as links between them, the example of the conceptual schema about the management of customers is shown in Fig 2.7. Note that the attributes of each entity type is placed inside the attributes compartment of the related UML class. The cardinalities and roles of the relationship types are explicitly shown in the ends of each UML association between classes.

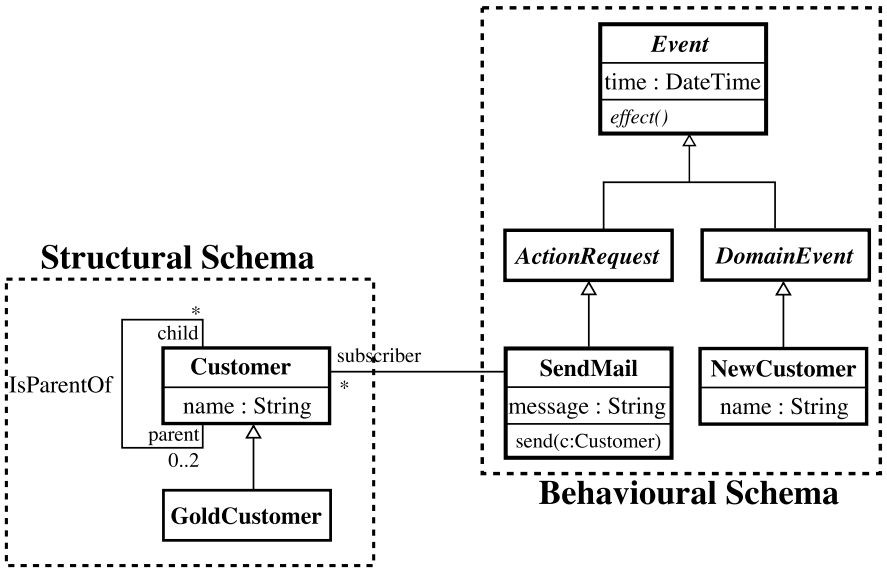


Figure 2.7. Example of conceptual schema specified in UML.

2.3.3 The Object Constraint Language

The Object Constraint Language (OCL) is a declarative language to formally describe rules in object-oriented models. The OCL is a widely accepted standard firstly introduced by IBM and now included in the UML and maintained by the Object Management Group (OMG).

The UML language provides a graphical notation based on diagrams to specify conceptual schemas as explained before. Nevertheless, invariants, derivation rules and pre- and postconditions may be expressed in natural language as comments in the diagrams of the schema. Thus, UML alone does not provide support for specifying schema rules. Since the adoption of the OCL as part of the UML, it is possible to express schema rules using formal notation through OCL expressions, constructions and statements. A full explanation (a little bit outdated) about the syntax and semantics can be found in [134]. Another (more complex) source is [85], and a review of tools supporting OCL is described in [95].

```
context Customer inv NotParentOfItself:
  self.parent->excludes(self)

context SendMail::effect()
  pre: message <> ''
  post: subscriber->forAll(c:Customer|self.send(c))

context NewCustomer::effect()
  pre: Customer.allInstances()->forAll(c:Customer|c.name <> self.name)
  post: cust.oclIsNew() and
        cust.oclIsTypeOf(Customer) and
        cust.name = self.name
```

Figure 2.8. Example of schema rules specified in OCL.

Figure 2.8 presents the schema rules of our previous example about customers, including the invariants and pre- and postconditions of both structural and behavioral subschemas. We assume that the operation *send* of SendMail returns a boolean value according to if the message was sent successfully or not, because the body of the `forAll` construction requires a boolean expression.

2.4 Summary

This chapter presented a brief introduction to the basic concepts used about conceptual modeling, conceptual schemas, and modeling languages that will be deeply studied in the next chapters. The formal definition about the two subschemas —the structural and the behavioral— contained in a complete conceptual schema and the enumeration of all of their components is the cornerstone to understand the input of our filtering methodology and the resulting output, which consists of a subset of a very large conceptual schema.

In addition to it, we present several examples of conceptual schemas and filtered schemas through the following chapters. In fact, the next chapter reviews the existing methodologies and tools to deal with large conceptual schemas. This study points out that most of the approaches in the literature work with entity-relationship diagrams. Nowadays, most of the modeling activities that are meant to construct conceptual schemas use the UML/OCL as the *de-facto* standard modeling languages. Consequently, we believe that our filtering approach, which is truly based on UML/OCL schemas —although it can be easily adapted to other schemas— faces a relevant topic that needs of further research.

640K ought to be enough
for anybody

Bill Gates (1981)

3

Conceptual Modeling in the Large

A conceptual schema defines the general knowledge about the domain that an information system needs to know to perform its functions. The conceptual schema of many real-world information systems and the ontologies of broad or general domains are too large to be easily managed or understood. In general, those conceptual schemas include several kinds of elements such as entity and relationship types, attributes, generalization relationships, event types, and many formal constraint expressions (also called schema rules), which are used for defining static or dynamic integrity constraints, derivation rules, default values, pre and postconditions of events and operations, or results of operations.

There are many information system development activities in which people needs to get a piece of the knowledge contained in the conceptual schema. For example, a conceptual modeler needs to check with a domain expert that the knowledge is correct, a database designer needs to implement that knowledge into a relational database, a software tester needs to write tests checking that the knowledge has been correctly implemented in the system components, or a member of the maintenance team needs to change that knowledge.

The largeness of conceptual schemas makes it difficult for a user to get the knowledge of interest to her. This chapter studies the different approaches and methods in the literature that deal with large conceptual schemas. Section 3.1 introduces the problem of manually extracting information from large schemas and its relation to the human capacity for processing information. Section 3.2 presents several requests for contributions in this area, and Sect. 3.3 enumerates and describes the existing proposals to improve end-user understanding of large schemas. Finally, Sect. 3.4 describes the filtering approach as an alternative to existing approaches, Sect. 3.5 compares the aforementioned proposals, and Sect. 3.6 summarizes the chapter.

3.1 Dealing with Large Conceptual Schemas

Nowadays, the need for representation and conceptualization of real world information has dramatically increased. Organizations evolution and diversification require the management and maintenance of large amounts of knowledge from their domains of interest. Also, data mining and knowledge extraction are becoming trending topics in business processes.

Real information systems often have extremely complex conceptual schemas. The visualization and understanding of these schemas require the use of specific methods, which are not needed in small schemas. That complexity also has an impact in the size of conceptual schemas of information systems, making them larger. The sheer size of those schemas transforms them into very useful artifacts for the communities and organizations for which they are developed. They are not only becoming a key artifact in software engineering, but also a conceptual representation of the whole *know-how* of the organization and information system they represent. However, the size of the schemas and their overall structure and organization make it difficult to manually extract knowledge from them, to understand their characteristics, and to change them.

To provide a conceptual schema of reduced size with the most relevant knowledge highlighted implies a rise of knowledge accessibility that benefits the understandability of the schema. Otherwise we have a case of information overload.

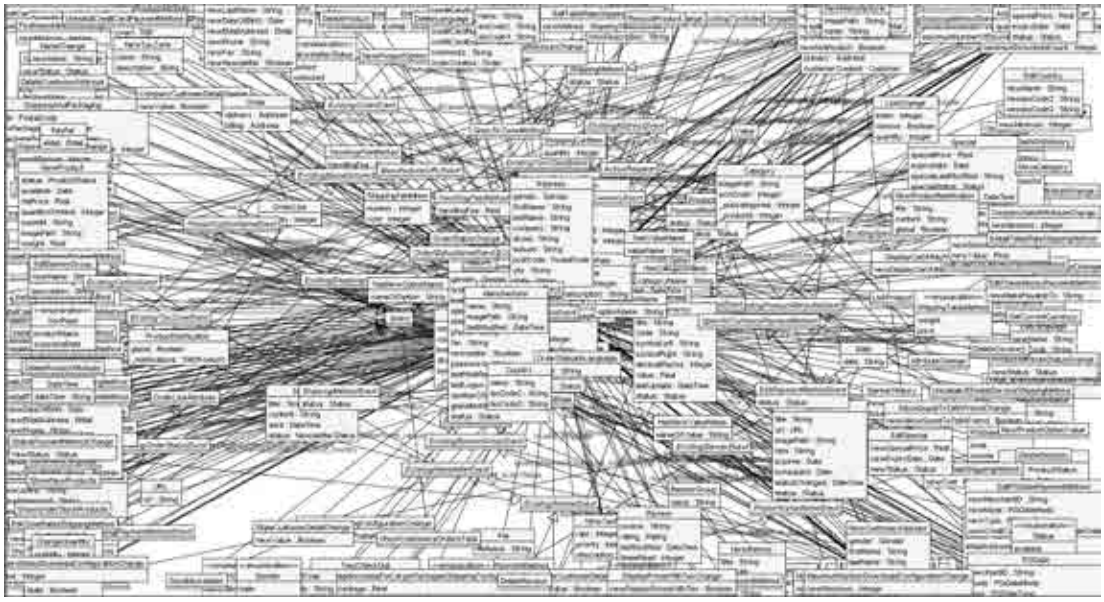


Figure 3.1. Conceptual Schema of the osCommerce system.

Figure 3.1 shows the conceptual schema of the osCommerce[1], an online shop e-commerce solution that offers a wide range of out-of-the-box features that allows online stores to be setup fairly quickly with ease, and is available for free as an Open Source based solution released

[1] `http://www.oscommerce.com`

under the GNU General Public License. In the same way, Fig. 3.2 presents the conceptual schema of the OpenCyc[2] ontology, the open source version of the Cyc technology, one of the largest and most complete general knowledge base. The osCommerce schema [118] contains over 350 entity types and 200 relationship types, including the specification of 260 event types that deal with the behavioral aspects of online stores. On the other hand, the OpenCyc [31] includes over 3000 entity types and 2700 relationship types.

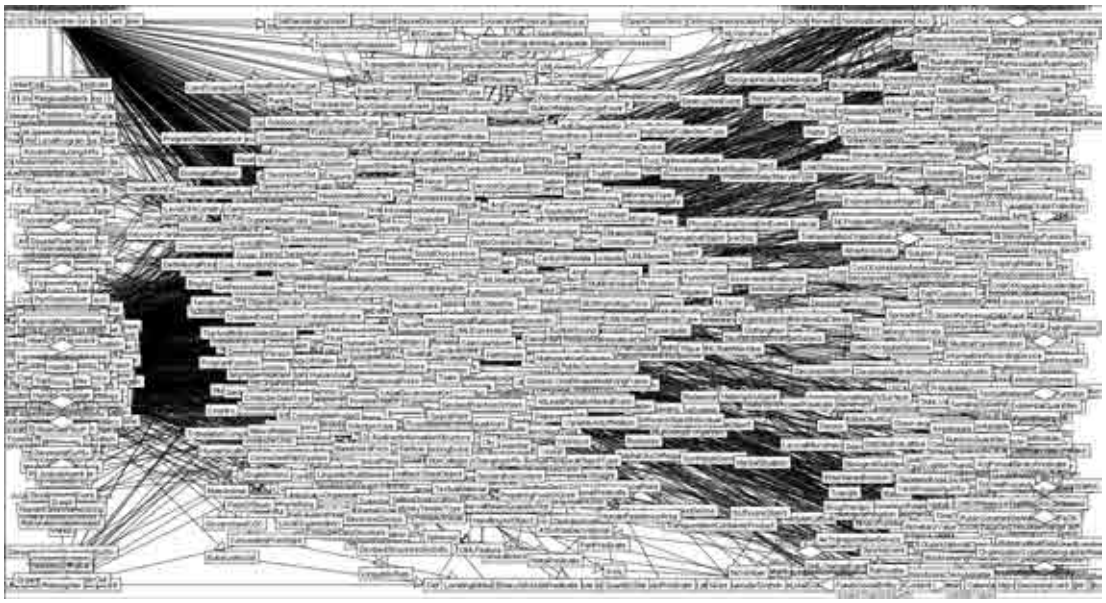


Figure 3.2. Conceptual Schema of the OpenCyc ontology.

Both are examples of large conceptual schemas containing a huge amount of information about a domain, that can be general as in the case of the OpenCyc. The required user effort to understand and work with this kind of schemas is unacceptable, and consequently a contribution in this field is absolutely necessary.

It is clear that to be useful, large containers of information, as large conceptual schemas are, need of tools that can extract the contained portion of knowledge of interest to a particular user at a time. That situation already happened with the World Wide Web. Until the uprising of web search engines, the knowledge that the web contained was only partially accessible due to its huge size and difficulty of finding what was sought. After that, the web has become into something useful and really easy to use due to the existing tool support. Nowadays not only experts can search the web, but also everybody with a computer and an information need is capable of that.

At present, conceptual schemas are gaining more presence in the software engineering field and beyond. Our proposal aims to contribute to the expansion of conceptual schemas by the study of its characteristics and the description of the structure and components of a filtering engine for large conceptual schemas. With our work, we expect to facilitate the use of conceptual schemas to those interested in their knowledge.

[2] `http://www.cyc.com/opencyc`

3.1.1 Human Capacity for Processing Information

The human brain is one of the most complex tools we have. As Marois and Ivanoff indicates in [75], its hundred billion neurons and several hundred trillion synaptic connections can process and exchange prodigious amounts of information over a distributed neural network in the matter of milliseconds.

However, the human capacity for processing unknown information (see Fig. 3.3) is very limited. It contains bottlenecks in visual short-term memory and causes problems to identify and held stimuli. Miller in [77] states that the limit on our capacity for processing information is contained in a range of seven, plus or minus two, items or *chunks* of information. Of course, a differentiation must be done between short- and long-term memory. Cowan explains in [33] that such memories differs in properties like timing of memory activation, coding and control, and storage capacity. Although long-term memory has a large storage capacity, its activation is slower and requires more attention time than short-term memory.

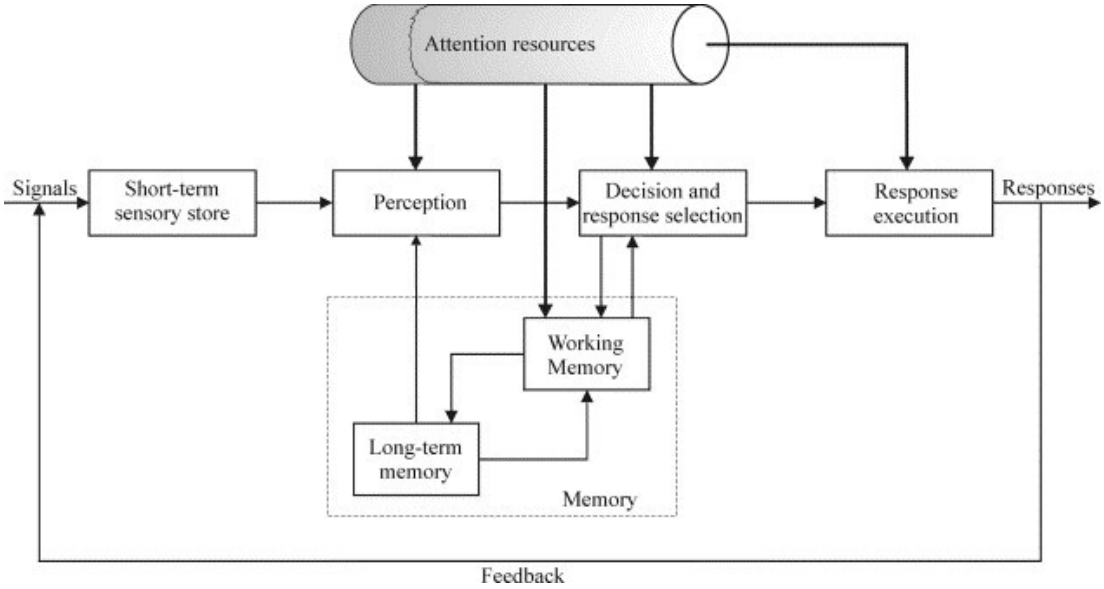


Figure 3.3. Human information processing. Extracted from [66].

We can say that human capacity for processing information has similar characteristics than a computer. To have a little number of stimuli is like to have a little amount of instructions to process. On the other hand, to deal with big amounts of stimuli saturates our brain as big amounts of instructions do with a computer processor. Of course, there exists some techniques to solve these problems, like parallel computing or to have several replications of core processors. Unfortunately, such solutions are not directly applicable to human beings.

As we cannot replicate our brains or (generally) parallelize tasks, our purpose is to cut down the amount of information to process. Some solutions in this are are presented to reduce bottlenecks in human capacity for information processing, like clustering, that consists on group items according to some sort of similarity, or filtering, that hides irrelevant information, increas-

ing the attention to important items. Such methods are highly recommendable in conceptual modeling to improve accessibility and comprehension of large conceptual schemas by reducing or changing the structure of their information.

3.1.2 Information Extraction

Nowadays information retrieval and, in a broader sense data mining, have become two of the most important disciplines to deal with large amounts of information. These subjects provide a set of different methods to extract knowledge from data. In-deep information about such topics can be found in [97] and [98]. More recent sources about information retrieval and data mining are [5] and [54], respectively.

Information retrieval was initially centered in searching information within documents until the birth of the web. As the information located in the Internet became bigger, increasingly diffuse, and complex to manage, information retrieval got a major prominence. One of the main contributions of information retrieval was the appearance of the web searchers.

On the other hand, data mining has a broader scope of application. Some of its methods are used in statistics, programming, and specially in data bases. As we will study in the following, literature contributions about reducing and filtering schemas have principally centered on database schemas. Along this thesis, some of the main techniques in this area will be used to compute the importance of schema elements (mainly entity types). Principally, link analysis and occurrence counting are the focus of the work described in Ch. 4. The relevance of schema elements is the basis of our filtering engine.

Link analysis studies the edges of a connected graph to provide a ranking of those nodes that are more important according to its inner and outer connections through edges. This method is recursively defined and needs iterative algorithms to solve the problem. That is because the importance of a node comes from the other nodes that point to it and, therefore, such importance flows to the nodes pointed by the node in question. This importance propagation must be computed in an equilibrium point where the graph nodes are balanced. Principally, such approach was introduced by Brin and Page in [21] as the foundation of Google's PageRank algorithm to compute the relevance of web pages.

Alternatively, occurrence counting is a basic technique that consists on counting how many times an element appears on a situation. It was centered on word occurrences in texts to discover the most/less used words or to state similarities between documents. As we will see in the next chapters, we apply this idea to conceptual schemas by counting the number of occurrences of schema elements in different contexts.

Figure 3.4 presents an example of occurrence counting that contains a tag cloud with the top-150 words included in the superstructure specification document of the UML 2 modeling language (see [84]). As the reader can see, a tag cloud is a cloud of words where the words with a greater number of occurrences have a greater size than the others. Without reading the UML 2 specification document, it is possible to say that probably such document contains information about an specification, a superstructure, states, actions, elements, types, objects,

Figure 3.4. Tag cloud of the superstructure specification of the UML 2 [84].

classifiers, attributes, associations, constraints and, specially, about the UML. This way, to decide whether two documents are similar or not is simpler and more efficient than to read them. Following the same idea, we can affirm that the words with more occurrences are the most important. This approach will be the main we follow in the application of occurrence counting to compute the importance of schema elements for our filtering engine.

3.2 Requests for Contributions

In 1975, DeRemer and Kron [36] distinguished the activity of writing large programs from that of writing small ones. By large programs they meant systems consisting of many small programs (modules), possibly written by different people. They were aware of the need for knitting those modules together into an integrated whole and for providing an overview that formally records the intent of the programmer(s) and that can be checked for consistency by a compiler. That was what they called *programming in the large*.

Following the ideas of the previous work, Bézivin et al. [15, 14] believe that the situation in the modeling area today is quite similar to the situation described at that time in the programming area. Starting from this similarity, they distinguish the two related activities of *modeling in the large* and *modeling in the small*. Both activities are different and require of a differentiated analysis and tool-support.

In their research agenda for Conceptual Schema-Centric Development (CSCD) [88], Olivé and Cabot describe that we need methods, techniques and tools to support designers and users in the development, reuse, evolution, and understanding of large schemas. They dedicate a section of their agenda to explain that the development of large conceptual schemas pose specific problems not found in small conceptual schemas. They also indicate that work on this topic has focused mainly on conceptual schemas for databases and, as a conclusion, it is

important to deal with modelizations of information systems and to take into account both the structural and behavioral schemas (including constraints and derivation rules). As explained in previous chapters, our work will follow these indications in order to compute some measures about schema elements.

Another request for contributions in this area is found in the study of Lindland, Sindre and Sølvberg about quality in conceptual modeling [72]. They classify filtering as a modeling activity to improve the comprehension goal and model properties like structuredness, executability and expressive economy. Concretely, they view filtering as a necessary activity because a person cannot grasp the entire schema at once and must concentrate on specific parts or aspects at a time. They claim that filtering may also include aspects of translation because a large schema may have a rather diverse audience. Therefore, different languages will be preferred by various groups. While end users may want to see business rules in natural language, analysts may want to see them in logic. This way, in addition to filtering they indicate that different views using different languages should be made in order to simplify the understandability of all kind of users of the conceptual schema.

Finally, Papazoglou indicates in [92] that to improve the utility of large and complex information systems, we need to make schema interfaces more perceptive, responsive, and efficient for all categories of users including casual users. The author also requests the use of schema semantics instead of only the structure of the schema to reach this goal.

3.3 Major Contributions

There exists some alternative methods to process raw conceptual schemas and produce the desired output consisting in a simpler version of the input. These methods generate indexed, clustered, filtered, summarized or focused schemas that are easier to visualize and to understand. In this section we will review some solutions proposed to solve the problem of dealing with the knowledge of large conceptual schemas in the conceptual modeling area.

3.3.1 Clustering Methods

Clustering can be defined as the activity of grouping elements according to a similarity function. Therefore, similar elements will be put together in the same group, or *cluster*.

There exists a huge amount of contributions in the literature about clustering of schemas, ontologies or, definitely, graphs. Estivill-Castro wonders in [42] why the existence of so many clustering algorithms. The answer here is clear: there are many clustering algorithms because there are many algorithms for each inductive principle and there are many inductive principles to solve the same problem. The author explains that clustering is in part in the eye of the beholder, meaning that every researcher can propose his own similarity function to approximate a solution. Because clustering is an optimization problem, the number of approximated solutions closer to the best solution is huge. In this section we will review some solutions proposed to solve the problem of clustering the elements of large conceptual schemas.

The paper from Feldman and Miller [43] is one of the foundational papers in the area. They explain the technique called entity model clustering and state that entity relationship diagrams can be manipulated in various ways to be more easily understood.

One of the problems the authors define is that the usefulness of any diagram is inversely proportional to the size of the model depicted. They consider any diagram with more than about 30 entity types to be reaching the limits of easy comprehension, depending on the number of relationships –the more relationships, the less comprehension is possible due to the accompanying increase in complexity. Therefore, it is possible to say that the two main problems of large conceptual schemas are about size and complexity.

The entity model clustering technique proposed by Feldman and Miller results in a decomposition of the diagram in a tree with three levels of abstraction. Some entity types are allowed to be duplicate in some branches of the tree according to the authors' experience. Although the number of levels is determined by the diversity and complexity of an organization, they state that in practice three levels of diagram have been found to be useful. The hierarchy of successively more detailed entity relationship diagrams is shown in Fig. 3.5.

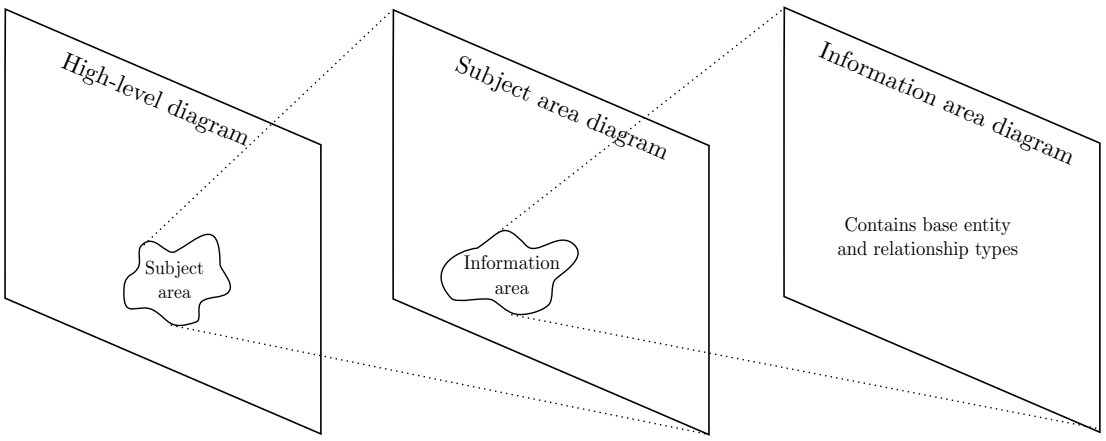


Figure 3.5. Three levels of abstraction diagramming. Inspired by [43].

First step consists on finding the major entity types. Occurrences of a major entity type should be uniquely identifiable from any related entity types. Furthermore, a major entity type should be of fundamental importance to more than one functional area of organization, i.e. should appear in more than one information area. The aforementioned concept of occurrence counting is also present in this approach.

Next step is to detect subject areas of the higher level. Subject areas and their information areas can be thought of as decompositions of the relationships between major entity types. Information areas are formed by first abstracting minor entities into a logical horizon and then successively abstracting the logical horizons and majority entity types. This process actually results in more than one information area relating to the same group of major entity types. These similar information areas are then abstracted to form a subject area, which is placed in the highest level.

Finally, Feldman and Miller classify the benefits of entity model clustering in:

- **Organizational benefits:** levels of diagrams are similar to levels in the organization.
- **End-user computing benefits:** they do not need to have to know of the existence of an entity type, but can be led to it through the succeeding levels of detail of the clustered entity model.
- **Information system development benefits:** development activities can be built without the complexity associated to large models.
- **Entity modeling benefits:** very large models become easy to communicate, validate and maintain.

Another foundational paper is the one from Teorey et al. [116]. Their approach is similar to that of Feldman and Miller's, but it contains some differences. The clustering method does not allow duplicate entities in different levels of the clustering hierarchy and the number of levels is not predefined. Roughly, the clustering technique from Teorey et al. follows the next steps:

1. **Define points of grouping within functional areas:** locate the dominant entities in a functional area, either through the natural relationships as obtained from the system requirements document, local n-ary relationships, integrity constraints, abstractions, or by just being the central focus of many simple relationships.
2. **Form entity clusters:** use basic grouping operations on elementary entities and their relationships to form higher-level objects, entity clusters.
3. **Form higher-level entity clusters:** apply the grouping operations recursively to any combination of elementary entities and entity clusters to form new levels of entity clusters (higher level objects).
4. **Validate the cluster diagram:** check for consistency of the interfaces (relationships) between objects at each level diagram. Verify the meaning of each level with the end users.

The result of applying the previous steps to a large entity-relationship diagram is shown in Fig. 3.6. The method of Teorey et al. may be adapted to other modeling languages such as UML. Shoval, Danoch and Balabam in [104, 105] proposed a revision of the approach of Teorey et al., previously defined. They call their new solution HERD (Hierarchical Entity-Relationship Diagrams) and basically includes minimum changes in grouping operations and their application.

Another different contribution was made by Jaeschke, Oberweis and Stucky [61]. The authors propose an approach to entity model clustering to allow top-down design. The main idea is to determine the major entity types and the coarse relationship types between them. Then these relationship types are refined iteratively by complex and simple relationship clustering, also involving entity clustering. After determining the major entity types, the detailed design of

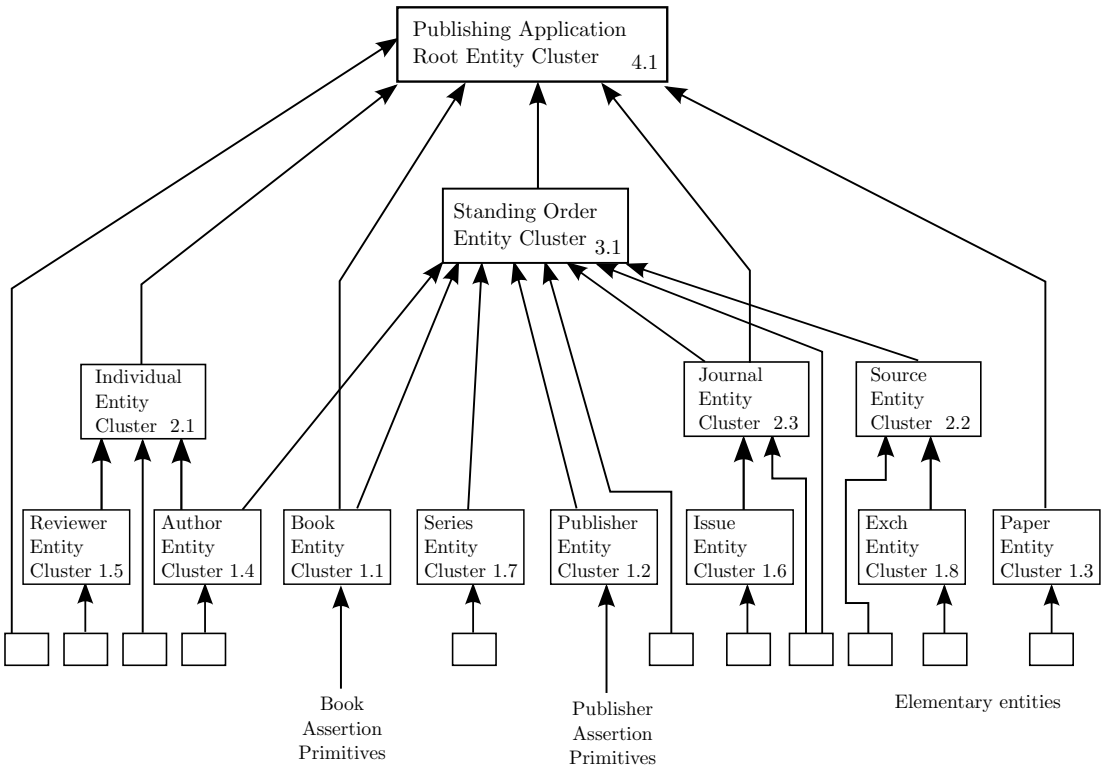


Figure 3.6. Example of Entity Cluster Levels. Extracted from [116].

the different relationship clusters can be realized simultaneously and independently by different project groups. This approach also supports database re-engineering. The clusters can be built bottom-up based on already existing database schemes while the redesign is realized top down. Roughly, the process consists on define high-level objects at first and, after that, redefine the relationships between them to complete the schema.

Francalanci and Pernici in [46] discuss the problem of the semi-automatic construction of abstract entity-relationship schemas and propose and algorithm for schema clustering, mainly based on the structure of the schema. Furthermore, they define *affinity* and *closeness* between concepts and *coupling* and *cohesion* between clusters as the operating criteria for clustering. *Affinity* captures semantic closeness among concepts, *closeness* corresponds to a quantitative evaluation of the links among concepts, *cohesion* corresponds to a value indicating how well internally connected are the concepts within clusters, and *coupling* corresponds to a value indicating the amount of connections between different clusters.

Next contribution that deserves a mention is Akoka and Comyn-Wattiau's paper [1] on entity-relationship and object-oriented automatic clustering. There, the authors propose a common clustering algorithm and a set of similarity functions that they call distances between elements. They define such distances and apply them in their algorithm. A short review of the distances, including object-oriented distances, is shown in the following list:

- **Visual distance:** two entities are said to be close if they are involved in the same relationship.
- **Hierarchical distance:** the distance between two entities is measured by the shortest relationship path between entities. The cardinalities of relationships are included in the computation of such distance.
- **Cohesive distance:** as with the preceding distance, the cohesive distance is measured by the shortest path between those entities. However, it is considered a different segment length and weight on such paths.
- **Structural-connective distance:** two elements are close if they are neighbors in a hierarchy (aggregation, generalization, whole-part structure). They are close if they are linked by an instance connection or a message connection. Otherwise the distance between two objects is the length of the shortest path between them.
- **Category distance:** two elements are considered to be very close if there exists a generalization relationship between them.
- **Communicative distance:** the communication between two objects expresses a semantic link between these objects. Therefore the communicative distance is based on message flowing.
- **Frequent communicative distance:** the message frequency is the number of times a message is flowing between objects for a given period of time. As a consequence, two objects are considered to be close when this frequency is high.

For their part, Campbell, Halpin and Proper formalize in [24] a method for the strictly automatic selection of major entity (or object) types. Their approach sets apart from others because it considers the detailed conceptual semantics hidden in the constraints and also the manner in which the facts within the domain are verbalized. In particular, their approach utilizes the detailed constraint specifications and verbalizations provided by Object Role Modeling (a modeling technique that allows a variety of data constraints to be specified on the conceptual schema).

The semantics of these constraints allow to make the selection of major objects. The authors claim that their approach more accurately imitates human intuition than previous methods. As a second goal, the paper utilize the selected major object types in an algorithm to derive abstractions for a flat conceptual schema.

Moody and Flitman in [80, 78] introduce the concept of Levelled Data Model, i.e. a data model organized into any number of levels, depending on the size. They propose a clustering algorithm to be applied to ER models. The authors also collects the major deficiencies identified in the existing literature:

- **Lack of cognitive justification:** to be truly effective in improving human understanding, clustering approaches need to be soundly based on principles of human information processing.

- **Lack of size constraints:** the aim of all existing methods is to reduce the model to parts of manageable size, but none of them define what this is.
- **Lack of automation:** only some approaches provide automated solutions to the problem.
- **Levels of decomposition:** most of the approaches proposed are limited to a fixed number of levels of decomposition.
- **Lack of empirical testing:** it is stated and argued by the authors that their methods provide an effective solution to the problem, but these claims are unsubstantiated.

Such deficiencies are not exclusive from clustering. They are shared between other kind of methods in the literature.

Another good contribution by Moody and Flitman is the definition of formal rules or principles for evaluating the quality of decompositions and choosing between alternatives. These principles are briefly reviewed in the following list:

- **Completeness:** each entity type must be assigned to at least one subject area. The union of the subject areas cover all the entities in the underlying model.
- **Non-Redundancy:** each entity type must be assigned to at most one subject area. This ensures that subject areas are disjoint.
- **Integration:** each subject area forms a fully connected subgraph of the original model.
- **Unity:** each subject area should be named after one of the entities on the subject area, called the central entity. The central entity forms the core of the subject area.
- **Cognitively Manageable:** each subject are must be of cognitively manageable size. It means a maximum of nine concepts.
- **Flexibility:** the partitioning of the data model into subject areas should allow adequate capacity for growth. A data model which consists of subject areas that are all of the size of nine will have to be repartitioned if even a single entity is added. To solve this situation, it is required that the average size of subject areas is as close as possible to seven entities.
- **Equal abstraction:** all subject areas should be similar in size.
- **Coupling:** the number of relationships between entities from different subject areas should be minimum.
- **Cohesion:** the number of relationships between entities on the same subject area should be maximum.

To conclude, Tavana et al. study in [115] the decomposition principles of Moody and Flitman previously described. The authors introduce a clustering algorithm that adopts some concepts and metrics from machine-part clustering in cellular manufacturing while exploiting some of the

characteristics of ER diagrams. The aim of the algorithm is to follow the principles of Moody and Flitman and, mainly, to reduce coupling and increase cohesion. In fact, the authors made a comparison between their algorithm and the solutions proposed by Feldman and Miller [43], and Moody and Flitman [80, 78]. In both cases, they claim that the results obtained by their algorithm are better than others. An example of resulting clustered entity relationship diagram from the application of such algorithm is shown in Fig. 3.7.

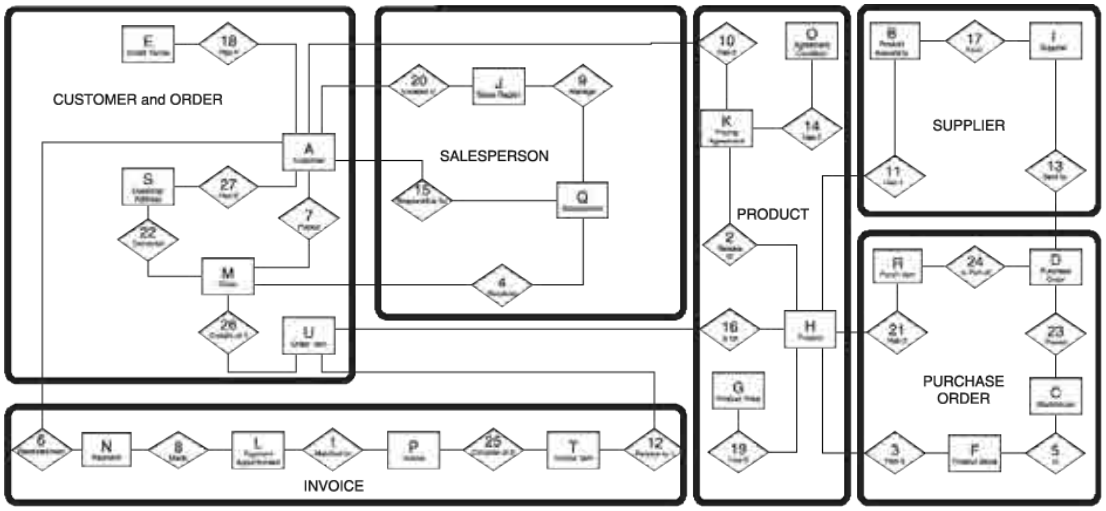


Figure 3.7. Application of clustering algorithm of Tavana et al.. Extracted from [115].

Table 3.1 summarizes the previous clustering contributions.

Table 3.1. Summary of clustering-based contributions.

Contribution	Characteristics
Feldman and Miller [43]	3-level hierarchichal clustering of entities of ER diagrams. Group entities in subject areas (bottom-up approach). Form higher-level entity clusters grouping subject areas. Duplication of entities in different subject areas is allowed.
Teorey et al. [116]	n-level hierarchichal clustering of entities of ER diagrams. Group entities in functional areas (bottom-up approach). Form higher-level entity clusters grouping functional areas. Duplication of entities in different subject areas is not allowed.
Shoval et al. [104, 105]	HERD: Hierarchical Entity-Relationship Diagrams. n-level hierarchichal clustering of entities of ER diagrams. Group entities through dominance grouping, accumulation, and abstraction grouping (bottom-up approach). Duplication of entities in different clusters is not allowed. Experimental comparison of HERD diagrams and ER diagrams.

Table 3.1. Summary of clustering-based contributions (continued).

Contribution	Characteristics
Jaeschke et al. [61]	Clustering of entities and relationships in ER diagrams. Focus on relationship clustering. Define high-level entities and relationships at first, then redefine them iteratively (top-down approach).
Francalanci and Pernici [46]	Semi-automatic construction of clustered ER diagrams. Clustering based on metrics over the structure of the diagram. Definition of *affinity* and closeness between entities. Definition of *coupling* and cohesion between clusters.
Akoka and Comyn-Wattiau [1]	Automatization of conceptual schema clustering leading to a unification of past approaches. Definition of three different distances (visual, hierarchical and cohesive) depending on the semantic richness of the schema. Object model clustering is based on structural, semantic and communication characteristics of objects.
Campbell et al. [24]	A method for the strictly automatic selection of major entity (or object) types. Considers the detailed conceptual semantics hidden in constraints. Utilize the selected major object types in an algorithm to derive clusters.
Moody and Flitman [80, 78]	Levelled Data Model (n-level hierachichal clustering of ER models). Collection and classification of the major deficiencies identified in the existing literature. Definition of formal rules or principles for evaluating the quality of decompositions and choosing between alternatives.
Tavana [115]	Application of some concepts and metrics from machine-part clustering in cellular manufacturing to ER diagrams. The aim is to follow the principles of Moody and Flitman and to reduce coupling and increase cohesion.

3.3.2 Relevance Methods

In contrast to clustering of conceptual schemas, the number of research contributions on relevance (also known as scoring or ranking) methods applied to conceptual schemas have been clearly lower. In this section we review some of the most important approaches in this area.

One of the first works on conceptual schema analysis was done by Castano, de Antonellis, Fugini and Pernici in [25]. Castano et al. state that the representative elements of a schema are its most relevant elements, that is, describing the purpose of the schema and its basic characteristics. Representative elements are determined on the basis of a *relevance* measure within the

schema. To compute the relevance of an element, they take into account the properties of the element and the links in which it participates. The rationale is that the number of properties and links in which the element participates can be used as a (heuristic) measure of its relevance within the schema. The greater this measure, the higher the relevance of the element, since this means that the element is characterized by several properties and is referred to by several other elements of the schema.

The method of Castano et al. only considers a small part of the structural subschema of a conceptual schema containing the entity types, their attributes, and the relationships between entity types (both association and generalization relationships). Roughly, for each entity type, its relevance is computed as the addition of the number of owned attributes plus the number of relationships in which the entity participates. In fact, each kind of characteristic (attribute, association, generalization) is weighted with a strength factor to denote a difference of the contribution of such kind in the final relevance value. The authors choose to give a higher strength to attributes, then generalizations and then association relationships. This is due to the fact that generalization/specialization links between entities in a hierarchy express a high connection between entity and its specializations, whereas relationships represent a weaker link, from the semantic point of view.

Fig. 3.8 presents an example of schema where the bold squares are the representative elements computed according to the previous method.

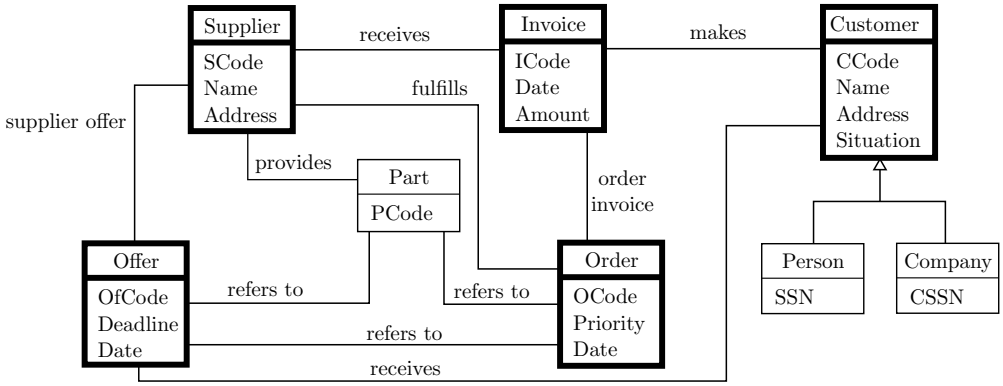


Figure 3.8. Representative elements of a conceptual schema. Extracted from [25].

Also, Geerts, Mannila and Terzi adapt in [48] link analysis algorithms to relational databases. In analogy to Web-search engines, which use the Web graph to rank web pages, they use the database graph to rank partial tuples. To obtain rankings for partial tuples they mimic the principles of link analysis algorithms.

The well-studied algorithms for the Web show that the structure of the interconnections of web pages has lots of valuable information. For example, Kleinberg's HITS algorithm [67] suggests that each page should have a separate *authority* rating (based on the links going to the page) and *hub* rating (based on the links going from the page). The intuition behind the algorithm is that important hubs have links to important authorities and important authorities are linked by important hubs.

Brin and Page's PageRank algorithm [21], on the other hand, globally calculates the PageRank of a web page by considering a random walk on the Web graph and computing its stationary distribution. The PageRank algorithm can also be seen as a model of a user's behavior where a hypothetical web surfer clicks on hyperlinks at random with no regard towards content. More specifically, when the random surfer is on a web page, the probability that he clicks on one hyperlink of the page depends solely on the number of outgoing links the latter has. However, sometimes the surfer gets bored and jumps to a random web page on the Web. The PageRank of a web page is the expected number of times the random surfer visits that page if he would click infinitely many times. Important web pages are ones which are visited very often by the random surfer.

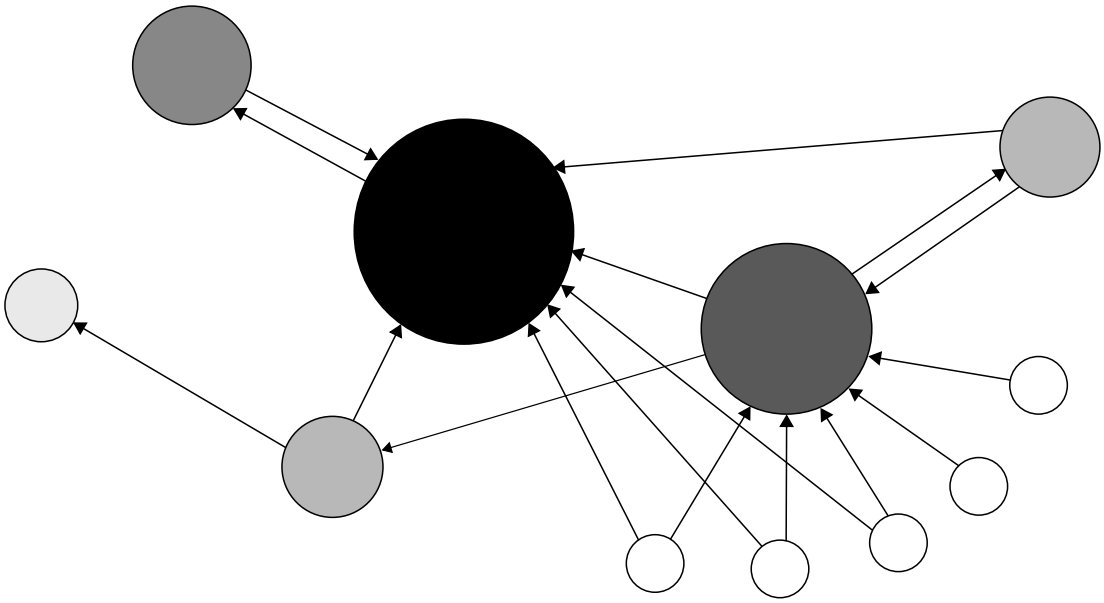


Figure 3.9. Example of PageRank.

For instance, in Fig. 3.9 size means relevance. It is easy to see that bigger (most relevant) circle is the most pointed (or linked) one, and that the circles linked by it get part of its importance becoming also big. That is due to the relevance flowing that link analysis algorithms produce according to their recursive definition: the relevance of an element is the addition of a proportional portion of the relevance of the elements that point to it. It is important to note that to execute such link analysis algorithms an iterative method is needed because of the recursive definition.

The same approach can be used to rank entity types in a conceptual schema. Concretely, Tzitzikas and Hainaut propose in [120] two PageRank-style algorithms, called EntityRank and BEntityRank, to obtain a rank of entity types according to their relevance in entity-relationship diagrams. The same algorithms, among others, are also included in the paper by Tzitzikas, Kotzinos and Theoaris [121], but applied to RDF schemas. Hsi et al. present a similar set of methods in [60].

These two variants take into account only some structural elements of the conceptual schema. Concretely, to calculate the relevance of an entity type they use the relationships (without make a differentiation between generalization or association relationships). BEntityRank algorithm also assumes that the initial relevance of entity types is the number of attributes they own. After every iteration of the algorithms, the relevance gets closer to the real one due to the relevance transfers the algorithms produce through the relationships between entities. EntityRank and BEntityRank are used to obtain the top-k entities in entity-relationship diagrams. Concretely, such algorithms produce a ranking that then is processed to filter those entities that are not the k most important ones. The example of Fig. 3.10 shows a complete entity-relationship diagram (Fig. 3.10(a)) and the top-5 diagram (Fig. 3.10(b)) after the application of one of these algorithms.

Table 3.2 summarizes the previous relevance-based contributions.

Table 3.2. Summary of relevance-based contributions.

Contribution	**Characteristics**
Castano et al. [25]	The representative elements of a UML schema are its most relevant elements. The relevance of an entity type is computed as the addition of the number of owned attributes plus the number of relationships in which the entity participates Each kind of characteristic (attribute, association, generalization) is weighted with a strength factor. Higher strength to attributes, then generalizations and then association relationships.
Geerts et al. [48]	Adaptation of link analysis algorithms to relational databases. Use the database graph to rank partial tuples of a database.
Kleinberg [67]	Definition of *authority* rating (based on the links going to the page) and *hub* rating (based on the links going from the page) for web pages. Intuition: hubs have links to important authorities and important authorities are linked by important hubs.
Brin and Page [21]	PageRank algorithm to web pages. The PageRank of a web page is the expected number of times the random surfer visits that page if he would click infinitely many times. The relevance of an element is the addition of a proportional portion of the relevance of the elements that point to it.
Tzitzikas et al. [120, 121]	2 PageRank-style algorithms (EntityRank and BEntityRank) to obtain a rank of entity types according to their relevance in ER diagrams. Use the relationships as links (without make a differentiation between generalizations or associations). BEntityRank algorithm assumes that the initial relevance of entity types is the number of attributes they own.

Table 3.2. Summary of relevance-based contributions (continued).

Contribution	Characteristics
Hsi et al. [60]	Definition of 5 metrics to compute the core concepts of an ontology. Structural metrics based on the number of attributes and relationships of a concept. Distance metrics based on the closeness of concepts. Link analysis metrics based on computing the relevance of a concept as the addition of a portion of the relevance of its surrounding concepts.

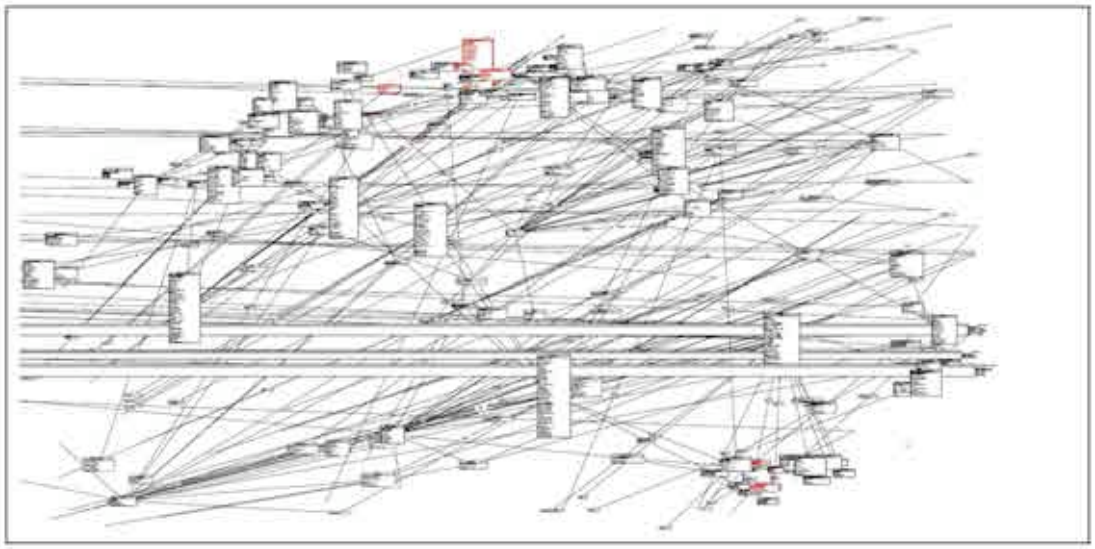
3.3.3 Summarization Methods

A schema summary uses abstract elements and abstract links to summarize a complex schema and provide the users with a concise overview for better understanding. While schema summaries are useful, creating a good summary is a non-trivial task. A schema summary should be concise enough for users to comprehend, yet it needs to convey enough information for users to obtain a decent understanding of the underlying schema and data.

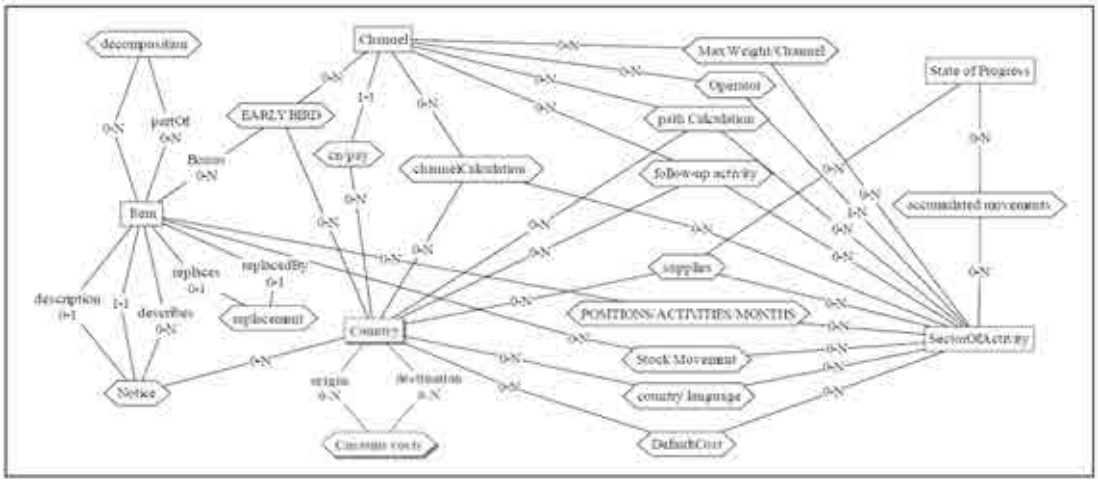
Yu and Jagadish [140] formally defines the concept of schema summary and present two desirable properties (in addition to minimizing size) of a summary: present important schema elements and achieve broad information coverage. Their method to obtain schema summaries is applied to database schemas. Each abstract element in the summary corresponds to a cluster of original schema elements (and other lower level abstract elements in the case of a multi-level summary), and each abstract link represents one or more links between the schema elements within those abstract elements.

Therefore, the importance of a schema element the authors define is reflected in two aspects —its connectivity in the schema and its cardinality in the database. The connectivity of an element in the schema graph provides a count of the number of other elements that are directly connected to it (via either relationship links). An important element is likely to be one from which many other elements can be reached easily. The cardinality of a schema element is the number of data nodes (database tuples) it corresponds to. If there are many data nodes of a schema element in the database, then that element is likely to be of greater importance than another one with very few data nodes.

Alternatively, Yang et al. [139] continue the previous work of Yu and Jagadish by defining the importance of each table in the database as its stable state value in a random walk over the database schema graph, where the transition probabilities depend on the entropy of table attributes. This ensures that the importance of a table depends both on its information content, and on how that content relates to the content of other tables in the database. They propose an algorithm under an importance function to cluster all tables in the database around the most relevant tables, and return the result as the summary. The authors conduct an extensive experimental study on a benchmark database, comparing their approach with the one by Yu



(a) Full ER diagram



(b) Top-5 entity types

Figure 3.10. EntityRank application. Extracted from [120].

and Jagadish, as well as with several hybrid models. They state that their approach not only achieves significantly higher accuracy than the previous state of the art, but is also faster and scales linearly with the size of the schema graph.

Finally, the work of Egyed [40] on automated abstraction of class diagrams provides a summary or simplification of the schema by removing details not necessary on a higher, more abstract level. This abstraction technique uses abstraction rules that have input and result patterns. Abstraction rules define the semantics of how a set of model elements can be replaced by a less complex, more-abstract model element. The proposed abstraction algorithm then

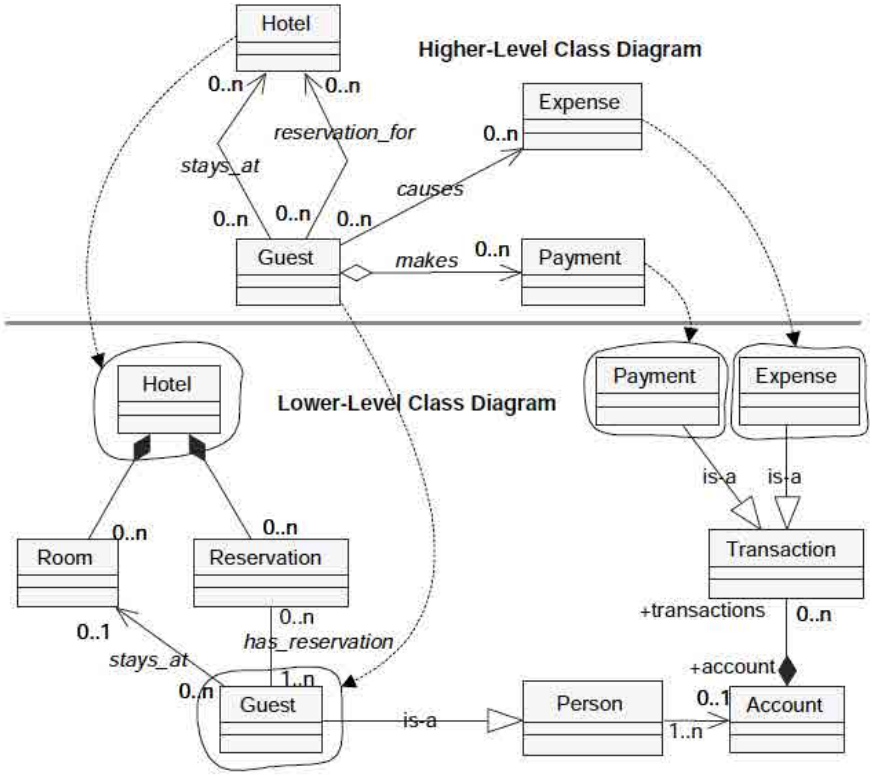


Figure 3.11. Example of summarization of a diagram. Extracted from [40].

performs syntactic matching of the abstraction rules on the model. Whenever an input pattern of a rule is encountered in the model, then that pattern is replaced by the result pattern of that rule. It follows that every application of a rule simplifies a given model. As an example, the higher-level diagram in Fig. 3.11 (top) summarizes the lower-level one (bottom) by omitting information considered less relevant.

Table 3.3 summarizes the previous summarization-based contributions.

Table 3.3. Summary of summarization-based contributions.

Contribution	Characteristics
Yu and Jagadish [140]	Definition of the concept of schema summary (applied to database schemas). Goals: present important schema elements and achieve broad information coverage. Importance of a schema element based on its connectivity in the schema and its cardinality in the database.

Table 3.3. Summary of summarization-based contributions (continued).

Contribution	Characteristics
Yang et al. [139]	Schema summary computation by using the entropy of table attributes in a link-analysis approach (applied to database schemas). Comparison with the approach of Yu and Jagadish through a benchmark database.
Egyed [40]	Automated abstraction of class diagrams (in UML). Bottom-up approach based on abstraction rules (every application of a rule simplifies a given model). Abstraction rules define the semantics of how a set of model elements can be replaced by a less complex, more-abstract model element.

3.3.4 Visualization Methods

The visualization of large-sized conceptual schemas to their final users is also in the scope of this thesis. The application of clustering or filtering methods must be followed by the application of visualization solutions in order to increase even more the understandability of schemas and provide a correct feedback.

Moody states in [79] that visual notations form an integral part of the language of software engineering, and have dominated research and practice since its earliest beginnings. They play a particularly critical role in communicating with end users and customers as they are believed to convey information more effectively to nontechnical people than text. Moody defines a theory of how visual notations communicate and based on this, a set of principles for designing cognitively effective visual notations, that must be taken into account. Moody also aims to raise awareness about the importance of visual representation issues in notation design, which has historically received very little attention.

Tzitzikas and Hainaut explain in [119] that diagram drawing is not a panacea. It has been recognized long ago that the usefulness of conceptual diagrams degrades rapidly as they grow in size. Although the article focus on visualization of ontologies, it could be translated to database or conceptual modeling schemas written in UML/OCL. The authors include filtering and clustering as visualization techniques. That is a good classification because such techniques improve the graphical understandability of schemas. Furthermore, context-based browsing is also introduced. It consists on showing only a short part of the whole schema so that the user could start browsing the diagram starting from any node of the schema. At each point in time, the neighborhood of the selected node is displayed. The user is then able to click on any other displayed node to change the focus. This way, the understanding of the schema is done gradually.

Another approach is presented by Streit et al. in [112] to manage large business process specifications. Such specifications can be seen as graphs and, therefore, the application of the

solution proposed can be extended to conceptual schemas. The solution here is adopted from the discipline of 3D computer graphics. It is possible to compare a large and complex diagram with a 3D representation of a full scale model. The authors explain that the purpose of simplification is to maintain a representation of the 3D model that is recognizable while reducing the processing and data requirements of the system. Similarly, in the case of conceptual schemas our purpose is to reduce the schema maintaining the relevant elements and a recognizable version of the whole, while reducing the understandability effort users should make. Figure 3.12 shows the structure of a 3D model of a plane. It is clear that the model continues looking as a plane although its level of detail (complexity and size) is lower in the left side than in the right.

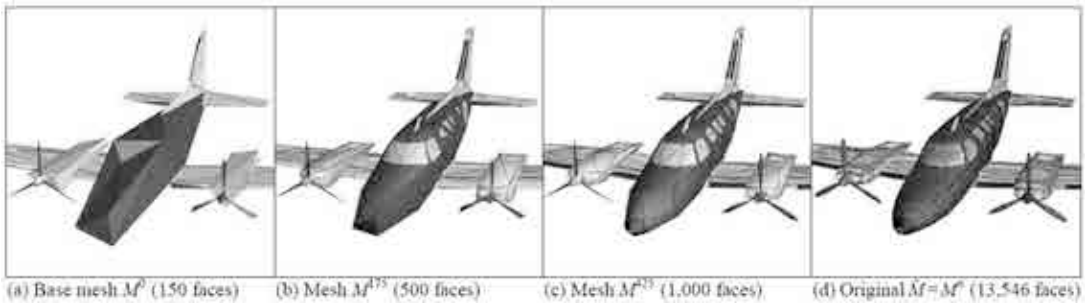


Figure 3.12. 3D mesh representing a plane at four different levels of detail. Extracted from [112].

The proposal of Streit et al. consists of calculating the relevance of each node according to a criterion function. The second step is to reduce the graph (or schema in our case) by collapse or decimation methods. Finally, the graph is displayed for the user inspection. We can affirm that such solution is close to filtering methods.

Focus+Context [81] is another visualization technique that worth a mention here. Kosara, Miksch and Hauser in [69] explain how to use focus+context techniques in information visualization to point out relevant information to a user. They present a method for doing this which uses selective blur to direct the user's attention. The main idea is to blur objects based on their relevance. As human perception divides our field of view into foreground and background, most relevant objects must be placed closer (in the foreground) than less important ones (in the background).

Another contribution in the literature is the review of Cockburn et al. [30]. It contains a summary of the different works and contributions in the visualization area and, in particular, those techniques classified in Overview+Detail, Zooming, or Focus+Context methods.

Overview+Detail techniques are characterized by the simultaneous display of both an overview and a detailed view of an information space, each in a distinct presentation space. The second category supporting both focused and contextual views is based on Zooming, which involves a temporal separation between views. User magnify (zoom in) or demagnify (zoom out) a fragment of the information in the visualization area to focus on desired elements. Finally, Focus+Context integrates a relevant focused view and the context into a single display where all parts are concurrently visible. The focus is displayed seamlessly within its surrounding context.

To finish this section, we also take into account the survey about ontology visualization methods of Katifori et al. [64]. Ontologies are sets of concepts with their relationships, so that we can apply the methods explained in such survey to conceptual schemas. The methods described in the survey are grouped in several categories, according to their visualization type.

The Zoomable and Focus+Context categories are the same as in the survey of Cockburn et al. [30]. On the other hand, the authors define Indented List techniques as lists where the elements of an ontology are hierarchically placed like in a directory tree view of file systems in common operating systems. The next category is Node-link and tree, which represents ontologies as a set of interconnected nodes, presenting the taxonomy with a top-down or left-to-right layout. The user is generally allowed to expand and retract nodes and their subtrees, in order to adjust the detail of the information shown and avoid display clutter. Finally, Space-filling techniques are based on the concept of using the whole of the screen space by subdividing the space available for a node among its children. The size of each subdivision corresponds to a property of the node assigned to it (its size, number of contained nodes, and so on).

Table 3.4 summarizes the previous visualization contributions.

Table 3.4. Summary of visualization contributions.

Contribution	**Characteristics**
Moody [79]	Principles for designing cognitively effective visual notations. Focuses on the physical (perceptual) properties of notations rather than their logical (semantic) properties. Identifies serious design flaws in some of the leading software engineering notations, together with practical suggestions for improving them.
Tzitzikas and Hainaut [119]	Focuses on the visualization requirements of large-sized ontology diagrams. Describes the main factors that determine whether an ontology diagram layout is satisfying or not.
Streit et al. [112]	A visualization technique to support the modelling and management of large business process specifications. Uses a set of criteria to produce views of the specification that exclude less relevant features. 3-step method: assessing the relevance of nodes, reducing the specification, and presenting the results.
Musial and Jacobs [81]	Application of focus+context visualization techniques to an interactive UML browser. Displaying software components at varying levels of detail according to a dynamic degree of interest function.
Kosara et al. [69]	Focus+context method that blurs objects based on their relevance (rather than distance) to direct the user's attention. Povides both detailed information of the currently most relevant objects, as well as giving users an idea of the context.

Table 3.4. Summary of visualization contributions (continued).

Contribution	Characteristics
Cockburn et al. [30]	Review and categorization of interface schemes that allow users to work at, and move between, focused and contextual views of a dataset. Four approaches: overview+detail, zooming, focus+context, and cue-based techniques.
Katifori et al. [64]	Present ontology visualization methods and categorize their characteristics and features in order to assist method selection and promote future research in the area of ontology visualization.

3.4 The Filtering Approach

Information filtering is a name used to describe a variety of processes involving the delivery of information to people who need it. As far as we know, there is no research study or contribution in the present literature about the application of specific information filtering methods and techniques to conceptual schemas. Nevertheless, we present close and relevant research related to our topic.

Belkin et al. consider in [10] the relationship between information filtering and information retrieval. They state that there is relatively little difference between the two, at an abstract level. First of all, their underlying goals are essentially equivalent. That is, both are concerned with getting information to people who need it, and both are concerned with more-or-less the same kind of context. Furthermore, most of the issues which appear at first to be unique to information filtering are, really, specializations of information retrieval problems. The conclusion they draw from this is that much of information retrieval research experience is directly relevant to filtering.

Researchers studying filtering also need to do a great deal of research on the dimensions of users information interests: what they might be, how to identify them, how to represent them, and how to modify them. This is especially the case because filtering is considering new classes of users, uses, and data, for which information retrieval does not, in general, have relevant results. For the case of filtering applied to large conceptual schemas, the study of the users and their needs is mandatory, in order to provide the right information to the correct user.

Hanani et al. explain in [55] that information filtering is one of the methods that is rapidly evolving to manage large information flows. The aim of information filtering is to expose users to only information that is relevant to them. Many information filtering systems have been developed in recent years for various application domains. Some examples of filtering applications are: filters for search results on the web that are employed in the Internet software, custom e-mail filters based on personal profiles, browser filters that block non-valuable information, filters designed to give children access only to suitable pages, filters for e-commerce applications

that address products and promotions to potential customers only, and many more.

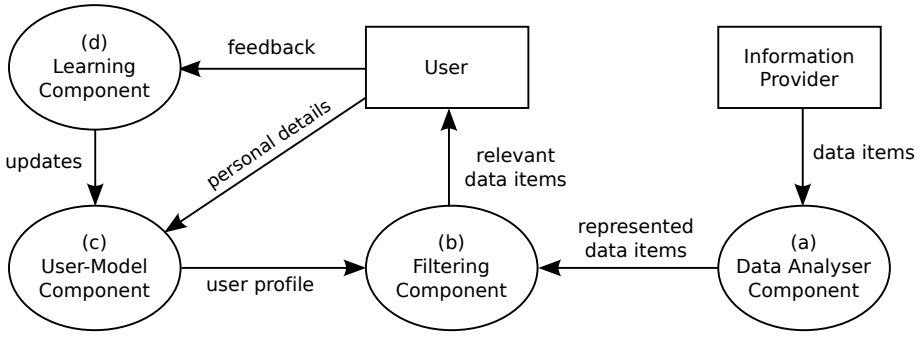


Figure 3.13. A generic model of information filtering systems. Extracted from [55].

The different systems use various methods, concepts, and techniques from diverse research areas like: Information Retrieval, Artificial Intelligence, or Behavioral Science. There are many systems of widely varying philosophies, but they all share the goal of automatically directing the most valuable information to users in accordance with their need, and of helping them use their limited reading time most optimally. The authors define a generic model of information filtering systems that includes four components (see also Fig. 3.13):

- **The data-analyzer component:** obtains or collects data items from several information providers.
- **The user-model component:** explicitly or implicitly gathers information about the users and their information needs.
- **The filtering component:** matches the user needs with the represented data items and decides if a data item is relevant to the user.
- **The learning component:** detects changes in the information or user needs to improve further filtering.

Kuflik and Shoval [71] present the representation of the user's need with user profiles. The quality of a user profile has a major impact on the performance of information filtering systems. Kuflik and Shoval focus on the study of user profile generation and update. Some of such methods are:

- **User-created profile:** the user specifies her area(s) of interest by a list of (possibly weighted) terms.
- **System-created profile by Automatic Indexing:** a set of data items, which have already been judged by the user as relevant, are analyzed in order to identify the most frequent and meaningful items. Those items are weighted and constitute the user profile.
- **System- plus user-created profile:** first, an initial profile is created automatically. Then, the user reviews the proposed profile and updates it.

- **System-created profile based on learning by artificial neural-network (ANN):** an ANN is trained with the data items selected by the user to serve as the user profile for future filterings. Extended studies about the usage of ANN in information filtering systems are presented in [20, 70].
- **User-profile inherited from a user-stereotype:** this method assumes that the information filtering system has pre-defined user-stereotypes. A new user is attached to a predefined stereotype to which she is close with respect to demographic and social attributes.
- **Rule-based filtering:** consists of a set of filtering rules. Questioning the user on her information usage and filtering behavior can generate such rules.

Shapira et al. [103, 102] suggest an advanced model for information filtering which is based on a two-phase filtering process and the use of stereotypes. The user profiling in the filtering method is constructed on the basis of the user areas of interest and on sociological parameters about her that are known to the system. The first phase is cognitive filtering, i.e., a correlation between the contents of the information to be filtered with a predefined weighted vector that represents the areas of interest to the user. The second phase implements a sociological filtering process. Each user profile contains personal sociological parameters (such as employment, affiliation, education, etc.). These parameters relate the user to one or more stereotypes known to the system, which maintains a database of known stereotypes that includes rules on their information retrieval needs and habits. During the filtering process, the system relates the user to one or more stereotypes and operates the appropriate stereotypical rules.

Finally, Maidel et al. [73] present a new ontological-content-based method for ranking the relevance of items in the electronic newspapers domain. The method is being implemented in a personalized electronic newspaper research project. The content-based part of the filtering method uses a hierarchical ontology of news items. The method considers common and *close* ontology concepts appearing in the user's profile and in the item's profile, measuring the hierarchical distance between concepts in the two profiles. Based on the number of common and related concepts, and their distances from each other, the filtering algorithm computes the similarity between items and users, and rank-orders the news items according to their relevancy to each user, thus providing a personalized newspaper. The relevant contribution here is the use of an ontology to define the profile similarity.

Table 3.5 summarizes the previous filtering contributions.

Table 3.5. Summary of filtering contributions.

Contribution	**Characteristics**
Belkin et al. [10]	Comparison between information retrieval and information filtering. Conclusion: much of information retrieval research experience is directly relevant to filtering.

Table 3.5. Summary of filtering contributions (continued).

Contribution	Characteristics
Hanani et al. [55]	A framework to classify information filtering systems according to several parameters. Discusses methods and measurements that are used for evaluation of information filtering systems and limitations of the current systems.
Kuflik and Shoval [71]	Focus on the study of user profile generation and update. Introduces methods for user profile generation, and proposes a research agenda for their comparison and evaluation.
Shapira et al. [103, 102]	Advanced model for information filtering which isbased on a two-phase filtering process. The first phase is cognitive filtering, i.e., a correlation between the contents of the information to be filtered with the areas of interest to the user. The second phase implements a sociological filtering process, where user characteristics are related to one or more stereotypes known to the system.
Maidel et al. [73]	Describes a new ontological content-based filtering method for ranking the relevance of items for readers of news items, and its evaluation. Computes the similarity between item and user profiles and ranks the news items according to their relevance to each user.

3.5 Comparison between Approaches

The previous sections explained the problem that very large conceptual schemas entail and reviewed the most popular approaches in the literature that deal with it. This section describes in more detail what we consider by information filtering and compares it with other similar approaches from the literature.

As aforementioned, there are many filtering systems of widely varying philosophies, but all share the goal of automatically directing the most valuable information to users in accordance with their needs, and of helping them use their limited time and information processing capacity most optimally.

Figure 3.14 shows an schematic overview of the information filtering process and steps applied to large conceptual schemas. Apart from the schema itself, information filtering methods require as input a representation of the user information need or interest in form of knowledge request. That way, the method adapts to the specific request and produces varying outputs. After the request processing and the analysis of the large schema, the filtering method selects which are the elements from the original large schema to include in the resulting schema of small size, which represents the feedback the user obtains.

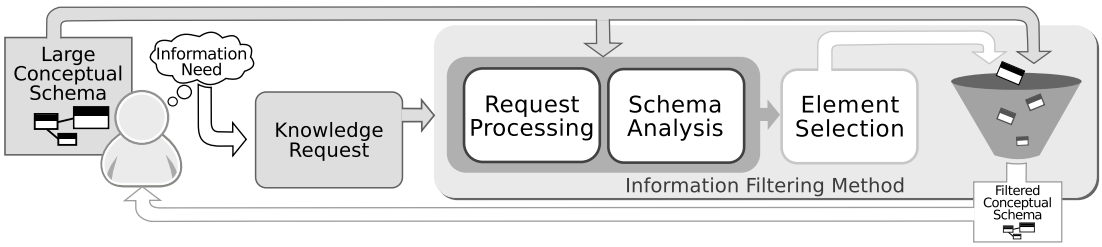


Figure 3.14. Information filtering in the context of large conceptual schemas.

We have seen that there exist several methods and techniques in the literature with the same purpose as information filtering. All of these families of proposed solutions to face the problem introduced by very large conceptual schemas have their own particular characteristics. Figure 3.15 shows an overview of the most popular families of methods:

- **Clustering methods:** classify the elements of the schema into groups, or clusters, according to a similarity function. For example, left of Fig. 3.15 describes 7 clusters.
- **Relevance methods:** apply a ranking function to the elements of the schema in order to obtain an ordered list (also called ranking of them according to their general relevance).
- **Summarization methods:** compute a reduced general schema from the large original one with the elements that are more relevant but also with a a high degree of coverage of the schema.

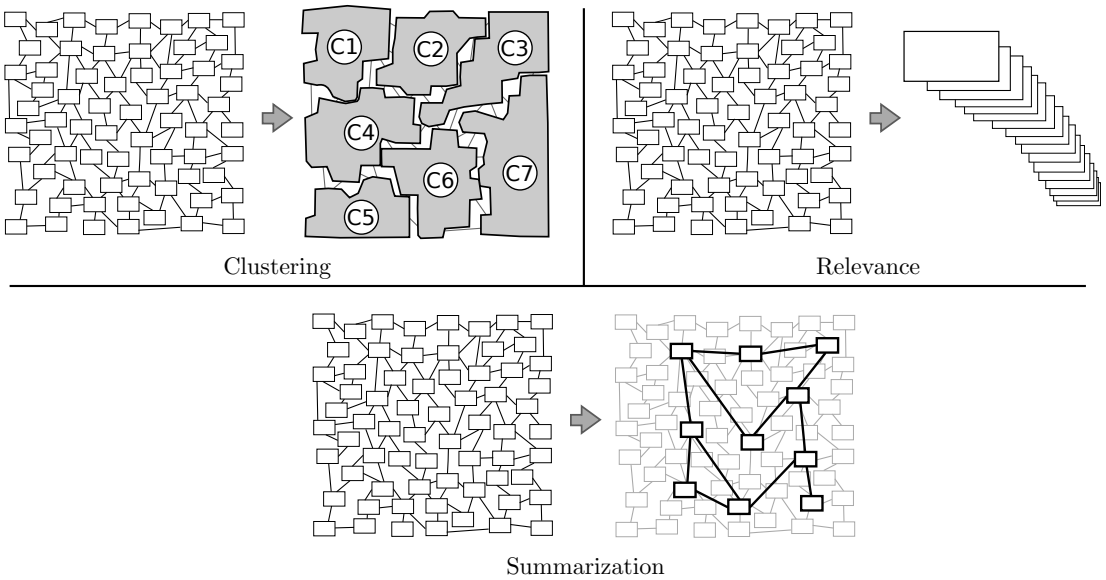


Figure 3.15. Existing families of methods in the literature.

Table 3.6 shows a comparison between the previous methods and the information filtering. First characteristics to study are the input and output of the methods. Generally, clustering, relevance, and summarization methods only need the conceptual schema as their input. Their output depends only on the input schema.

Clustering methods return a classification of the elements in clusters. Relevance methods return a ranking of the elements according to importance in the schema. And finally, summarization methods return a reduced schema that is called *summary*. Consequently, the retrieved knowledge by all those families of methods is general, i.e., it is always the same —it does not change unless the schema changes— and it is not affected by specific user interests.

Alternatively, the family of filtering methods also need a representation of the user need or interest in order to derive a specific output from the general one. That is why the retrieved knowledge in this case is marked as specific in Tab. 3.6 while it is denoted as general for the other alternatives.

	Clustering	Relevance	Summarization	**Filtering**
Input	schema	schema	schema	**schema and user request**
Output	clustered schema	ranking	schema summary	**filtered schema**
Retrieved Knowledge	general	general	general	**specific**
User Interaction	static exploration	static exploration	static exploration	**dynamic request/response**

Table 3.6. Comparison of methods to deal with large conceptual schemas.

Finally, the filtering-based family of methods provides users with a dynamic request/response interaction that simplifies the knowledge extraction process. Other alternatives provide static access to their output, which in many cases contain only a fragment of the whole knowledge that may be out of the scope of interest to the user. Therefore, the development of a filtering approach to deal with large conceptual schemas covers a relatively unexplored area in the literature and helps to providing a more dynamic and flexible solution in comparison to the existing methodologies that contribute to this topic.

3.6 Summary

The analysis presented in the previous sections can be summarized in two main conclusions.

On one hand, it is clear that the major contributions in the literature dealing with the problems of conceptual modeling in the large are mainly applied to database or entity-relationship schemas. As seen earlier in the chapter, although nowadays UML/OCL are the de-facto standard modeling languages, the number of methods to reduce or restructure conceptual schemas defined using UML/OCL is very small. Furthermore, the amount of contributions in the filtering area is also minimal. Most of the existing methods do not support user queries or a valuable interaction.

The existing literature denotes a lack in the proposal of contributions from the field of information retrieval and, particularly, information filtering, to the problem of dealing with the huge amount of information and the complex structure large conceptual schemas entail. Conceptual modeling and recent model-driven approaches require an extensive use of those large schemas to carry out their activities.

What is needed is to study and define the different phases of an information filtering engine, in the context of very large conceptual schemas. Also, it is mandatory to detect the main issues in the study of the characteristics of the knowledge large schemas contain and propose methods to select and represent the user interest in order to specialize the results of the filtering engine. Furthermore, it is mandatory to provide knowledge extraction techniques aligned with the user interest representation, and to present such knowledge in an appropriate way to simplify its understanding.

On the other hand, most of the contributions do not take into account the whole knowledge provided by the conceptual schema. Concepts like constraints, derivation rules, definition of actions and events are commonly avoided. However, there exists the thought of the more knowledge used, the more complete the results will be.

Regarding these trends, we notice that a new approach to solve the problem of large conceptual schemas should be provided by a filtering engine applied to UML/OCL schemas. Such an engine should include in their computation new measures, which could enclose the knowledge that schemas contain. To this end, Ch. 4 presents specific metrics to compute the relevance of schema elements that take into account knowledge from the structural and behavioral components of a large conceptual schema. These metrics are the core element of the filtering method for large conceptual schemas that we present in Ch. 5 as one of the main contributions of this thesis.

Controlling complexity is
the essence of computer programming
Brian Kernigan

4

Relevance Metrics for Large Conceptual Schemas

The methods for understanding of large conceptual schemas require computing the importance of each entity and event type in a large UML conceptual schema. Each method is defined by means of a set of one or more atomic metrics. Such metrics gather the particularities of a conceptual schema and define its characteristics. In principle, the totality of knowledge defined in the schema could be relevant for the computation of that metrics but existing methods only take into account a small part of the knowledge represented in a schema. In this chapter, we complete those methods with additional knowledge, and construct the basis for the core component of our filtering engine.

Section 4.1 describes the motivation of the chapter in the context of the filtering engine for large conceptual schemas. Section 4.2 presents the details about basic measures to structure and characterize a large schema, and our proposed extensions to such measures in order to take into account additional knowledge that must be included in the computation of the relevance of schema elements. Section 4.3 introduces the set of importance-computing methods used by our filtering engine, and compares them. Section 4.4 describes the need to develop new metrics to obtain a user-centered approach of the definition of relevance. Section 4.5 completes the importance methods with the analysis of a simple closeness-computing method adapted to UML schemas. The combination of importance and closeness methods produces the interest-computing method specified in Section 4.6. Our method computes the interest of each entity and event type in the schema with respect to the specific user information needs, and allows to construct the filtering method of this thesis. Finally, Sect. 4.7 summarizes the chapter.

4.1 Motivation

In this section we describe how a large conceptual schema can be filtered using the relevance metrics described in this chapter. The main idea of the overall process is to extract a reduced and self-contained view from the large schema, that is, a filtered conceptual schema with the knowledge of interest to the user. Figure 4.1 presents the three steps of our filtering methodology. The full details about this process are presented in Ch. 5

The first step consists in preparing the required information to filter the large schema according to the specific needs of a user. Basically, the user focus on a set of schema elements she is interested in, which is a subset of the large conceptual schema, and our method complements them with additional schema knowledge highly relevant to user selection. Therefore, it is mandatory for the user to select a non-empty initial focus set of elements of interest in order to avoid empty requests.

The relevance metrics are the main tool to structure and analyze the whole knowledge represented within a large conceptual schema. Such metrics allow our filtering methodology to transform an information need of a particular user into an specific feedback of interest to her. Opposite to clustering or summarization methodologies, the resulting filtered conceptual schema our method obtains changes and adapts according to the concrete input of the user. To achieve this goal, we combine importance-computing metrics that calculate the general relevance of the elements of a large schema, with a closeness-computing metric that selects those elements that are relevant and have a high relation with the elements of the user input.

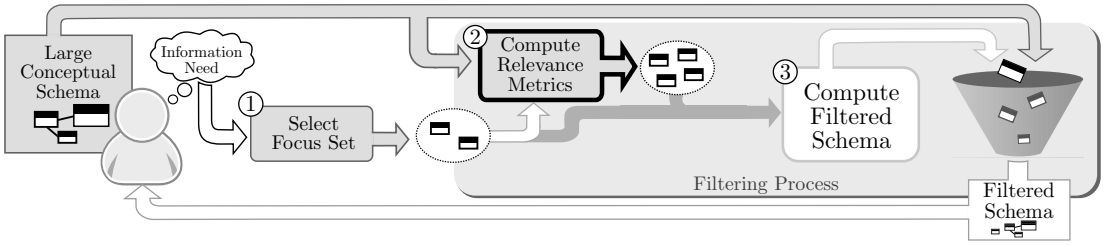


Figure 4.1. General structure of the filtering process.

During the second step our method computes the required metrics to automatically select the most interesting schema elements to extend the knowledge selected in the focus set of the first step. The main goal of these metrics is to discover those elements that are relevant in the schema but also that are close (in terms of structural distance over schema) to the elements of the focus set. A detailed definition of such metrics is described in the following sections.

Finally, the last step receives the set of most interesting schema elements selected in the previous step and puts them together with the elements of the focus set in order to create a filtered conceptual schema with the elements of both sets. The main goal of this step consists in filtering information from the original schema involving elements in the filtered schema. To achieve this goal, the method explores the attributes, schema rules, relationships, and generalizations/specializations in the original schema that are defined between those filtered elements and includes them in the filtered schema to obtain a connected schema.

4.2 Topological Measures of Conceptual Schemas

Network topology is the layout pattern of interconnections of the various elements (links, nodes, etc.) of a computer or biological network. Conceptual schemas can be seen as networks or graphs whose entity and relationship types are the nodes and link of the network, respectively. Therefore, we can specify several topological measures over the graph structure of a large conceptual schema. In the following, we study a set of basic topological measures and we propose extensions to them taking into account additional knowledge in order to define complex measures of interest to our filtering methodology.

4.2.1 Basic Measures

Discussing about the relevance of schema elements has a point of ambiguity since there are a lot of types of elements in a conceptual schema including entity, event, and relationship types, schema rules, attributes, and generalizations. Although computing the relevance of attributes, associations or constraints could be a research topic of interest, our approach, similarly as major contributions in the literature, has the computation of the relevance of entity types as the main goal of the thesis. Furthermore, since we define event types as particular entity types [89], we can also compute their relevance with the same methods than entity types.

The most important kind of single elements in a conceptual schema are entity types, because they are the representation of concepts of the real world. Therefore, the application of methods for focusing on the most relevant ones must contribute in a higher degree to increase understandability and usability. In this section we go over the main concepts and the notation we have used to define the knowledge of conceptual schemas, as explained in Ch. 2. In this thesis we deal with schemas written in the UML[84]/OCL[85], although our method can be directly adapted to schemas defined in other modeling languages. Table 4.1 summarizes the notation (inspired by [121, 6]) used in the rest of the document.

As aforementioned, conceptual schemas contain a structural subschema and a behavioral subschema. The structural schema consists of a taxonomy of entity types (a set of entity types with their generalization/specialization relationships and the taxonomic constraints), a set of relationship types (either attributes or associations), the cardinality constraints of the relationship types, and a set of other static constraints formally defined in OCL.

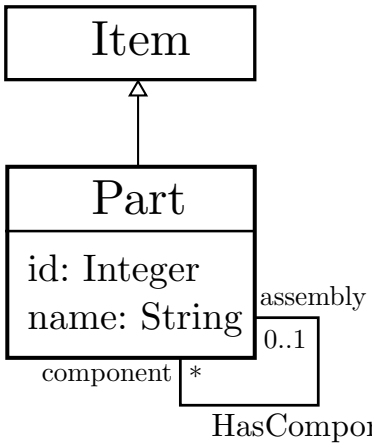
We denote by $\mathcal{E}$ the set of entity types defined in the schema. For a given $e \in \mathcal{E}$ we denote by $par(e)$ and $chi(e)$ the set of directly connected ascendants (parent entity types) and descendants (children entity types) of e, respectively, and by $gen(e)$ the union of both sets. The set of attributes defined in the schema is denoted by $\mathcal{A}$. If $a \in \mathcal{A}$ then $entity(a)$ denotes the entity type where a is defined. The set of attributes of an entity type e is denoted by $attr(e)$.

The set of associations (relationship types) defined in the schema is denoted by $\mathcal{R}$. If $r \in \mathcal{R}$ then $members(r)$ denotes the set of entity types that participate in association r, and $assoc(e)$, where $e \in \mathcal{E}$, the set of associations in which e participates. Note that an entity type e may participate more than once in the same association, and therefore $members(r)$ and $assoc(e)$ are multisets (may contain duplicate elements).

Notation		Definition
$par(e)$	=	$\{e' \in \mathcal{E} \mid e \ IsA \ e'\}$
$chi(e)$	=	$\{e' \in \mathcal{E} \mid e' \ IsA \ e\}$
$gen(e)$	=	$par(e) \cup chi(e)$
$attr(e)$	=	$\{a \in \mathcal{A} \mid entity(a) = e\}$
$members(r)$	=	$\{e \in \mathcal{E} \mid e$ is a participant of $r\}$
$assoc(e)$	=	$\{r \in \mathcal{R} \mid e \in members(r)\}$
$conn(e)$	=	$\uplus_{r \in assoc(e)} \{members(r) \backslash \{e\}\}$[1]
$par_{inh}(e)$	=	$par(e) \cup \{par_{inh}(e') \mid e' \in par(e)\}$
$chi_{inh}(e)$	=	$chi(e) \cup \{chi_{inh}(e') \mid e' \in chi(e)\}$
$attr_{inh}(e)$	=	$attr(e) \cup \{attr_{inh}(e') \mid e' \in par(e)\}$
$assoc_{inh}(e)$	=	$assoc(e) \uplus \{assoc(e') \mid e' \in par_{inh}(e)\}$
$conn_{inh}(e)$	=	$conn(e) \uplus \{conn(e') \mid e' \in par_{inh}(e)\}$

Table 4.1. Definition of basic metrics.

Moreover, $conn(e)$ denotes the multiset of entity types connected to a particular entity type e through associations. For example, if r is the association HasComponent(assembly:Part, component:Part), then $members(r)$={Part, Part}, $assoc$(Part)={HasComponent, HasComponent} and $conn$(Part)={Part}. Figure 4.2 depicts this example.



par(Item) = Ø
chi(Item) = {Part}
gen(Item) = {Part}
attr(Item) = Ø
assoc(Item) = Ø
conn(Item) = Ø

par(Part) = {Item}
chi(Part) = Ø
gen(Part) = {Part}
attr(Part) = {id, name}
assoc(Part) = {HasComponent, HasComponent}
conn(Part) = {Part}

members(HasComponent) = {Part, Part}

Figure 4.2. Example of basic metrics.

[1]Note that "\" denotes the difference operation of multisets as in $\{a, a, b\} \backslash \{a\} = \{a, b\}$ and "$\uplus$" denotes the multiset (or bag) union that produces a multiset as in $\{a, b\} \uplus \{a\} = \{a, a, b\}$

The last row section in Table 4.1 defines the notation we use to take into account the inherited properties from the ancestors of entity types. As a special case, $chi_{inh}(e)$ is the set that includes all the descendants of e. The other metrics capture the entire hierarchy of each entity type (e.g., $assoc_{inh}(e)$ collects all the associations where $e \in \mathcal{E}$ and its parents —and the parents of them, recursively— participates). Since $assoc(e)$ is a multiset, $assoc_{inh}(e)$ can also contain repeated elements.

The set of basic metrics of Table 4.1 does not include metrics to gather the knowledge contained in the behavioral schema nor the schema rules of the structural schema. The reason is that the relevance methods we have selected to study (described in Section 4.3.2) do not use such information. The notation for these extended metrics is introduced in Section 4.2.3 before the description of the extended versions of the basic methods.

4.2.2 Extended Characteristics of Conceptual Schemas

This section presents some of the new approaches introduced in this thesis to gather the knowledge of the additional components of the conceptual schema schema. First of all, we introduce how to deal with complex relationship types, including relationships whose degree is $n > 2$ and reified relationships.

Next step consists in describing the power OCL brings us to take into account the knowledge provided by the schema rules ($\mathcal{C}$) of the conceptual schema. Our research here allows to uncover existing relationships between entity types placed in OCL expressions that are useful in the relevance computation process.

Finally, we mix the previous techniques to introduce new metrics to squeeze the knowledge previously extracted, mainly from the behavioral schema and the schema rules of the structural schema. Such metrics will be the keystone in the extension of the base versions of the selected methods from the literature introduced in Section 4.3.2.

Complex Relationship Types

The Unified Modeling Language (UML) allows to specify a special kind of relationship type that can have the same behaviour as entity types: the association classes. Thus, the concept of *reification* is introduced as viewing a relationship as an entity. Reification can easily be defined in UML, as shown in left sides of Fig. 4.3 and Fig 4.4 for binary (entity C) and n-ary (entity AC) relationship types.

Since current methods in the literature are applied to schemas defined with entity-relationship diagrams but not with UML/OCL, the problem of how to take into account association classes is not explored in this field. To solve this, we propose to follow the same approach than Olivé in [87]. It consists in implicitly reifying the current association class into another entity type associated through new binary relationships with its previous participants. Figure 4.3 depicts the reification process.

It is important to maintain the same semantics as before the explicit reification. Thus, the modeler has to take care of cardinality constraints. In the case of a binary reified relationship (left side of Fig. 4.3), the cardinality constraints after the explicit reification are interchanged between the ends of the relationship and the association class. This way, the cardinality constraints of a relationship end go to the new relationship between the other end and the new entity representing the previous association class, in the side of the new entity type. In the side of the previous ends the cardinality equals 1.

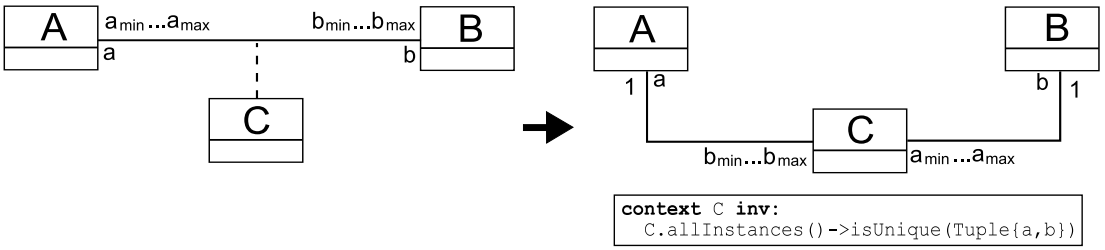


Figure 4.3. Implicit reification of a binary relationship with association class.

Furthermore, to maintain the semantics we also need a new uniqueness constraint for the new entity type after the implicit reification. Such constraint cannot be defined graphically, so the modeler requires the use of OCL. Such constraint can be written as shown in Fig. 4.3. Basically, we need and invariant to check that the same pair of instances of both ends (A and B in the image) cannot be linked with the same instance of the new entity type (C in the example) more than once.

After this process, it is possible to use the same metrics previously defined because the conceptual schema only contain common relationship types —the association classes have been converted into a group of binary relationships connecting a new entity type that represents the original association class.

To sum up, in the case of binary relationships with association class the steps are as follows:

1. Define the **explicit entity type** representing the association class.
2. Define the **new binary relationship types with** its cardinality constraints.
3. Define the **uniqueness constraint** to maintain the original semantics.

Association classes in n-ary relationships need a different conversion process to be transformed into explicit reifications. First step follows the same rules as in the case of binary association classes. Figure 4.4 indicates that the association class (AC in the figure) is changed by a common entity type with new relationships reaching each of the initial ends.

After that, the new cardinality constraints (see right side of Fig. 4.4) are *1..1* for the association end in the side of each participant entity type in the original n-ary association, and *0..** for the association end in the side of the entity type representing the association class. All the cardinality constraints are equal. To maintain the original semantics, we need schema rules to control any original cardinality constraint in the reified schema.

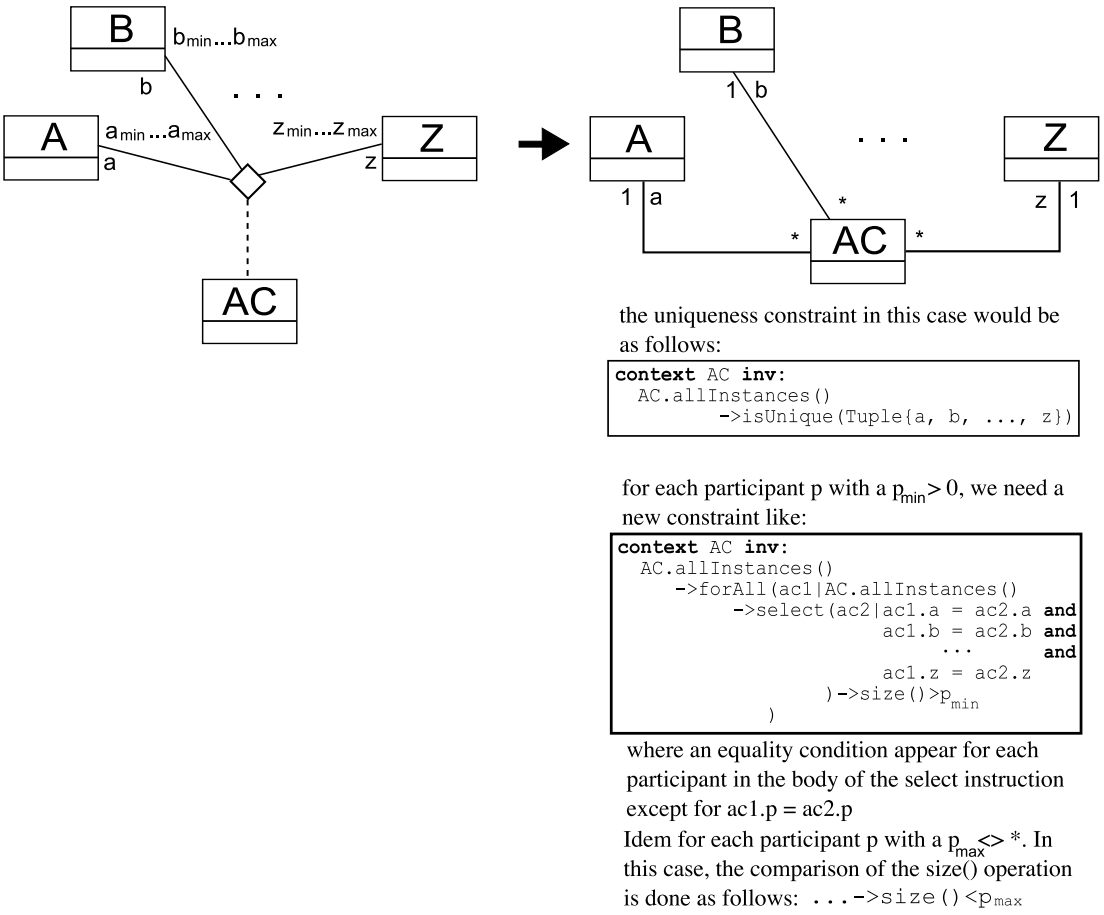


Figure 4.4. Implicit reification of a n-ary (n>2) relationship with association class.

As before, in the case of binary relationships with association class the steps are as follows:

1. Define the **explicit entity type** representing the association class.
2. Define the **new binary relationship types** with its cardinality constraints.
3. Define the **uniqueness constraint** to maintain the original semantics.
4. Define as many constraints as needed to maintain the same **cardinality constraints** as before.

In this case a uniqueness constraint is also needed and follows the same idea as in the case of binary relationships although this time the `Tuple` inside the OCL expression has as many members as the degree of the current reified association —each one of the members is one of the ends. Furthermore, for each original cardinality constraint (left side of Fig. 4.4) a new schema rule has to be introduced to maintain the semantics. Such rule must be defined in OCL following the same pattern as the one shown in Fig. 4.4.

Another example of an implicit reification is shown in Fig. 4.5. We have a ternary association connecting the entity types Table, Customer, and Date, reified into the association class Reservation. Following the previous steps, the association class Reservation is transformed into a new entity type with three relationships types connecting it with its original ends. Each of these relationships have the same cardinality constraints. Concretely, *1..1* in the side of the entity types Table, Customer and Date, and *0..** in the side of the entity type Reservation. After constructing the new structure, we need to specify the uniqueness constraint which consists on the uniqueness of a tuple with three elements as shown in Fig. 4.5.

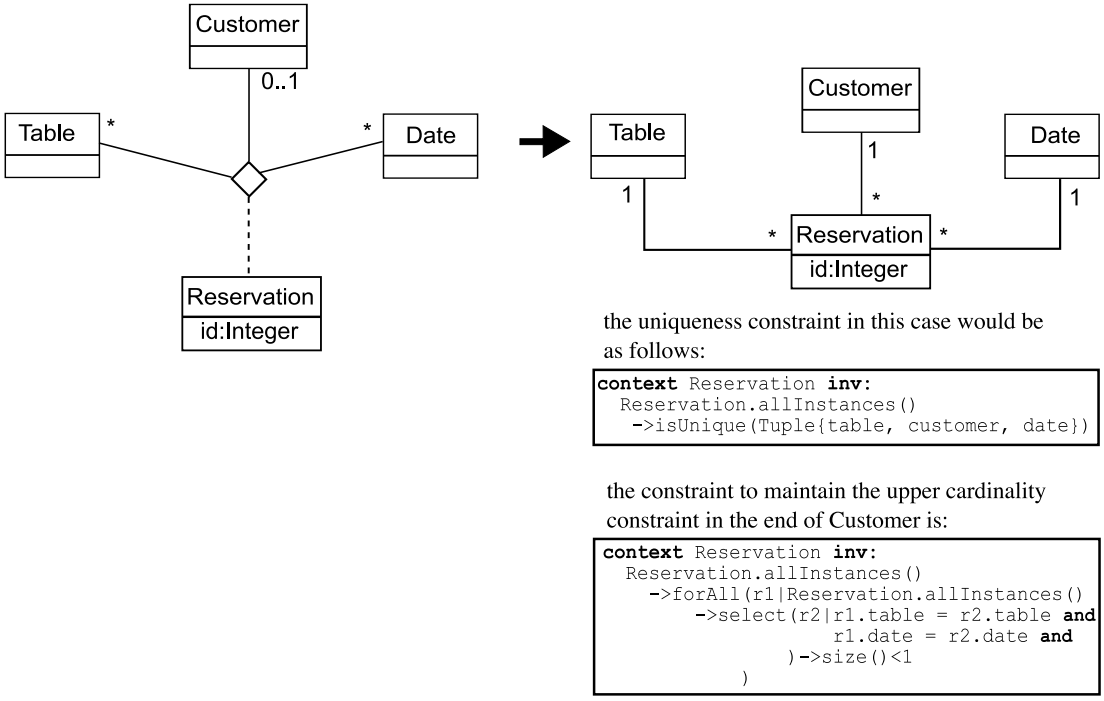


Figure 4.5. Implicit reification of a ternary relationship with association class.

Finally, the upper cardinality constraint *0..1* in the end of Customer have to be maintained. To do that, another schema rule taking into account the semantics of such cardinality constraint is also shown in Fig. 4.5. The schema rule controls that given an specific date and an specific table, only a unique reservation is allowed to the same client.

The implicit reification of association classes solves the problem of how to use the metrics with association classes. However, a new doubt appears about whether the relevance of the new entity types representing the reified association classes should be computed or not. After some evaluations, we decide to include such entities in the importance computation process because, since they are entity types, the concepts they represent really matter.

The Power of OCL

The schema rules defined in a conceptual schema are now taken into account. In this section we present a new approach to include the knowledge provided by such rules.

One of the most important type of elements in conceptual schemas are the relationship types. Such relationships are the key in relevance-computing methods based on link analysis. Our main goal here is to extract additional links between entity types from the navigations and elements that appear in schema rules.

As has been said, the Object Constraint Language provides a powerful textual notation to specify schema rules in a declarative way. The notation of the OCL includes a set of expressions (see Fig 4.6), some of them are used to describe navigations between elements of the schema (e.g., access to attributes and navigation to association ends with *PropertyCallExp* expression of Fig. 4.7).

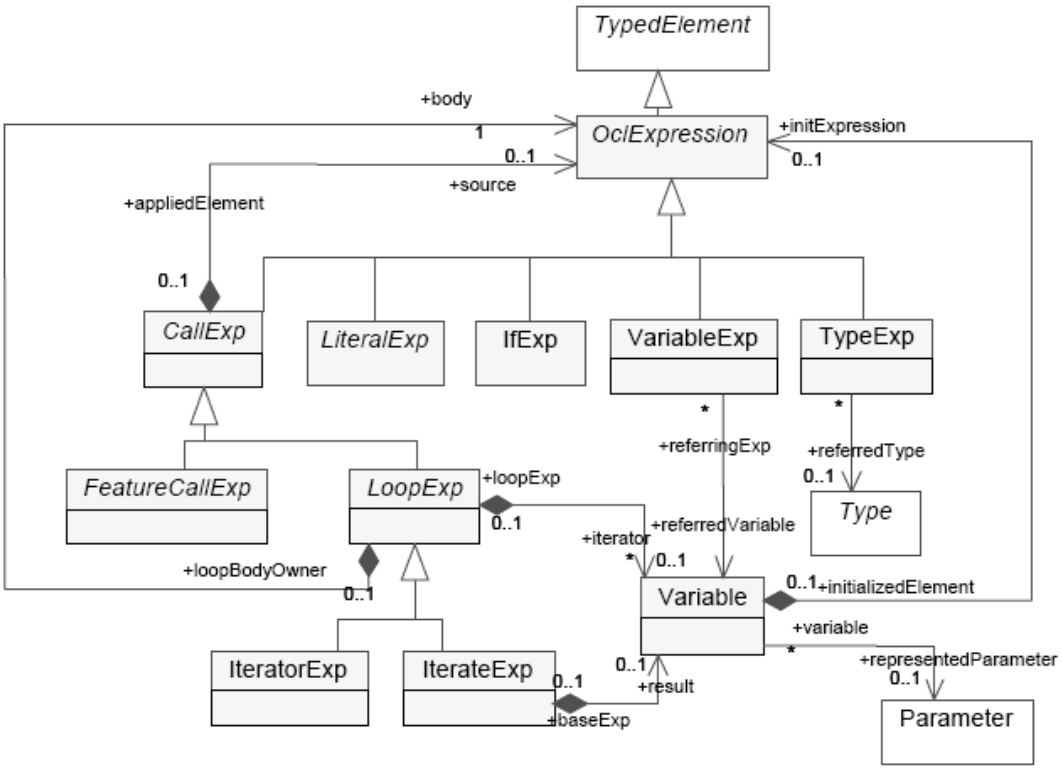


Figure 4.6. Fragment of the OCL metamodel including and overview of the expressions. Extracted from [85].

Furthermore, the body of a schema rule specified in OCL may contain several navigations to the association ends owned by entity types that participate in relationship types. It also may indicate navigations to a set of attributes defined in the context of other entity types, call to operations, usages of other schema elements, and so on. Such set of references can be used in the relevance computation process.

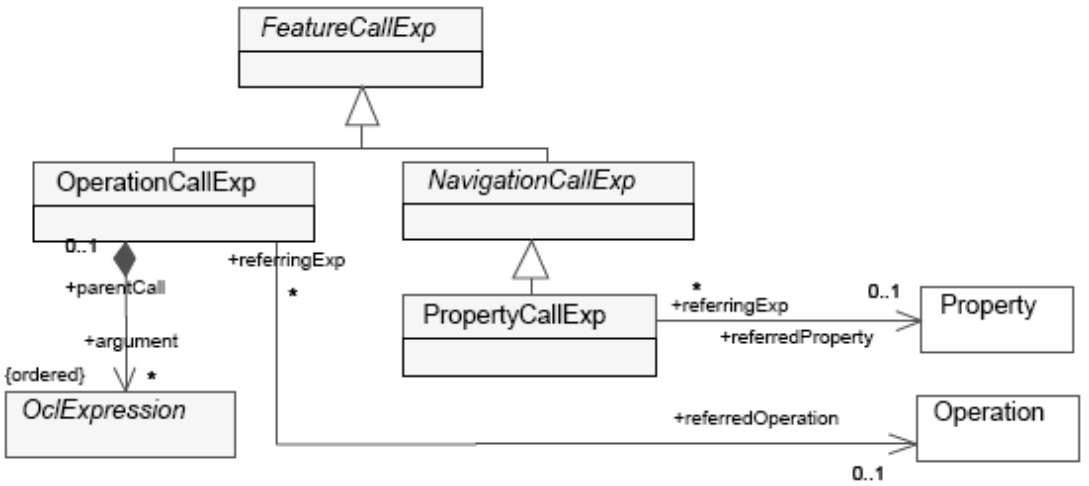


Figure 4.7. Fragment of the OCL metamodel including navigation expressions. Extracted from [85].

Each schema rule is defined in the context of an entity type. If we go through the schema rules defined in a conceptual schema we note that the first part of each rule is a *context* environment where to declare the entity that owns the rule. If we want to specify an invariant to constraint the values of an attribute, it is clear that the context of such rule will be the entity type that owns the attribute. Thus, if we mix the references with the concept of context we obtain a set of links joining the context entity type with each of the referenced (let's say participants) entity types in the body of the rule. The new uncovered links act as virtual edges in the graph of entities and relationships types that conforms the conceptual schema. Such edges must be included in the computation process in order to take into account the knowledge added by invariants, derivation rules, and pre- and postconditions, all of them written in OCL.

To understand our approach we show an example in Fig. 4.8 with a simple schema containing three entity types and two relationships. The top of the figure describes a schema rule that checks the number of seats of a room. Concretely, such number must be greater or equal than the number of attendants (the audience) of the related meeting. Furthermore, in the right side of the top of the figure there is a breakdown of the different sections of the OCL expressions to show the referenced entity types. It is possible to distinguish between the entity types that participate in the body of this schema rule (indicated with dashed lines) and the context entity type (indicated with a dotted line). Also, bold arrows indicate the structural navigations through relationship types in the schema rule (from self, i.e. Room, to Meeting, and from Meeting to Person using the rolename *audience*).

On one side, our approach consists in creating new links between the context entity type and each one of the participants of the schema rule. We name these kind of links *Context-Participant Non-Structural* links (CPNS). Although an entity type may participate more than once in the same body of a schema rule, only one CPNS link is created between the context entity type an such entity type. It is possible to see the CPNS links of the example in the right side of the bottom part of Fig. 4.8 (and with dotted lines in the schema).

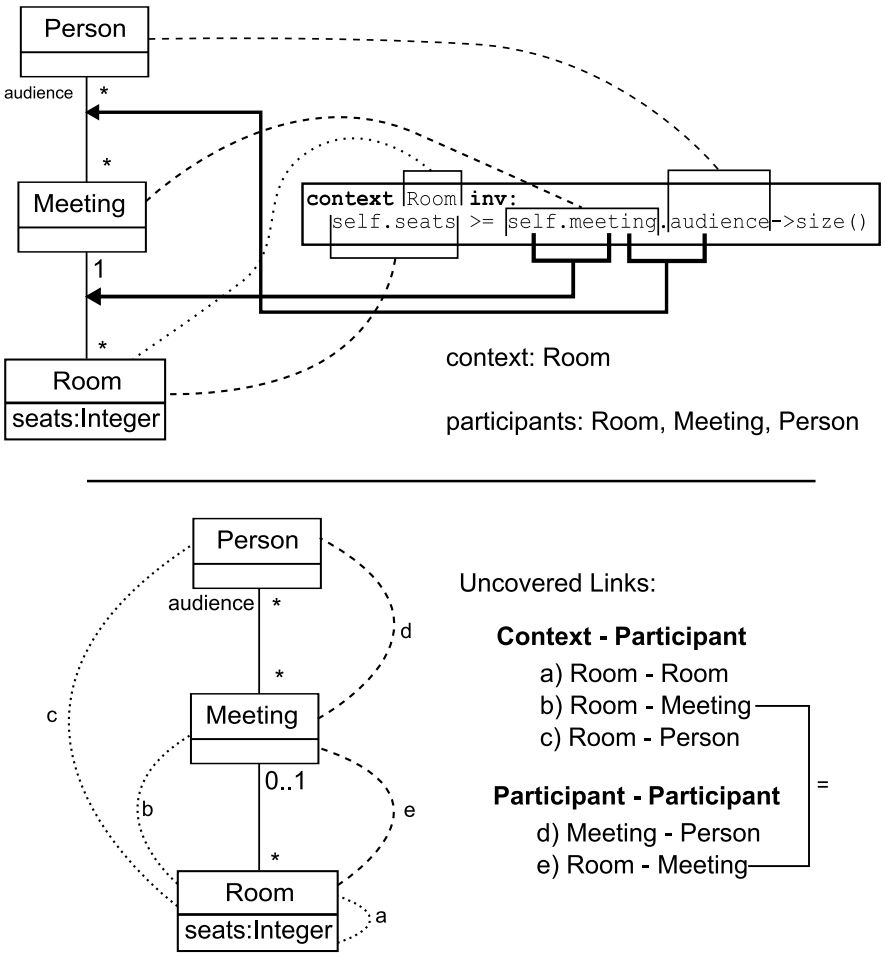


Figure 4.8. Example of uncovered links extracted from the OCL.

On the other hand, we also create links between the participants that appear as source and target in a navigation expression of the OCL. In the example of Fig. 4.8, the bold arrows of the top indicate two structural navigations. We mean structural navigations as such navigations through relationship types. In this case we have the OCL expression `self.meeting`, which indicates a navigation through the relationship between Room and Meeting (`self` references the context of the schema rule, in this case, Room). Also we have `meeting.audience` that shows the navigation through the relationship between Meeting and Person.

Then, our approach consists in creating new special links to note the use of navigations through relationships in the body of a schema rule. We name these kind of links *Participant-Participant Structural* links (PPS). It is possible to see them in the bottom part of Fig. 4.8, with dashed lines.

Finally, once we have extracted the CPNS and PPS links from the schema rule, next step consists in avoiding repetitions of links. Therefore, the final set of uncovered links from a schema

rule is the union of both sets. This way, the repeated link Room-Meeting in the example will appear only once.

To sum up, the steps we have to follow to obtain new links between entity types from a schema rule are:

1. Detect the **context** entity type of the schema rule.
2. Detect the **referenced entity types** in the body of the schema rule.
3. Construct links between the context and the referenced entity types (**CPNS links**).
4. Construct links between the entity types that participate in navigation expressions of the OCL (**PPS links**).
5. Apply the **union** operation with the previous sets to delete repetition of links.
6. The result of the previous step is the set of **uncovered links** of the schema rule.

Next, we will introduce new metrics that will take into account the extracted links from each one of the schema rules of the conceptual schema.

Another knowledge in the schema that has not been considered before is the information provided by cardinality constraints in association ends of relationship types. Such information can be added to the relevance computation process of entity types using the OCL. Cardinality constraints can be converted into schema rules specified in OCL. The way to do it is quite simple. It only consists in creating an invariant whose context will be the entity type in the opposite side of the cardinality constraint, for each cardinality constraint with a lower multiplicity greater than zero. The same has to be done with cardinality constraints with an upper multiplicity distinct than the asterisk (*).

As an example, if we go back to the Fig. 4.8, there is a cardinality constraint whose upper multiplicity is 1 (the one in the side of Meeting). Therefore, the new schema rule is as follows:

```
context Room inv:  self.meeting->size()<=1
```

Thus, this schema rule will be processed according to the steps explained before and the obtained links will be added into the computation of the values for the complex measures of the next section.

4.2.3 Complex Measures

The behavioral schema contains a set of event types. We adopt the view that events can be modeled as particular entity types [89]. Event types have characteristics, constraints, and effects. The characteristics of an event are its attributes and the associations in which it participates. The constraints are the conditions that events must satisfy to occur.

Each event type has an operation called *effect()* that gives the effect of an event occurrence. The effect is declaratively defined by the postcondition of the operation, which is specified in OCL (see chp. 11 of [87]).

We denote by $\mathcal{C}$ (Schema Rules) the set of constraints, derivation rules and pre- and post-conditions. Each rule $c \in \mathcal{C}$ is defined in the context of an entity type, denoted by $context(c)$. In OCL, each rule c consists of a set of OCL expressions (see OCL [85]) which we denote by $expr(c)$. An expression exp may refer to several entity types which are denoted by $members(exp)$. The set of entity types that are referred to in one or more expressions of a rule c is denoted by $ref(c)$.

We also include in $\mathcal{C}$ the schema rules corresponding to the equivalent OCL invariants of the graphical cardinality constraints in UML. For example, in Fig. 4.9 the cardinality *1..** between Company and Employee is transformed into the invariant:

```
context Company inv:  self.employee->size()>=1
```

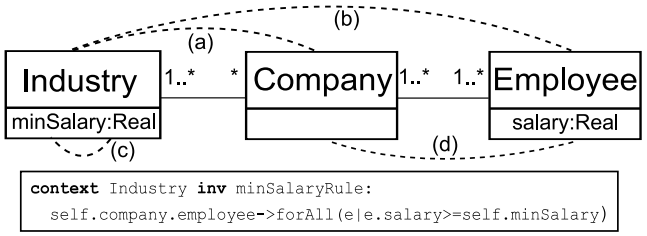
A special kind of OCL expression is the navigation expression that define a schema navigation from an entity type to another through an association (see *NavigationCallExp* of OCL in [85]). We use $expr_{nav}(c)$ to indicate the navigation expressions inside a rule $c \in \mathcal{C}$. Such expressions only contain two entity types as its participants, i.e. the *source* entity type and the *target* one (see the example in Fig. 4.9). The navigation from the source to the target is specified in OCL with a dot expression, which connects the source and the target with a dot symbol: `source.target`.

Notation		Definition
$context(c)$	=	$e \in \mathcal{E} \mid c \in \mathcal{C} \wedge c\ DefinedIn\ e$
$members(exp)$	=	$\{e \in \mathcal{E} \mid e$ is a participant of $exp\}$
$expr(c)$	=	$\{expr \mid expr$ is contained in $c\}$
$ref(c)$	=	$\cup_{exp \in expr(c)}\{members(exp)\}$
$expr_{nav}(c)$	=	$\{expr \in expr(c) \mid expr$ is a navigation expression$\}$
$nav_{expr}(c)$	=	$\cup_{exp \in expr_{nav}(c)}\{\{e, e'\} \subset \mathcal{E} \mid \{e, e'\} = members(exp)\}\}$
$nav_{context}(c)$	=	$\{\{e, e'\} \subset \mathcal{E} \mid e = context(c) \wedge e' \in ref(c)\}$
$nav(c)$	=	$nav_{context}(c) \cup\ nav_{expr}(c)$
$rconn(e)$	=	$\uplus_{sr \in \mathcal{C}}\{e' \in \mathcal{E} \mid \{e, e'\} \subset nav(c)\}$[2]
$rconn_{inh}(e)$	=	$rconn(e) \uplus \{rconn_{inh}(e') \mid e' \in par(e)\}$

Table 4.2. Definition of extended metrics.

[2]Note that "$\uplus$" denotes the multiset (or bag) union that produces a multiset as in $\{a, b\} \uplus \{a\} = \{a, a, b\}$. Apart from it, we also force that $rconn(e) = \emptyset$ (empty set) if $conn_{inh}(e) = \emptyset$.

We denote by $nav_{expr}(c)$ the set of pairs that participate in the navigation expressions of $c \in \mathcal{C}$. We also denote by $nav_{context}(c)$ the sets of pairs of entity types composed by the context of the rule c and every one of the participant entity types of such rule ($e \in ref(c)$). Finally, we define $nav(c)$ as the union of $nav_{context}(c)$ with $nav_{expr}(c)$ and, $rconn(e)$ as the multiset of entity types that compose a pair with e in $nav(c)$. Note that since we use $\uplus$, $rconn(e)$ may contain duplicates because it takes into account each rule c and an entity type e can be related to another one e' in two or more different rules. Intuitively, $rconn(e)$ is the multiset of entity types to which an entity type e is connected through schema rules.



$context$(minSalaryRule) = Industry
$expr_{nav}$(minSalaryRule) = {self.company, company.employee}
ref(minSalaryRule) = {Industry, Company, Employee}
$nav_{context}$(minSalaryRule) = {{Industry, Industry}, {Industry, Company}, {Industry, Employee}}
nav_{expr}(minSalaryRule) = {{Industry, Company}, {Company, Employee}}
nav(minSalaryRule) = {{Industry, Industry}, {Industry, Company}, {Company, Employee}, {Industry, Employee}}

Figure 4.9. Example of navigations of minSalaryRule. Dashed lines (a), (b) and (c) represent the elements in $nav_{context}$(minSalaryRule) while (d) and (a) are the connections through navigation expressions (see nav_{expr}(minSalaryRule)).

4.3 Importance-Computing Methods

The measures that allow to structure and index large conceptual schemas are the backbone that allows constructing methods to compute the importance of the elements of such schemas. This section presents a set of importance-computing methods to obtain the specific relevance of entity and event types according to their definition in the schema. These methods were deeply studied in [125]. The adaption of these methods to compute the importance of relationship types is fully detailed in [129].

4.3.1 Importance-computing Principles

According to the experience obtained after the review of relevance-computing methods in Ch. 3, it is possible to say that most of the methods in the literature are based on subjective approaches. A big number of solutions are evaluated comparing the results obtained by them and the solutions provided by an expert (or a set of them) in the information area where the conceptual schema defines the concepts of an information system. Such expert, also known as *oracle*, indicates the major elements in the schema, sometimes in a ranked list, according to his or her own experience.

This kind of evaluation that we can name as *subjective similarity* tends to a deviation

produced by the inherent subjectivity of the validation. Roughly speaking, two *oracles* in the same area can have different points of view even if they work or have experience in the same aspects of the elements of a conceptual schema. Furthermore, to have the possibility of relying on an *oracle* that collaborate in the process of the application of a method to compute the most important schema elements is not possible in many situations.

Thus, there exists the need of objective evaluations that do not change according to the influence of external factors like stakeholders of the information system. To reach this goal we present a set of objective metrics that only take into account the information represented in the conceptual schema. This way, the figure of the *oracle* can be avoided. The first point is the definition of an axiom we follow along this thesis, and that has been studied in other research areas like text searching or document indexing in information retrieval or data mining. We call it the principle of *high appearance* of elements in a scope, which is defined as follows:

Definition 4.3.1. *(Principle of High Appearance)*
The more occurrences an item/element has in an scope, the more important such item becomes.

According to the definition, the importance of an element has a proportional relation to the frequency with which such element appears in a scope. In the scope of conceptual schemas we redefine such principle for entity and event types:

Definition 4.3.2. *(Principle of High Appearance in Conceptual Schemas)*
The most important entity and event types of a conceptual schema are those that have the greater number of attributes, participate in a major number of associations, have more ascendants and descendants, appear in the structure of schema rules, and definitely, have a bigger participation in the conceptual schema than others.

From the other point of view, the most important entity types are those that the designer of the conceptual schema requires a bigger effort and produces more information in the schema to define them.

Since the principle of *high appearance* induces that the most important elements are those that the conceptual schema has more information about, the relevance of such elements directly depends on the requirements. This way we have an objective approach based on schema metrics to denote the relevant elements, which has a component of subjectivity according to the information added by modelers and requirements engineers. Therefore, our approach is sufficiently objective to avoid deviations, but includes the required subjectivity to be useful.

4.3.2 Basic Methods

This section presents the methods from the literature that were selected to be included in the study of this thesis. These methods are based on occurrence counting and on link analysis. A brief description was done at Ch. 3 about the state of the art in the field of dealing with large conceptual schemas.

As such methods were principally applied to entity-relationship schemas, the definitions here have been adapted with minor changes to accomplish the needs of the UML modeling language. The knowledge provided by OCL will be included in the extended versions of the methods in next sections.

Our purpose here is to formally describe a subset of methods in order to extend them in next sections. Thus, the next are the base versions of the methods, using the information gathered into the previously mentioned metrics —a subset of the structural schema including entity and relationship types, attributes and generalizations.

The Connectivity Counter (CC)

The first method we present was firstly introduced by Moody and Flitman in [80]. They suggest that central entities should be chosen as the entities of highest business importance —the *core* business concepts in the model. Of course, *business importance* is quite a subjective concept, and requires human judgment. However a useful heuristic for identifying central entities is to identify entities with the most relationships. Usually the most highly connected entities are also the most important entities from a business viewpoint.

Following these indications we define the Connectivity Counter method as the method that computes the importance of each entity type as the number of relationships it participates in. Formally:

$$I_{CC}(e) = |assoc(e)|$$

The schema shown in Fig. 4.10 contains six entity types representing information about a simple store with products, customers and more. The application of this method results in the values shown at Table 4.3 for the importance of the entity types presented in Fig. 4.10. Then, the most relevant entities are Customer and Product.

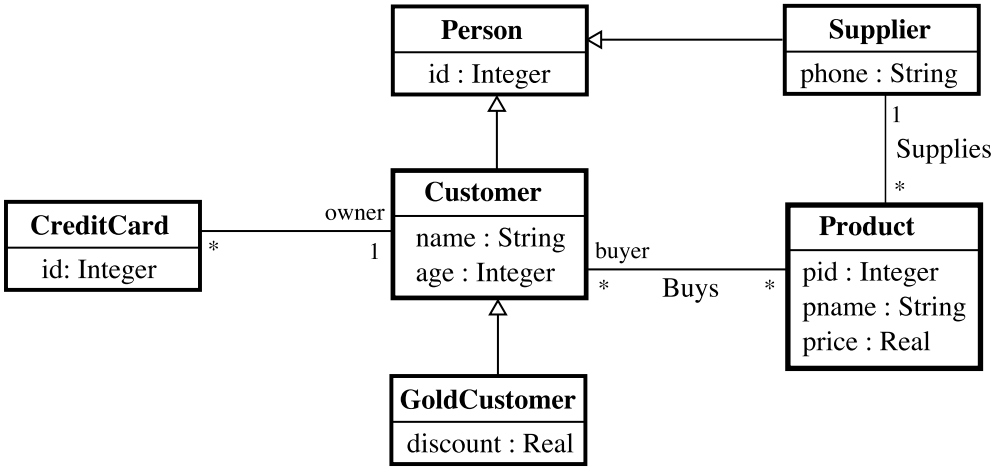


Figure 4.10. Example of schema.

e	$I_{CC}(e)$
CreditCard	**1**
Customer	**2**
GoldCustomer	**0**
Person	**0**
Product	**2**
Supplier	**1**

Table 4.3. Results for CC applied to example of Fig 4.10.

The Simple Method (SM)

This method was introduced in [121] (called m_0) and takes into account only the number of directly connected characteristics of each entity type. Formally, the importance $I_{SM}(e)$ of an entity type e is defined as:

$$I_{SM}(e) = |par(e)| + |chi(e)| + |attr(e)| + |assoc(e)|$$

Therefore, the importance of an entity type directly depends on the number of directly connected ascendants and descendants it has, the number of attributes it owns, and the number of participations in relationship types it does.

The values obtained after the application of the Simple Method to the previous schema are indicated in Table 4.4. There, it is possible to check that the most important entity type is Customer followed by Product.

e	$\|par(e)\|$	$\|chi(e)\|$	$\|attr(e)\|$	$\|assoc(e)\|$	$I_{SM}(e)$
CreditCard	0	0	1	1	**2**
Customer	1	1	2	2	**6**
GoldCustomer	1	0	1	0	**2**
Person	0	2	1	0	**3**
Product	0	0	3	2	**5**
Supplier	1	0	1	1	**3**

Table 4.4. Results for SM applied to example of Fig 4.10.

The Weighted Simple Method (WSM)

This is a variation to the simple method that assigns a strength to each kind of component of knowledge in the equation, such that the higher the strength, the greater the importance of such component [25]. The definition of importance here is:

$$I_{WSM}(e) = q_{inh}(|par(e)| + |chi(e)|) + q_{attr}|attr(e)| + q_{assoc}|assoc(e)|$$

where q_{attr} is the strength for attributes, q_{inh} is the strength for generalization/specialization relationships, and q_{assoc} is the strength for associations. Each of them with values in the interval *[0,1]*. Concretely, we have selected the same strengths than the originals in [25]: $q_{attr} = 1$, $q_{inh} = 0.6$, and $q_{assoc} = 0.4$.

The results of the application of the weighted simple method to the previous example are shown in Table 4.5. Again, the most relevant entity type is Customer, followed by Product.

e	$q_{inh}(\|par(e)\| + \|chi(e)\|)$	$q_{attr}\|attr(e)\|$	$q_{assoc}\|assoc(e)\|$	$\boldsymbol{I_{WSM}(e)}$
CreditCard	0.6x0	1x1	0.4x1	**1.4**
Customer	0.6x2	1x2	0.4x2	**4**
GoldCustomer	0.6x1	1x1	0.4x0	**1.6**
Person	0.6x2	1x1	0.4x0	**2.2**
Product	0.6x0	1x3	0.4x2	**3.8**
Supplier	0.6x1	1x1	0.4x1	**2**

Table 4.5. Results for WSM applied to example of Fig 4.10.

The Transitive inheritance Method (TIM)

This is a variation of the simple method taking into account both directly defined features and inherited ones (see m_5 in [121]). For each entity type the method computes the number of ascendants and descendants and all specified attributes and accessible associations from it or any of its ascendants. Formally:

$$I_{TIM}(e) = |par_{inh}(e)| + |chi_{inh}(e)| + |attr_{inh}(e)| + |assoc_{inh}(e)|$$

As before, the results of the application of the TIM method to the previous example are shown in Table 4.6. Here, the most important entity type is GoldCustomer, followed by Customer and Product.

One of the problems of this method is that it gives more relevance to those entity types deeper in a hierarchy because such entities obtain the value (e.g., inherited attributes and participations in relationships) of their parents. That's why GoldCustomer is more relevant than Customer.

e	$\|par_{inh}(e)\|$	$\|chi_{inh}(e)\|$	$\|attr_{inh}(e)\|$	$\|assoc_{inh}(e)\|$	$\boldsymbol{I_{TIM}(e)}$
CreditCard	0	0	1	1	**2**
Customer	1	1	3	2	**7**
GoldCustomer	2	0	4	2	**8**
Person	0	3	1	0	**4**
Product	0	0	3	2	**5**
Supplier	1	0	2	1	**4**

Table 4.6. Results for TIM applied to example of Fig 4.10.

EntityRank (ER)

The EntityRank method [120, 121] is based on link analysis following the same approach than Google's PageRank [21]. Roughly, each entity type is viewed as a state and each association between entity types as a bidirectional transition between them.

The importance of an entity type is the probability that a random surfer is at that entity type with random jumps (q component) or by navigation through relationships ($1-q$ component). Therefore, the resulting importance of the entity types correspond to the stationary probabilities of the Markov chain, given by:

$$I_{ER}(e) = \frac{q}{|\mathcal{E}|} + (1-q) \sum_{e' \in conn(e)} \frac{I_{ER}(e')}{|conn(e')|}$$

Such formulation produces a system of equations. Concretely, for the example of Fig. 4.10 the equation system would be as follows:

$$\begin{aligned}
I_{ER}(CreditCard) &= \tfrac{q}{6} + (1-q)\left(\tfrac{I_{ER}(Customer)}{2}\right) \\
I_{ER}(Customer) &= \tfrac{q}{6} + (1-q)\left(I_{ER}(CreditCard) + \tfrac{I_{ER}(Product)}{2})\right) \\
I_{ER}(GoldCustomer) &= \tfrac{q}{6} \\
I_{ER}(Person) &= \tfrac{q}{6} \\
I_{ER}(Product) &= \tfrac{q}{6} + (1-q)\left(\tfrac{I_{ER}(Customer)}{2} + I_{ER}(Supplier)\right) \\
I_{ER}(Supplier) &= \tfrac{q}{6} + (1-q)\left(\tfrac{I_{ER}(Product)}{2}\right)
\end{aligned}$$

It is important to note that the relevance of an entity comes from the relevance of the entities connected to it. The relevance flows through associations. For the case of Customer, its relevance come from CreditCard and from Product. As Product is connected with two entity types, a fragment of its relevance (the middle) goes to each of its connected entities (Customer and Supplier).

To compute the importance of the entity types we need to solve this equation system. If we fix that $\sum_{e \in \mathcal{E}} I_{ER}(e) = 1$ then the results obtained are shown in Table 4.7. We have chosen a $q = 0.15$, which is a common value in the literature.

e	$\boldsymbol{I_{ER}(e)}$
CreditCard	**0.16**
Customer	**0.30**
GoldCustomer	**0.04**
Person	**0.04**
Product	**0.30**
Supplier	**0.16**

Table 4.7. Results for ER applied to example of Fig 4.10.

BEntityRank (BER)

This is a variation of the previous method specifying that the probability of randomly jumping to each entity type is not the same for each entity type, but it depends on the number of its attributes [120, 121]. The higher the number of attributes, the higher the probability to randomly jump to that entity type. That is:

$$I_{BER}(e) = q\frac{attr(e)}{|\mathcal{A}|} + (1-q)\sum_{e' \in conn(e)} \frac{I_{BER}(e')}{|conn(e')|}$$

As in the case of EntityRank, the formulation produces a system of equations. In this case the system would be as follows:

$$\begin{aligned}
I_{BER}(CreditCard) &= q\tfrac{1}{9} + (1-q)\left(\tfrac{I_{BER}(Customer)}{2}\right) \\
I_{BER}(Customer) &= q\tfrac{2}{9} + (1-q)\left(I_{BER}(CreditCard) + \tfrac{I_{BER}(Product)}{2})\right) \\
I_{BER}(GoldCustomer) &= q\tfrac{1}{9} \\
I_{BER}(Person) &= q\tfrac{1}{9} \\
I_{BER}(Product) &= q\tfrac{3}{9} + (1-q)\left(\tfrac{I_{BER}(Customer)}{2} + I_{BER}(Supplier)\right) \\
I_{BER}(Supplier) &= q\tfrac{1}{9} + (1-q)\left(\tfrac{I_{BER}(Product)}{2}\right)
\end{aligned}$$

It is important to note that if an entity type is not connected to others through associations, its relevance only consists on its percentage of attributes multiplied by the coefficient of random jumps (q).

Similarly than with EntityRank, to compute the importance of the entity types we need to solve this equation system. We fix that $\sum_{e \in \mathcal{E}} I_{BER}(e) = 1$ then the results obtained are shown in Table 4.8. We have already chosen a $q = 0.15$, which is a common value in the literature.

One more time, the most important pair of entity types are Product and Customer. Therefore a focused view of the schema shown in Fig 4.10 could contain such two entities and the relationship type Buys placed between them.

e	$\boldsymbol{I_{BER}(e)}$
CreditCard	**0.15**
Customer	**0.31**
GoldCustomer	**0.02**
Person	**0.02**
Product	**0.33**
Supplier	**0.16**

Table 4.8. Results for BER applied to example of Fig 4.10.

CEntityRank (CER)

Finally, the method that we call CEntityRank (m_4 in [121]) follows the same idea than EntityRank and BEntityRank, but including the generalization relationships. Each generalization between ascendants and descendants is viewed as a bidirectional transition, as in the case of associations. Formally:

$$I_{CER}(e) = q_1 \frac{attr(e)}{|\mathcal{A}|} + q_2 \sum_{e' \in gen(e)} \frac{I_{CER}(e')}{|gen(e')|} + (1 - q_1 - q_2) \sum_{e'' \in conn(e)} \frac{I_{CER}(e'')}{|conn(e'')|}$$

The formulation produces a system of equations as in the other two methods based on link analysis. In this case the system would be as follows:

$$\begin{aligned}
I_{CER}(CreditCard) &= q_1 \tfrac{1}{9} + (1 - q_1 - q_2) \left(\tfrac{I_{CER}(Customer)}{2} \right) \\
I_{CER}(Customer) &= q_1 \tfrac{2}{9} + q_2 \left(\tfrac{I_{CER}(Person)}{2} + I_{CER}(GoldCustomer) \right) \\
&\quad + (1 - q_1 - q_2) \left(I_{CER}(CreditCard) + \tfrac{I_{CER}(Product)}{2}) \right) \\
I_{CER}(GoldCustomer) &= q_1 \tfrac{1}{9} + q_2 \left(\tfrac{I_{CER}(Customer)}{2} \right) \\
I_{CER}(Person) &= q_1 \tfrac{1}{9} + q_2 \left(\tfrac{I_{CER}(Customer)}{2} + I_{CER}(Supplier) \right) \\
I_{CER}(Product) &= q_1 \tfrac{3}{9} + (1 - q_1 - q_2) \left(\tfrac{I_{BER}(Customer)}{2} + I_{BER}(Supplier) \right) \\
I_{CER}(Supplier) &= q_1 \tfrac{1}{9} + q_2 \left(\tfrac{I_{CER}(Person)}{2} \right) + (1 - q_1 - q_2) \left(\tfrac{I_{BER}(Product)}{2} \right)
\end{aligned}$$

Similarly than with EntityRank and BEntityRank, to compute the importance of the entity types we need to solve this equation system. We require that $\sum_{e \in \mathcal{E}} I_{CER}(e) = 1$, and then the results obtained are shown in Table 4.9. We have chosen $q_1 = 0.1$ and $q_2 = 0.2$, which are some good values as indicated in [121].

As in the previous methods the most important pair of entity types are Product and Customer. However, we have obtained the similar rankings because the schema shown in Fig 4.10 is very small. The methods presented here must be tested with large schemas to bring out the differences between them.

e	$\boldsymbol{I_{CER}(e)}$
CreditCard	**0.13**
Customer	**0.32**
GoldCustomer	**0.05**
Person	**0.08**
Product	**0.27**
Supplier	**0.16**

Table 4.9. Results for CER applied to example of Fig 4.10.

4.3.3 Extended Methods

This section presents our approach to extend the methods introduced in Section 4.3.2. Such extensions take into account the schema rules, including the conversion of cardinality constraints into OCL invariants, as we introduced in Sect. 4.2.2.

We formally define the new components of each method and use a extended schema with a set of schema rules to increase the understandability of the methods. Such example is a reformulation of the one in Fig. 4.10, shown at Fig. 4.11.

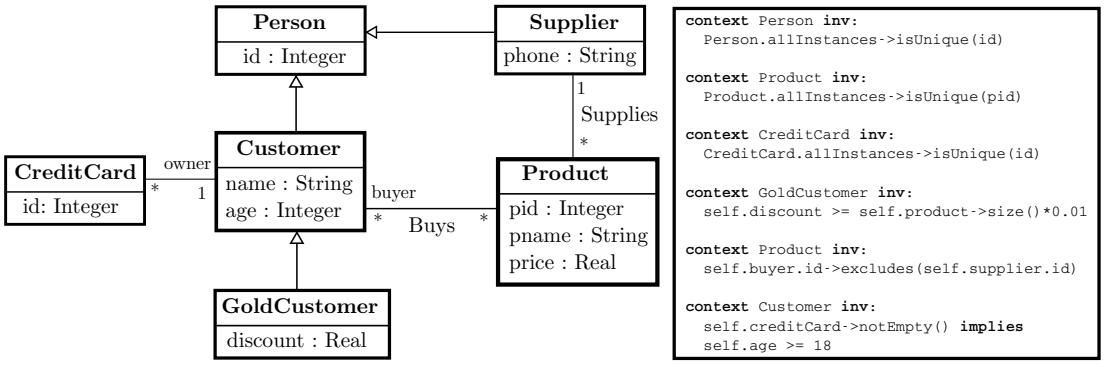


Figure 4.11. Extension of the schema in Fig.4.10 with some OCL invariants.

The conversion of the cardinality constraints of the schema in Fig. 4.11 is as follows:

```
context Product inv:  self.supplier->size()>=1

context Product inv:  self.supplier->size()<=1

context CreditCard inv:  self.employee->size()>=1

context CreditCard inv:  self.customer->size()<=1
```

It is important to note that we create a new schema rule defined in OCL for each cardinality constraint. A possible change here could be were the lower an upper values of a multiplicity have the same value, as in the previous example where the value is 1. Thus, the conversion will result in:

```
context Product inv:  self.supplier->size()=1

context CreditCard inv:  self.employee->size()=1
```

However, we select to convert as in the first case because if we would have a lower value greater than zero and an upper value distinct than asterisk, and such values were different, then the conversion in this case will produce a pair of schema rules, while if the values of the upper and the lower values are equal it produces only one schema rule. To be consistent, we prefer to create a schema rule for each upper or lower value.

The Connectivity Counter Extended (CC+)

Our extension to this method follows the same idea than the base version but also including the number of participations of each entity type in the navigation relationships extracted from the schema rules specification, i.e., derivation rules, invariants and pre- and postconditions (and cardinality constraints). On the other hand, we now take into account (in $|assoc(e)|$) the associations of each entity type e with the event types of the behavioral schema (in case of such events were defined). Formally:

$$I^{+}_{CC}(e) = |assoc(e)| + |rconn(e)|$$

Table 4.10 presents the obtained results once this method has been applied to example of Fig 4.11. One more time, we conclude that Product and Customer are the most relevant entity types.

e	$\|assoc(e)\|$	$\|rconn(e)\|$	$\boldsymbol{I^{+}_{CC}(e)}$
CreditCard	1	6	**7**
Customer	2	10	**12**
GoldCustomer	0	4	**4**
Person	0	0	**0**
Product	2	14	**16**
Supplier	1	4	**5**

Table 4.10. Results for CC+ applied to example of Fig 4.11.

Note that although the value of $|rconn(Person)|$ is not zero, there is a condition into the extended metrics to be consistent. Such condition states that the entity types that are not directly or indirectly (with inherited associations from their parents) connected through relationships with other entities ($conn_{inh}(e) = \emptyset$), must have a $rconn$ equal to the empty set. The cause of it is to avoid give importance to entity types from virtual connections (that should enforce real connections) when such entities are really unconnected.

The Simple Method Extended (SM+)

As in the extended version of the Connectivity Counter, the Simple Method has been extended including the number of participations of each entity type in the navigation relationships extracted from the schema rules specification (and cardinality constraints). We also take into account (in $|assoc(e)|$) the associations of each entity type e with the event types of the behavioral schema (in case of such events were defined). Formally:

$$I^{+}_{SM}(e) = |par(e)| + |chi(e)| + |attr(e)| + |assoc(e)| + |rconn(e)|$$

For instance, if both the extended version of the simple method, and also the simple method, were applied into the schema shown in Fig. 4.9, we would have I_{SM}(Company)=2

and I^{+}_{SM}(Company)=8, because $|par$(Company)$|=|chi$(Com-pany)$|=|attr$(Company)$|$ =0, also $|assoc$ (Company)|=2, and $|rconn$(Company)|=6, of which two come for the invariant (minSalaryRule) and the other four from the OCL equivalent to the cardinality constraints of multiplicity *1..** in its relationships with Industry and Employee.

The values obtained after the application of the extended version of the simple method to the previous schema shown at Fig 4.11 are indicated in Table 4.11. There, it is possible to check that the most important entity type is Customer followed by Product.

e	$\|par(e)\|$	$\|chi(e)\|$	$\|attr(e)\|$	$\|assoc(e)\|$	$\|rconn(e)\|$	$\boldsymbol{I^{+}_{SM}(e)}$
CreditCard	0	0	1	1	6	**8**
Customer	1	1	2	2	10	**16**
GoldCustomer	1	0	1	0	4	**6**
Person	0	2	1	0	0	**3**
Product	0	0	3	2	14	**19**
Supplier	1	0	1	1	4	**7**

Table 4.11. Results for SM+ applied to example of Fig 4.11.

As the reader can observe, it is possible to say that the Simple Method is also a extended version of the Connectivity Counter (in both base and extended cases).

The Weighted Simple Method Extended (WSM+)

Our extension to this method consists on adding the *schema rules navigation* component to the importance computation. In the same way as the other components, we selected a strength (q_{rule}) to specify the weight of navigation relationships in the schema rules. The definition is now:

$$I^{+}_{WSM}(e) = q_{inh}(|par(e)| + |chi(e)|) + q_{attr}|attr(e)| + q_{assoc}|assoc(e)| + q_{rule}|rconn(e)|$$

We use the same strengths than in the base version of the method. Concretely $q_{attr} = 1$, $q_{inh} = 0.6$, and $q_{assoc} = 0.4$. Furthermore, we use for q_{rule} the same strength as q_{assoc}. Although it is possible to use a different strength for q_{rule}, since the behaviour of $|rconn(e)|$ is similar than for $|assoc(e)|$ due to both represent participations of entities, we decided to use the same strength.

The results of the application of the weighted simple method to the example in Fig 4.11 are shown in Table 4.12. The most relevant entity type is already Product, followed by Customer. As the reader can observe, this pair of entities are selected by all the methods presented here as the most important ones. The littleness of the example is the main cause of this behaviour.

e	$q_{inh}(\|par(e)\| + \|chi(e)\|)$	$q_{attr}\|attr(e)\|$	$q_{assoc}\|assoc(e)\|$
CreditCard	0.6x0	1x1	0.4x1
Customer	0.6x2	1x2	0.4x2
GoldCustomer	0.6x1	1x1	0.4x0
Person	0.6x2	1x1	0.4x0
Product	0.6x0	1x3	0.4x2
Supplier	0.6x1	1x1	0.4x

e	$q_{rule}\|rconn(e)\|$	$\boldsymbol{I^{+}_{WSM}(e)}$
CreditCard	0.4x6	**3.8**
Customer	0.4x10	**8**
GoldCustomer	0.4x4	**3.2**
Person	0.4x0	**2.2**
Product	0.4x14	**9.4**
Supplier	0.4x4	**3.6**

Table 4.12. Results for WSM+ applied to example of Fig 4.11.

The Transitive Inheritance Method Extended (TIM+)

In the same way as with the previous methods, we extend it with the *schema rules navigation* component. This time the computation of such component also takes into account the *rconn* measure of the ancestors:

$$I^{+}_{TIM}(e) = |par_{inh}(e)| + |chi_{inh}(e)| + |attr_{inh}(e)| + |assoc_{inh}(e)| + |rconn_{inh}(e)|$$

The results of the application of the TIM+ method to the previous example are shown in Table 4.13. Here, the most important entity type is GoldCustomer, followed by Product and Customer. Note that since GoldCustomer is a descendant of Customer, its values for the measures of inherited ascendants and connections through associations and schema rules are the ones from Customer plus the ones related to GoldCustomer itself. As a consequence, GoldCustomer becomes the most relevant entity type.

e	$\|par_{inh}(e)\|$	$\|chi_{inh}(e)\|$	$\|attr_{inh}(e)\|$
CreditCard	0	0	1
Customer	1	1	3
GoldCustomer	2	0	4
Person	0	3	1
Product	0	0	3
Supplier	1	0	2

e	$\|assoc_{inh}(e)\|$	$\|rconn_{inh}(e)\|$	$\boldsymbol{I^+_{TIM}(e)}$
CreditCard	1	6	**8**
Customer	2	10	**17**
GoldCustomer	2	14	**22**
Person	0	0	**4**
Product	2	14	**19**
Supplier	1	4	**8**

Table 4.13. Results for TIM+ applied to example of Fig 4.11.

EntityRank Extended (ER+)

In our extension to the EntityRank method we add a new component to the formula in order to jump not only to the connected entity types but also to the virtually connected ones through the navigation relationships uncovered in the schema rules. The definition is now:

$$I^+_{ER}(e) = \frac{q}{|\mathcal{E}|} + (1-q)\left(\sum_{e' \in conn(e)} \frac{I^+_{ER}(e')}{|conn(e')|} + \sum_{e'' \in rconn(e)} \frac{I^+_{ER}(e'')}{|rconn(e'')|}\right)$$

If we fix that $\sum_{e \in \mathcal{E}} I^+_{ER}(e) = 1$ then the results obtained are shown in Table 4.14. Remember that we have chosen a $q = 0.15$, which is a common value in the literature.

e	$\boldsymbol{I^+_{ER}(e)}$
CreditCard	**0.15**
Customer	**0.25**
GoldCustomer	**0.11**
Person	**0.03**
Product	**0.34**
Supplier	**0.12**

Table 4.14. Results for ER+ applied to example of Fig 4.11.

BEntityRank Extended (BER+)

Our extension for the BEntityRank method is in the same way as in ER+. The difference between the formula of ER+ and BER+ is that BER+ takes into account the values of the attribute measure of BEntityRank. The definition is:

$$I^+_{BER}(e) = q\frac{attr(e)}{|\mathcal{A}|} + (1-q)\left(\sum_{e'\in conn(e)} \frac{I^+_{BER}(e')}{|conn(e')|} + \sum_{e''\in rconn(e)} \frac{I^+_{BER}(e'')}{|rconn(e'')|}\right)$$

Similarly than other methods based on link analysis, to compute the importance of the entity types we need to solve an equation system (the same as in BER but including the new components provided by *rconn* measure). We fix that $\sum_{e\in\mathcal{E}} I^+_{BER}(e) = 1$ then the results obtained are shown in Table 4.15. We have already chosen a $q = 0.15$, which is a common value in the literature.

e	$\boldsymbol{I^+_{BER}(e)}$
CreditCard	**0.15**
Customer	**0.26**
GoldCustomer	**0.10**
Person	**0.02**
Product	**0.36**
Supplier	**0.11**

Table 4.15. Results for BER+ applied to example of Fig 4.11.

CEntityRank Extended (CER+)

One more time, our extension includes the uncovered navigations of the schema rules as bidirectional transitions for the random surfer. The new definition is the same as for CER but including the measure of the knowledge provided by the schema rules:

$$\begin{aligned} I^+_{CER}(e) \;=\; & q_1\frac{attr(e)}{|\mathcal{A}|} + q_2 \sum_{e'\in gen(e)} \frac{I^+_{CER}(e')}{|gen(e')|} \\ & + (1-q_1-q_2)\left(\sum_{e''\in conn(e)} \frac{I^+_{CER}(e'')}{|conn(e'')|} + \sum_{e'''\in rconn(e)} \frac{I^+_{CER}(e''')}{|rconn(e''')|}\right) \end{aligned}$$

We already fix that $\sum_{e\in\mathcal{E}} I^+_{CER}(e) = 1$ then the results obtained are shown in Table 4.16. We have chosen $q_1 = 0.1$ and $q_2 = 0.2$, which are some good values as indicated in [121].

As in the previous methods the most important pair of entity types are Product and Customer, and we have obtained similar rankings because the schema shown in Fig 4.11 is very

small. The methods presented here must be tested with large schemas to bring out the differences between them.

e	$I^{+}_{CER}(e)$
CreditCard	**0.12**
Customer	**0.27**
GoldCustomer	**0.12**
Person	**0.06**
Product	**0.31**
Supplier	**0.12**

Table 4.16. Results for CER+ applied to example of Fig 4.11.

4.3.4 Comparison between Methods

In the previous sections the algorithms that conform a sample of the existing methods in the literature about how to compute the importance of conceptual schema's entity types have been explained. Here we want to discover some differences between them.

After the study of each method and measure, it is possible to detect two main characteristics. On one hand, there are two types of methods —those based on occurrence counting and those that follow the link analysis approach. Take a look at Table 4.17 to see this classification.

Occurrence Counting	Link Analysis
CC	ER
SM	BER
WSM	CER
TIM	
CC+	ER+
SM+	BER+
WSM+	CER+
TIM+	

Table 4.17. Classification of selected methods according to their approach.

Methods based on occurrence counting are similar to those methods in the field of text searching that count the frequency of words to select the most important ones. These methods count for each entity type the number of related elements to it (or owned by it, as in the case of attributes), plus the number of occurrences of such entity type in schema rules and in the conversion of cardinality constraints to rules. This way, we follow the principle of high appearance —the more occurrences an item has in an scope, the more important such item becomes.

Methods based on link analysis take into account the importance of the other related elements to the current one because the importance of the current is an addition of fragments of the importance of the others. Remember the example in Fig 3.9 of Ch. 3. Thus, methods based on link analysis approach need special iterative algorithms to solve the importance computation based on a system of equations.

Such iterative methods attempt to solve a problem by finding successive approximations to the solution starting from an initial point. These iterative solvers, like the Jacobi algorithm, the power method or the inverse power method (see [124]), are computationally expensive and slower than the methods based on occurrence counting. However, solutions obtained from these methods have better results taking into account lower amounts of knowledge, as we will see in next sections.

On the other hand, there exists another difference between base methods and extended ones: the amount of knowledge taken into account in the computation process of the relevance of entity types. As explained before, methods to calculate the relevant entity types of a conceptual schema use some metrics whose values are gathered from the structural subschema. Our approach introduces some new measures to convert the knowledge within the whole conceptual schema (including the behavioral subschema) into meters for relevance. Table 4.18 shows the different elements of knowledge of the conceptual schema taken into account for each of the (base and extended) methods. Cells marked with an **x** indicate that the algorithm in that row uses the information of the column.

First four columns include entity and relationship types, generalization/specialization relationships and attributes in the structural schema. In the base versions, the methods only use the knowledge of such columns depicted in the structural schema. Extended versions also use the entity and relationship types, attributes and generalizations included in the behavioral schema, and therefore the schema rules and elements of the complete conceptual schema. Last columns of Table 4.18 represent the knowledge extracted from the schema rules, including derivation rules, invariants, pre- and postconditions and, of course, the conversion of cardinality constraints into schema rules.

First rows in the table shows the base versions of the methods. Such methods only use information from the structural schema. As the event types are part of the behavioral schema, they are not applicable in this part of the table. Finally, last rows are the extended versions of the algorithms in the first rows. The gap in the rows of *CC*, *ER* and *BER* methods referencing the use of generalization relationships and attributes is maintained in the extended versions *CC+*, *ER+* and *BER+* to follow the same approach than in the base versions. In the case of ER+ and *BER+*, the complete link analysis-based method that takes into account the whole knowledge is the extension of the CEntityRank (*CER+*).

It is clear that the extended versions of the selected methods from the literature take into account more knowledge from the conceptual schema than the base versions. Therefore, according to the *Principle of High Appearance in Conceptual Schemas*, the results obtained with the extended versions are more realistic because there is more amount of knowledge that contributes to calculate the importance of entity types.

Base Methods (Structural Schema)	Knowledge: Entity Types	Relationship Types	Generalization Relationships	Attributes	Cardinality Constraints	Derivation Rules	Invariants	Pre- and Postconditions	Event Types
CC	x	x							n/a
SM	x	x	x	x					n/a
WSM	x	x	x	x					n/a
TIM	x	x	x	x					n/a
ER	x	x							n/a
BER	x	x		x					n/a
CER	x	x	x	x					n/a
Extended Methods (Complete Schema)									
CC+	x	x			x	x	x	x	x
SM+	x	x	x	x	x	x	x	x	x
WSM+	x	x	x	x	x	x	x	x	x
TIM+	x	x	x	x	x	x	x	x	x
ER+	x	x			x	x	x	x	x
BER+	x	x		x	x	x	x	x	x
CER+	x	x	x	x	x	x	x	x	x

Table 4.18. Comparison of knowledge used between both base and extended versions of the selected methods.

4.3.5 Experimental Evaluation

We have implemented the seven methods described in the previous section, both the original and our extended versions. We have then evaluated the methods using three distinct case studies: osCommerce [118], the UML metaschema [84, 8], and EU-Rent [47]. The original methods have been evaluated with the input knowledge they are able to process: entity types, attributes, associations, and generalization/specialization relationships of the structural schemas.

	osCommerce	UML Metaschema	EU-Rent schema
Entity Types	84	293	65
Event Types	262	-	120
Attributes	209	93	85
Associations	183	377	152
General Constraints and Derivation Rules	204	161	117
Pre- and Post conditions	220	-	166

Table 4.19. Schema contents of the case studies.

For osCommerce and EU-Rent, the extended methods have been evaluated with the complete structural schema and the complete behavioral schema (including event types and their pre/post conditions). The schema of osCommerce comprises 84 entity types and 262 event types, 209 attributes, 183 associations, 204 general constraints and derivation rules, and 220 pre- and post conditions. The schema of EU-Rent contains 65 entity types and 120 event types, 85 attributes, 152 associations, 117 general constraints, and 166 pre- and post conditions.

For the UML metaschema there is no behavioral schema and therefore we have only used the complete structural schema. The version of the UML metaschema we have used comprises 293 entity types, 93 attributes, 377 associations, and 161 general constraints. Tab. 4.19 summarizes the characteristics of the three case studies. The OCL constraints corresponding to the cardinalities, taxonomies, and association classes are not included in the information of Tab. 4.19.

In case of two or more entity types get the same importance, our implementation is nondeterministic: it might rank first any of those. Some enhancements can be done to try to avoid ranking equally-important entity types in a random manner, like prioritizing those with a higher amount of attributes or relationships (or any other measure) in case of ties. However, this does not have an impact to our experimentation.

In the following, we summarize the two main conclusions we have drawn from the study of the result data.

Correlation Between the Original and the Extended Versions

We study the correlation between the original and the extended version for the previously described methods. Our research aims to know which methods give similar results in both versions.

Fig. 4.12 shows, for each method, the results obtained in the original and the extended versions for the osCommerce. The horizontal axis has a point for each of the 85 entity types of the structural schema, in descending order of their importance in the original version. The vertical axis shows the importance computed in both versions. The importance has been normalized such that the sum of the importances of all entity types in each method is 100.

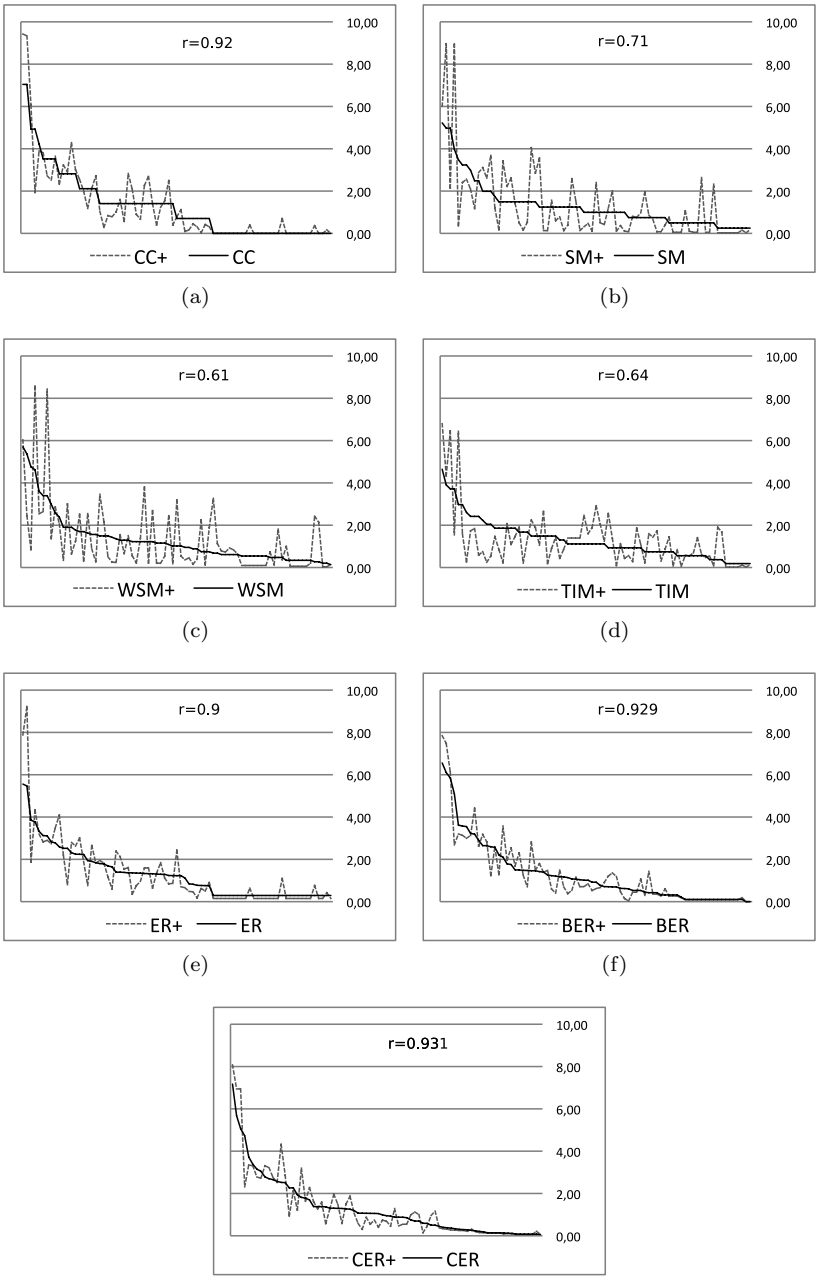


Figure 4.12. Comparison between base and extended methods applied to the osCommerce.

As shown in Fig. 4.12(g) the highest correlation between the results of both versions is for the CEntityRank (r=0.931), closely followed by the BEntityRank (r=0.929). The lowest correlation is for the Weighted Simple Method (r=0.61). Similar results are obtained for the UML metamodel. In this case the correlation between the two versions of the Weighted Simple

Method is 0.84 and that of the CEntityRank is 0.96. Table 4.20 summarizes the correlation results for the three case studies.

Our conclusion from this result is that the methods that produce more similar results in both versions are the BEntityRank and the CEntityRank. If this conclusion were confirmed by further experiments, the practical implication would be that, if we only have the fragment of the structural schema comprising the entity and relationship types, their attributes, and the specialization/generalization relationships, then CEntityRank and BEntityRank are the methods of choice. The reason is that using only those elements, the BEntityRank and CEntityRank methods give results more similar to those that would be obtained taking into account the more of the schema. That is, the more knowledge we take into account the better, but if we only have that fragment of the structural schema then the methods of choice are the CEntityRank and BEntityRank.

Methods	osCommerce	UML Metaschema	EU-Rent
CC - CC+	0.92	0.87	0.93
SM - SM+	0.71	0.85	0.89
WSM - WSM+	0.61	0.84	0.77
TIM - TIM+	0.64	0.94	0.71
ER - ER+	0.9	0.94	0.91
BER - BER+	0.93	0.95	0.94
CER - CER+	0.93	0.96	0.93

Table 4.20. Correlation coefficients between original and extended methods.

On the other hand, the methods based on link analysis (ER, BER and CER) are more constant than those based on occurrence counting (CC, SM, WSM and TIM). The main reason for this behavior is the recursive definition of its formulas. The link analysis methods need the importance of other entity types in order to compute the importance of an entity type. This dependency implies an iterative computation to achieve the convergence to a state where the importance flows are in equilibrium. In the case of occurrence counting methods, the computation of the relevance for an entity is totally independent from the other entity types.

This conclusion contrasts with the results reported in the previous work of [121], which, based on subjective evaluations given by evaluators, concludes that the method that gives the best results is the Simple Method. However, Fig. 4.12(b) shows that the result given by that method considerably changes when more schema knowledge is taken into account.

Variability of the Original and the Extended Versions

The second experimentation consists of the study of the correlation between original and extended versions separately. Our aim is to know whether it is possible to compare the results of the importance methods and to search for a common behavior. We analyze if the methods take into account the complete conceptual schema (extended version of methods) or only the structural schema (original methods).

Table 4.21, Tab. 4.22 and Tab. 4.23 show the correlation between each pair of methods (separately, originals and extended versions), in all case studies. It can be seen that, if we exclude the Transitive Inheritance Method (TIM) because it gives the worst results, the correlation in the original versions of the methods ranges from 0.59 to 0.99, while in the extended versions the range is from 0.81 to 0.99.

	I_{SM}	I_{WSM}	I_{TIM}	I_{ER}	I_{BER}	I_{CER}
I_{CC}	0.87	0.79	0.06	0.96	0.85	0.86
I_{SM}		0.98	0.15	0.82	0.79	0.92
I_{WSM}			0.16	0.73	0.77	0.90
I_{TIM}				0.06	0.07	0.11
I_{ER}					0.82	0.83
I_{BER}						0.91

	I^+_{SM}	I^+_{WSM}	I^+_{TIM}	I^+_{ER}	I^+_{BER}	I^+_{CER}
I^+_{CC}	0.99	0.97	0.23	0.93	0.82	0.81
I^+_{SM}		0.99	0.26	0.93	0.83	0.86
I^+_{WSM}			0.27	0.91	0.85	0.89
I^+_{TIM}				0.25	0.24	0.30
I^+_{ER}					0.84	0.84
I^+_{BER}						0.91

Table 4.21. Correlation coefficients between results of original and extended methods for the UML metaschema.

	I_{SM}	I_{WSM}	I_{TIM}	I_{ER}	I_{BER}	I_{CER}
I_{CC}	0.76	0.61	0.43	0.98	0.94	0.94
I_{SM}		0.97	0.79	0.74	0.87	0.88
I_{WSM}			0.79	0.59	0.78	0.76
I_{TIM}				0.40	0.54	0.61
I_{ER}					0.94	0.94
I_{BER}						0.97

	I^+_{SM}	I^+_{WSM}	I^+_{TIM}	I^+_{ER}	I^+_{BER}	I^+_{CER}
I^+_{CC}	0.99	0.99	0.78	0.98	0.92	0.92
I^+_{SM}		0.99	0.79	0.98	0.93	0.93
I^+_{WSM}			0.79	0.98	0.94	0.94
I^+_{TIM}				0.78	0.73	0.83
I^+_{ER}					0.94	0.93
I^+_{BER}						0.97

Table 4.22. Correlation coefficients between results of original and extended methods for the osCommerce.

	I_{SM}	I_{WSM}	I_{TIM}	I_{ER}	I_{BER}	I_{CER}
I_{CC}	0.88	0.71	0.06	0.99	0.97	0.97
I_{SM}		0.95	0.24	0.87	0.92	0.95
I_{WSM}			0.24	0.71	0.81	0.84
I_{TIM}				0.05	0.06	0.13
I_{ER}					0.97	0.97
I_{BER}						0.99

	I^+_{SM}	I^+_{WSM}	I^+_{TIM}	I^+_{ER}	I^+_{BER}	I^+_{CER}
I^+_{CC}	0.99	0.98	0.65	0.99	0.97	0.95
I^+_{SM}		0.99	0.66	0.99	0.97	0.98
I^+_{WSM}			0.66	0.97	0.96	0.98
I^+_{TIM}				0.66	0.61	0.69
I^+_{ER}					0.98	0.96
I^+_{BER}						0.96

Table 4.23. Correlation coefficients between results of original and extended methods for the EU-Rent.

The conclusion from this result is that the extended versions of the methods, excluding TIM, produce remarkably similar results, which does not happen in the original version. That is, the results obtained by extended versions have a lower degree of variance. This conclusion is also significant because it assures that the use of the Simple Method (extended version) whose computational cost is very low, and on the other hand it allows the incremental recalculation of the importance of entity types when the schema changes, produces *good-enough* results.

4.3.6 Extending the Target of Importance-Computing Methods

The visualization and the understanding of large conceptual schemas require the use of specific methods. These methods generate indexed, clustered, summarized or focused schemas that are easier to visualize and understand. Almost all of these methods require computing the importance of each element in the schema but, up to now, only the importance of entity types has been studied in the literature.

The computed importance induces an ordering of the elements, which plays a key role in the steps and result of the methods that deals with large schemas. In the previous sections, we have studied several methods to compute the importance of entity and event types because in this thesis we focus on the relevance of entity and event types. However, there exist other schema elements worthy of being studied because they play a key role in the specification of conceptual schemas. Fortunately, we can adapt the methods from Sect. 4.3.2 and Sect. 4.3.2 to be able to process those additional elements, as we presented in [129].

As an example, we can analyze the existing methods for measuring the importance of entity and event types, and then to adapt them to be able to work with relationship types. Our approach takes into account the knowledge defined in the schema about relationships, including

their participant entity or event types, the cardinality constraints of those participations, the general constraints of the schema, and the specification of behavioral events. All of them contribute to measure the importance of relationship types. Our approach transforms the schema by reifying the relationship types into entity types, es explained in Sect. 4.2.2.

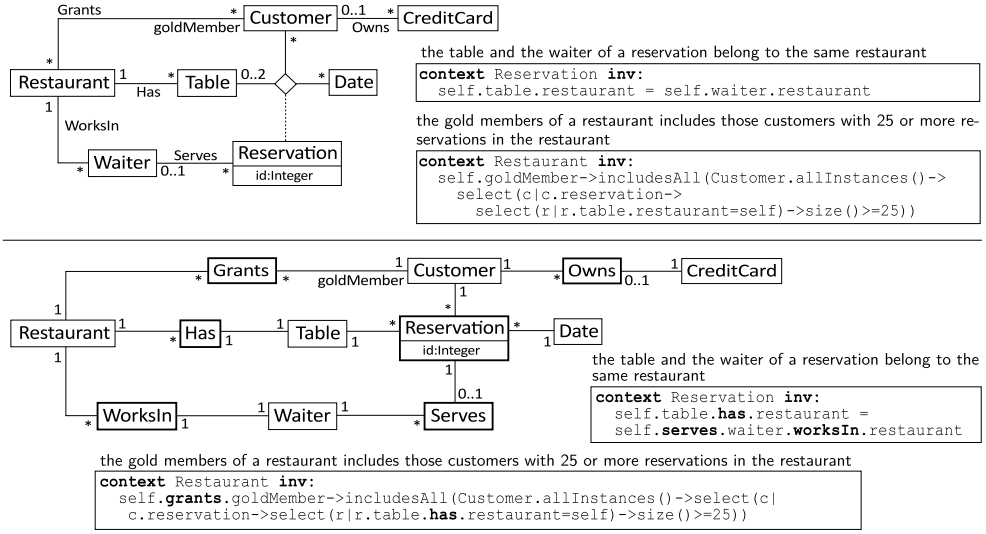


Figure 4.13. A fragment of conceptual schema (up) and its version with reifications (down).

Figure 4.13 (top) shows an example of conceptual schema with 6 associations (one of them, an association class of degree 3) and two OCL constraints that describe a fragment of an information system for reservations in restaurants. Figure 4.13 (bottom) shows the same schema with reifications (marked with bold rectangles). Note that the two constraints include the new navigations (bold text) to match with the new schema. Basically, a navigation $e \rightarrow e'$ has to be changed by including the navigation to the entity type that reifies the association in the middle of the expression, producing a navigation like $e \rightarrow e_r \rightarrow e'$. The reification of the schema produces 6 additional uniqueness constraints and a constraint to preserve the cardinality 0..2 in *Table*. These 7 constraints are not shown in Fig. 4.13 for the sake of simplicity.

As a result, we use the existing importance-computing methods from the literature to compute the importance of the entity and event types from the schema with reifications. Therefore, we compute the importance of the entity types e_r that are reifications of relationship types. Then, such importance is directly the importance of each relationship type $r \in \mathcal{R}$ of the original schema because each e_r is the representation of a relationship r in the schema with reifications.

4.4 A User-centered View of Relevance

The relevance methods to compute the importance of entity and event types (and, possibly, other elements in the schema) produce a general ranking of elements that does not change unless the definition of the schema itself changes. Therefore, all the users interested in exploring a large conceptual schema will obtain the same ranking regardless of their specific knowledge requirements. Figure 4.14 (left) presents an example of this situation where any user obtains the same top set of important elements.

What is needed is a more user-centered view of the relevance of elements in a large conceptual schema. To achieve this goal, the filtering methodology introduced in this thesis allows users to focus on fragments of interest from the large schema of small size, and then adapts the general concept of relevance in order to obtain feedback of high relevance with respect to the user focus. Figure 4.14 (right) shows a demonstration of this user-based perspective. Basically, a user focus on two concepts from the large schema and she obtains those elements with a higher degree of relevance and that are also close in distance with the elements of focus. Therefore, the feedback the user obtains changes whenever the user focus changes, providing a useful interaction between the user and the large schema.

In the following sections, we introduce the concepts of closeness and interest, and their formal definition, in order to achieve this goal.

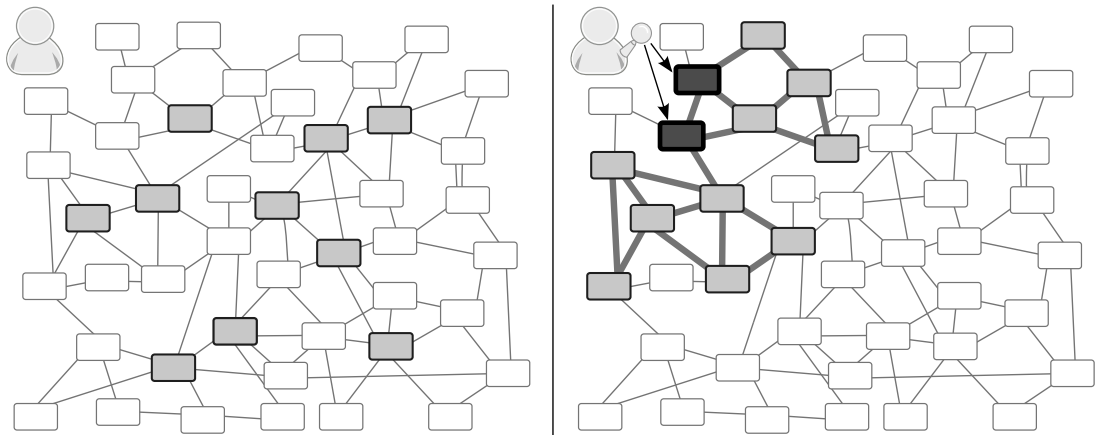


Figure 4.14. Comparison between common relevance (left) and user-centered relevance (right) in a large conceptual schema.

4.5 Closeness-Computing Method

Apart from importance-computing methods, our filtering process uses the closeness between entity types. Concretely, the closeness between each *candidate* entity type in the schema and a set of entity types of focus to the user $\mathcal{FS}$, denoted by $\Omega(e, \mathcal{FS})$. We say that e is a *candidate* entity type if $e \notin \mathcal{FS}$.

There may be several ways to compute the closeness $\Omega(e, \mathcal{FS})$ of a candidate entity type e with respect to the entity types of $\mathcal{FS}$. Intuitively, the closeness of e should be directly related to the inverse of the distance of e to the focus set $\mathcal{FS}$. For this reason, we formally define:

$$\Omega(e, \mathcal{FS}) = \frac{|\mathcal{FS}|}{\sum_{e' \in \mathcal{FS}} d(e, e')},$$

where $|\mathcal{FS}|$ is the number of entity types of $\mathcal{FS}$ and $d(e, e')$ is the minimum distance between a *candidate* entity type e and an entity type e' belonging to the focus set $\mathcal{FS}$. Intuitively, those entity types that are closer to more entity types of $\mathcal{FS}$ will have a greater closeness $\Omega(e, \mathcal{FS})$. We assume that a pair of entity types e, e' are directly connected to each other if there is a direct relationship r(e,e') $\in \mathcal{R}$ between them or if one entity type is a direct specialization of the other (e *IsA* e' or e' *IsA* e). For these cases, $d(e, e') = 1$.

Otherwise, when e, e' are not directly connected, $d(e, e')$ is defined as the length of the shortest path between them traversing relationship types and/or ascending/descending through *IsA* relationships. In these cases, $d(e, e') > 1$.

Note that $\sum_{e' \in \mathcal{FS}} d(e, e') = |\mathcal{FS}|$ when e is directly connected to all entity types of $\mathcal{FS}$. If e and e' are not connected (because at least one of them does not participate in relationship types nor *IsA* relationships, or both belong to different connected components of the graph denoted by the schema), then we define $d(e, e') = |\mathcal{E}|$.

4.6 Interest-Computing Method

The importance metric is useful when a user wants to know which are the most important entity types, but it is of little use when the user is interested in a specific subset of entity types, independently from their importance. What is needed then is a metric that measures the interest of a candidate entity type e with respect to a focus set $\mathcal{FS}$. This metric should take into account both the absolute importance of e (as explained in Section 4.3) and the closeness measure of e with regard to the entity types in $\mathcal{FS}$. For this reason, we formally define:

$$\Phi(e, \mathcal{FS}) = \alpha \times \Psi(e) + (1 - \alpha) \times \Omega(e, \mathcal{FS}),$$

where $\Phi(e, \mathcal{FS})$ is the interest of a candidate entity type e with respect to $\mathcal{FS}$, $\Psi(e)$ the importance of e, and $\Omega(e, FS)$ is the closeness of e with respect to $\mathcal{FS}$. Note that α is a balancing parameter in the range [0,1] to set the preference between closeness and importance for the retrieved knowledge. An $\alpha > 0.5$ benefits importance against closeness while an $\alpha < 0.5$ does the opposite. The default α value is set to 0.5 and can be modified by the user.

The computation of the interest $\Phi(e, \mathcal{FS})$ for candidate entity types returns a ranking which is used by our filtering method to select the $\mathcal{K} - |\mathcal{FS}|$ top candidate entity types. As an example, Table 4.24 shows the top-8 entity types with a greater value of interest when the user defines

$\mathcal{FS}$ = {*TaxRate, TaxClass*} and $\alpha = 0.5$ in the osCommerce conceptual schema [118]. Within the top of interest there may be entity types directly connected to all members of the focus set as in the case of *TaxZone* (Ω(*TaxZone*,$\mathcal{FS}$) = 1.0) but also entity types that are not directly connected to any entity type of $\mathcal{FS}$ (although they are closer/important).

Rank	Entity Type (e)	Importance $\Psi(e)$	Distance $d(e,\ TR)$	Distance $d(e,\ TC)$	Closeness $\Omega(e, \mathcal{FS})$	Interest $\Phi(e, \mathcal{FS})$
1	TaxZone	0.57	1	1	1.0	0.785
2	Product	0.84	2	1	0.66	0.75
3	Language	1.0	3	2	0.4	0.7
4	Customer	0.62	3	2	0.4	0.51
5	Zone	0.35	2	2	0.5	0.425
6	Order	0.41	3	2	0.4	0.405
7	Special	0.29	3	2	0.4	0.345
8	Currency	0.4	4	3	0.28	0.34

(*TR = TaxRate, TC = TaxClass*)

Table 4.24. Top-8 entity types of interest with regard to $\mathcal{FS}$ = {*TaxRate, TaxClass*} in osCommerce [118].

4.7 Summary

The visualization and the understanding of large conceptual schemas require the use of specific methods. These methods generate indexed, clustered, summarized or focused schemas that are easier to visualize and understand, as explained in Ch. 3. Almost all of these methods require computing the importance of each entity type in the schema. We have argued that the objective importance of an entity type in a schema should be related to the amount of knowledge that the schema defines about it. There are several proposals of metrics for entity type importance. All of them are mainly based on the amount of knowledge defined in the schema, but they only take into account the fragment of that knowledge consisting in the number of attributes, associations and specialization/generalization relationships. A complete conceptual schema also includes cardinalities, general constraints, derivation rules and the specification of events, all of which contribute to the knowledge of entity types.

We have analyzed the influence of that additional knowledge on a representative set of seven existing importance methods. We have developed extended versions of those methods by taking into account additional measures from the schema. We have evaluated original and extended versions of those methods in three large real-world schemas. The two main conclusions are: (1) among the original versions of the methods, the methods of choice are those based on the link-analysis approach; and (2) The extended versions of most methods produce remarkably similar results, which does not happen in the original version.

Finally, we have specified a closeness-computing method, and an interest-computing method that makes use of the concepts of importance and closeness. The interest method is the key measure for our filtering methodology. Chapter 5 includes a detailed explanation of the usage of this method to filter large conceptual schemas according to the needs of an specific user to obtain the appropriate feedback.

Quality is more important than quantity.
One home run is much better than two doubles.

Steve Jobs (1955-2011)

5

Filtering Method for Large Conceptual Schemas

Information filtering [55] is a rapidly evolving field to handle large information flows. The aim of information filtering is to expose users only to information that is relevant to them. We present an interactive method in which the user specifies one or more concepts of interest and the method automatically provides a (smaller) subset of the knowledge contained in the conceptual schema that is likely to be relevant. The user may then start another interaction with different concepts, until she has obtained all knowledge of interest. We present the theoretical background behind this methodology throughout this chapter.

The chapter is structured as follows. Section 5.1 introduces the proposed filtering methodology and describes its application to large conceptual schemas. In Sect. 5.2 we describe the general structure of the filtering method. It presents the required input and the resulting output of the filtering process, and introduces the concept of filtered conceptual schema, which is deeply analyzed in Sect. 5.3 showing the elements within its structural and behavioral components, and its relation with the large conceptual schema the user needs to explore. Finally, Sect. 5.4 introduces the seven filtering stages that are the backbone of the filtering methodology. Each filtering stage contains a detailed explanation of its steps and their sequential order of application, the characteristics of its inputs and output, and formal descriptions of the algorithms behind the different phases to filter a large conceptual schema and their intended interaction to satisfy a user's information need.

5.1 The Filtering Methodology

The aim of information filtering is to expose users to only information that is relevant to them. There are many filtering systems of widely varying philosophies, but all share the goal of automatically directing the most valuable information to users in accordance with their needs, and of helping them use their limited time and information processing capacity most optimally.

At present, conceptual schemas are gaining more presence in the software engineering field and beyond. Our proposal aims to contribute to the expansion of conceptual schemas by the study of its characteristics and the description of the structure and components of a filtering methodology for large conceptual schemas. With our work, we expect to facilitate the use of conceptual schemas to those interested in the knowledge represented within them.

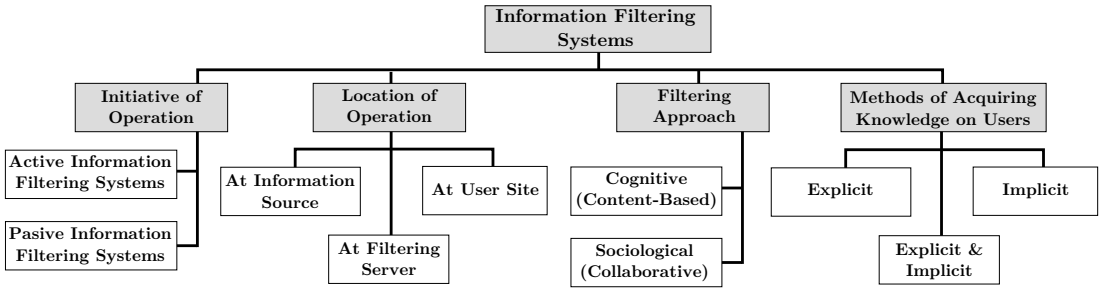


Figure 5.1. Classification of information filtering systems (adapted from [55]).

Hanani et al. introduced in [55] a comprehensive framework that define and classify information filtering systems, in accordance to a multi-layer model. They classify information filtering systems according to four parameters, that we use in order to place our filtering methodology proposal in context and introduce its main characteristics. The parameters of the classification are:

- **Initiative of operation** distinguishing between active and passive information filtering systems.
- **Location of operation** distinguishing between systems located at the information source, filtering servers, and user sites.
- **Filtering approach** distinguishing between cognitive and social filtering.
- **Method for acquiring knowledge on users** distinguishing between explicit, implicit, and combined methods.

Figure 5.1 represents the taxonomy of Hanani et al. with the aforementioned parameters to classify information filtering systems. The following subsections describe each parameter and discuss the classification of our filtering methodology according to the proposed model.

5.1.1 Initiative of Operation

This aspect of the filtering methodology concerns *who* initiates the filtering process. Hanani et al. distinguish between two types:

- **Active information filtering systems:** Systems that actively seek relevant information for their users. The system is provided with the user's profile, and it searches the space in order to collect and send relevant information to the user. The system *pushes* relevant information to the user.
- **Passive information filtering systems:** Systems that omit irrelevant information from incoming streams of data items. The role of the filtering system is to determine the relevance of data items to the user, according to the user's profile. Some filtering systems filter out irrelevant data items, while other provide the user with all available data items, rank-ordered by their relevance.

The filtering method we propose requires the user intervention to start the filtering process. The user focus on a set of elements from the large schema. The user is aware of those elements or she has accessed them via previous interactions with the large schema. Then our filtering system seeks for additional elements in the large schema of high relevance in order to surround the elements of focus with more knowledge about the schema. Therefore, our filtering system is **passive** because do not *pushes* results without the direct intervention of the user.

5.1.2 Location of Operation

Hanani et al. indicate that the filtering process can take place in three possible locations:

- **At the information source:** In this approach, a user posts his/her profile to an information provider. In return, the user is supplied with information that matches the profile.
- **At a filtering server:** Some filtering systems are implemented at special intermediate servers. On the one hand, users post their profiles to the servers, and on the other hand, information providers send data items to these servers, which filter and distribute relevant items to respective users.
- **At the user site:** This is the most popular location of filtering operation. Each incoming stream of data items is evaluated by a local filtering system, which removes the irrelevant items, or rank-orders them by their relevance. Filtering at the user site implements passive filtering, as the data items flow in automatically, and only then are they evaluated.

Our proposed filtering methodology is located **at the information source**. As Ch. 8 explains, we develop the filtering engine that supports our filtering method as a web service. Therefore, the server side of the filtering engine keeps the knowledge from the large schema an accepts the specific filtering requests of clients that indicate a focus set and want a small schema as the filtering result.

5.1.3 Filtering Approach

This aspect of the filtering methodology distinguishes two main filtering approaches:

- **Cognitive filtering:** Systems based on the correlation between the content of the data items and the user model that represents the user's cognitive style and personality, user goals and plans.
- **Sociological filtering:** Systems that automate the process of human recommendations. A data item is recommended to a user on the basis of its being relevant to other users having similar habits.

The proposed filtering method takes a focus set of selected schema elements that represents the interest point of a user and indexes the large conceptual schema in order to extract a small fragment with those additional elements that have a higher relation with the elements of focus. This interest-based approach uses the relevance metrics of Ch. 4 to construct the different stages of a **cognitive** filtering process.

5.1.4 Method of Acquiring Knowledge on Users

Different information filtering systems use different methods to acquire knowledge about their users. Hanani et al. indicate that the methods for acquiring knowledge about users include an *explicit approach*, which is based on user interrogation, an *implicit approach*, which infers the user model automatically by recording user behavior, and a mixed approach.

- **Explicit:** Systems utilizing this method usually require their users to fill out a form describing their areas of interest or other relevant parameters; provide the user with a set of terms that represent each domain, from which she can construct a personal profile; or allow users to determine terms and their weights of importance, or to choose a search strategy.
- **Implicit:** Systems that do not require active user involvement in the knowledge acquisition task. Instead, the user's reaction to each incoming data item is recorded, in order to learn from it about the actual relevancy of the data item to the user.
- **Explicit & implicit:** Systems that lie between the explicit and the implicit approaches, as they require minimum user involvement. The users are asked to provide explicit information about themselves to enable the start of the system. Then, the filtering system infers user profiles. Any new incoming data item is tested for its similarity to the profiles. If similarity of the new data item is above a certain relevance threshold, it is considered relevant.

Our filtering methodology follows an explicit process of acquiring knowledge on users based on user interrogation. The method expects the intervention of the user to obtain the filtering preferences that conform the input of the filtering process, which contains as core element the focus set of schema elements of interest.

5.2 General Structure of the Filtering Method

In this section we describe how a large conceptual schema can be filtered using the methodology that we propose, which corresponds to the main contribution of this thesis. The main idea is to extract a reduced and self-contained view from the large schema, that is, a filtered conceptual schema with the knowledge of interest to the user. Figure 5.2 presents the three phases of our filtering process.

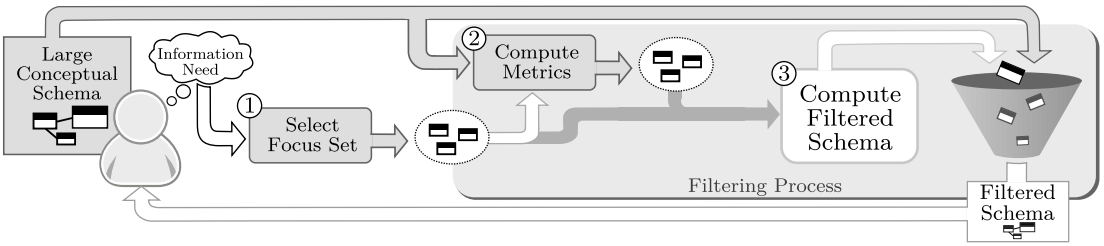


Figure 5.2. General structure of the filtering method.

The first phase consists in preparing the required information to filter the large schema according to the specific needs of the user. Basically, the user focus on a set of schema elements she is interested in and our method surrounds them with additional related knowledge from the large schema. Therefore, it is mandatory for the user to select a non-empty initial focus set of schema elements of interest.

During the second phase our method computes the required metrics to automatically select the most interesting schema elements to extend the knowledge selected in the focus set of the first phase. The main goal of these metrics is to discover those schema elements that are relevant in the schema but also that are close (in terms of structural distance over schema) to the elements of the focus set. We presented a detailed definition of such metrics in Ch. 4.

Finally, the last phase receives the set of most interesting elements selected in the previous phase and puts it together with the schema elements of the focus set in order to create a filtered conceptual schema with the elements of both sets. The main goal of this step consists in filtering information from the original schema involving the elements in the filtered schema. To achieve this goal, the method explores the relationships and generalizations/specializations in the original schema that are defined between those elements and includes them in the filtered schema to obtain a connected schema.

Apart from the schema itself, the information filtering method requires as input a representation of the user's information need through a request of knowledge. Thus, the method adapts itself to the specific request and produces different results every interaction. After processing the request and analyzing the whole schema, the filtering method selects which elements of the original schema have to be included into a resulting reduced schema, which is shown to the user. The process is repeated until the user needs are satisfied.

The following sections of the chapter present the characteristics of the required input and resulting output of the filtering method, as well as the details of the different stages of the filtering process within it.

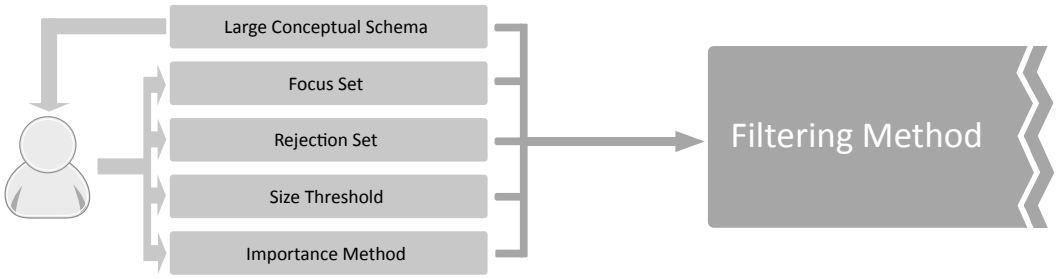


Figure 5.3. Input of the filtering method.

5.2.1 Common Input of the Filtering Method

As aforementioned, the filtering method needs a specific input to filter a large conceptual schema. The output of our filtering method strongly depends on the particular characteristics of its input. The main components of the input for the filtering method (Fig. 5.3) are:

Large Conceptual Schema It is the backbone of the input and the reference to compute the relevance metrics presented in Ch. 4. The large schema consists of a structural and a behavioral subschemas, as introduced in Ch. 2. The different schema elements included within them conform the knowledge represented in the schema about a specific domain of interest.

Focus Set It includes the schema elements the user wants to focus on, and works as the conceptual schema viewpoint of the user. Therefore, a focus set is an initial point that should be extended with more knowledge from the schema. The filtering method allows different kinds of elements to be included in the focus set. It means that the behavior of the filtering method and, therefore, the obtained output from its application to the conceptual schema, will both depend on the specific elements in the input. In Ch. 6 we present a catalog of filtering requests that the user may apply according to the different kind of elements within the focus set and her filtering requirements.

Rejection Set It specifies the schema elements the user denotes as not interesting for her knowledge request. Our filtering method ignores those elements and will not provide knowledge about them to the user. The filtering method allows different kinds of elements to be included in the rejection set. An element in the rejection set cannot appear in the focus set nor in the filtered conceptual schema.

Size Threshold It denotes how much knowledge in form of schema elements the user wants to obtain from the original large schema into the filtered conceptual schema. The value of the size threshold limits the size of the output according to the user requirements. An inexpert user may select a small filtered conceptual schema while a more experienced one may want a larger and more detailed output.

Importance Method It denotes which method to compute the importance of entity, event, and relationship types among those defined in Ch. 4 has to be used in the filtering method.

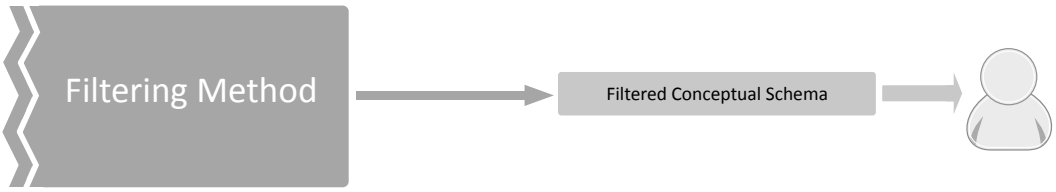


Figure 5.4. Output of the filtering method.

5.2.2 Common Output of the Filtering Method

The filtering method produces a filtered conceptual schema as output of its filtering process (Fig. 5.4). The characteristics of such filtered schema depend on the particular user information needs represented in the input of the filtering method. The common characteristics of the resulting filtered conceptual schema are:

Knowledge Subsetting The knowledge contained in the filtered conceptual schema is a subset of the knowledge of the original large conceptual schema. It means that the filtering process does not create new knowledge apart from the knowledge of the original schema. Therefore, the schema elements that appear in the resulting filtered conceptual schema come from the original large schema through a process of knowledge extraction based on the user interest. The resulting schema is a focused view over the general schema.

Valid Instantiation The filtered conceptual schema is a valid instance of the UML metamodel. It means that the elements included within the filtered conceptual schema are concrete instances of the metaclasses of the UML metamodel, and also that the filtered conceptual schema is syntactically and semantically correct according to the UML metamodel. Only common and standard constructors from the UML are used to create the resulting schema of the output.

Interest-driven Approach There is a relationship of consequence between the user's interest and the knowledge in the filtered conceptual schema. It means that the filtered conceptual schema changes in the same way that the user's information need does. Since the user selects the specific filtering input to use in the filtering process, and also focus on a particular set of elements of interest, the resulting filtered conceptual schema will contain the knowledge with a highest relation with the user's selection. Therefore, if such selection changes, the filtering process changes and, as a result, the output of the filtering method will also change.

Size Reduction The size of the filtered conceptual schema is smaller than the size of the original large conceptual schema. It is clear that in order to reduce the cost in both time and effort to understand the knowledge represented in the resulting schema, it must be of reasonable size. It is specially mandatory when who has to work with the schema has not enough knowledge about the information domain that such schema represents.

A detailed description of the different components that are part of a filtered conceptual schema is presented in the next section of the chapter.

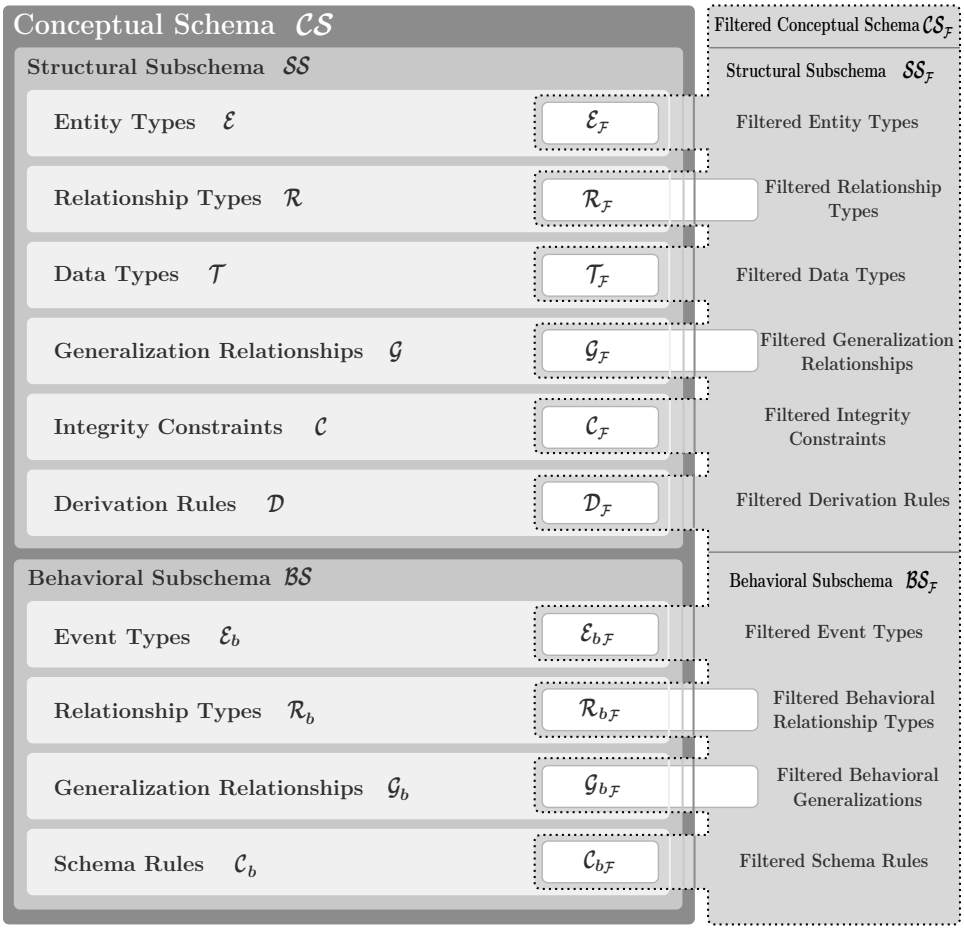


Figure 5.5. Structure of a filtered conceptual schema.

5.3 Filtered Conceptual Schema

A filtered conceptual schema is a conceptual schema with the same characteristics as presented in Ch. 2, but with a portion of the knowledge contained in the original large conceptual schema. The main task of our filtering method consists in constructing a filtered conceptual schema $\mathcal{CS}_\mathcal{F}$ including the elements of a focus set selected by the user and the more relevant elements computed in the filtering process that represent additional knowledge of interest from the original schema $\mathcal{CS}$.

Formally, we define a filtered conceptual schema as a tuple $\mathcal{CS}_\mathcal{F} = \langle \mathcal{SS}_\mathcal{F}, \mathcal{BS}_\mathcal{F} \rangle$, where $\mathcal{SS}_\mathcal{F} = \langle \mathcal{E}_\mathcal{F}, \mathcal{R}_\mathcal{F}, \mathcal{T}_\mathcal{F}, \mathcal{G}_\mathcal{F}, \mathcal{C}_\mathcal{F}, \mathcal{D}_\mathcal{F} \rangle$ is the structural subschema, and $\mathcal{BS}_\mathcal{F} = \langle \mathcal{E}_{b\mathcal{F}}, \mathcal{R}_{b\mathcal{F}}, \mathcal{G}_{b\mathcal{F}}, \mathcal{C}_{b\mathcal{F}} \rangle$ is the behavioral subschema. Figure 5.5 depicts the structure of a filtered conceptual schema with all the components of the structural and behavioral subschemas, and their relation with the large conceptual schema from which they are filtered. It is important to note that the construction of a filtered conceptual schema does not produce a unique result. A user may obtain different filtered schemas by using different inputs on several execution of the method.

5.3.1 Structural Subschema

The structural subschema $\mathcal{SS}_{\mathcal{F}}$ of a filtered conceptual schema $\mathcal{CS}_{\mathcal{F}}$ that has been filtered from a large conceptual schema $\mathcal{CS}$ contains the following components:

- $\mathcal{E}_{\mathcal{F}}$ is a set of entity types filtered from $\mathcal{E}$ of the original schema $\mathcal{CS}$. Formally, $\mathcal{E}_{\mathcal{F}} \subset \mathcal{E}$. Sections 5.4.1 and 5.4.2 present the general details about the construction of $\mathcal{E}_{\mathcal{F}}$.
- $\mathcal{R}_{\mathcal{F}}$ is a set of relationship types filtered from $\mathcal{R}$ of the original schema $\mathcal{CS}$. Formally, $\mathcal{R}_{\mathcal{F}} \sqsubset \mathcal{R}$, which means that the relationship types of $\mathcal{R}_{\mathcal{F}}$ belong to $\mathcal{R}$ or are projections of relationship types of $\mathcal{R}$. Section 5.4.3 presents the general details about the construction of $\mathcal{R}_{\mathcal{F}}$.
- $\mathcal{T}_{\mathcal{F}}$ is a set of data types filtered from $\mathcal{T}$ of the original schema $\mathcal{CS}$. Formally, $\mathcal{T}_{\mathcal{F}} \subset \mathcal{T}$. Section 5.4.6 presents the general details about the construction of $\mathcal{T}_{\mathcal{F}}$.
- $\mathcal{G}_{\mathcal{F}}$ is a set of generalization relationships filtered from $\mathcal{G}$ of the original schema $\mathcal{CS}$. Formally, $\mathcal{G}_{\mathcal{F}} \sqsubset \mathcal{G}$, which means that the generalization relationships of $\mathcal{G}_{\mathcal{F}}$ belong to $\mathcal{G}$ or are artificial direct generalizations that represent a path of generalizations of $\mathcal{G}$. Section 5.4.4 presents the general details about the construction of $\mathcal{G}_{\mathcal{F}}$.
- $\mathcal{C}_{\mathcal{F}}$ is a set of integrity constraints filtered from $\mathcal{C}$ of the original schema $\mathcal{CS}$. Formally, $\mathcal{C}_{\mathcal{F}} \subset \mathcal{C}$. Section 5.4.5 presents the general details about the construction of $\mathcal{C}_{\mathcal{F}}$.
- $\mathcal{D}_{\mathcal{F}}$ is a set of derivation rules filtered from $\mathcal{D}$ of the original schema $\mathcal{CS}$. Formally, $\mathcal{D}_{\mathcal{F}} \subset \mathcal{D}$. Section 5.4.5 presents the general details about the construction of $\mathcal{D}_{\mathcal{F}}$.

5.3.2 Behavioral Subschema

The behavioral subschema $\mathcal{BS}_{\mathcal{F}}$ of a filtered conceptual schema $\mathcal{CS}_{\mathcal{F}}$ that has been filtered from a large conceptual schema $\mathcal{CS}$ contains the following components:

- $\mathcal{E}_{b\mathcal{F}}$ is a set of event types filtered from $\mathcal{E}_b$ of the original schema $\mathcal{CS}$. Formally, $\mathcal{E}_{b\mathcal{F}} \subset \mathcal{E}_b$. Sections 5.4.1 and 5.4.2 present the general details about the construction of $\mathcal{E}_{b\mathcal{F}}$.
- $\mathcal{R}_{b\mathcal{F}}$ is a set of relationship types filtered from $\mathcal{R}_b$ of the original schema $\mathcal{CS}$. Formally, $\mathcal{R}_{b\mathcal{F}} \sqsubset \mathcal{R}_b$, which means that the relationship types of $\mathcal{R}_{b\mathcal{F}}$ belong to $\mathcal{R}_b$ or are projections of relationship types of $\mathcal{R}_b$. Section 5.4.3 presents the general details about the construction of $\mathcal{R}_{b\mathcal{F}}$.
- $\mathcal{G}_{b\mathcal{F}}$ is a set of generalization relationships filtered from $\mathcal{G}_b$ of the original schema $\mathcal{CS}$. Formally, $\mathcal{G}_{b\mathcal{F}} \sqsubset \mathcal{G}_b$, which means that the generalization relationships of $\mathcal{G}_{b\mathcal{F}}$ belong to $\mathcal{G}_b$ or are artificial direct generalizations that represent a path of generalizations of $\mathcal{G}_b$. Section 5.4.4 presents the general details about the construction of $\mathcal{G}_{b\mathcal{F}}$.
- $\mathcal{C}_{b\mathcal{F}}$ is a set of invariants, pre- and postconditions filtered from $\mathcal{C}_b$ of the original schema $\mathcal{CS}$. Formally, $\mathcal{C}_{b\mathcal{F}} \subset \mathcal{C}_b$. Section 5.4.5 presents the general details about the construction of $\mathcal{C}_{b\mathcal{F}}$.

5.4 The 7 Stages of the Filtering Method

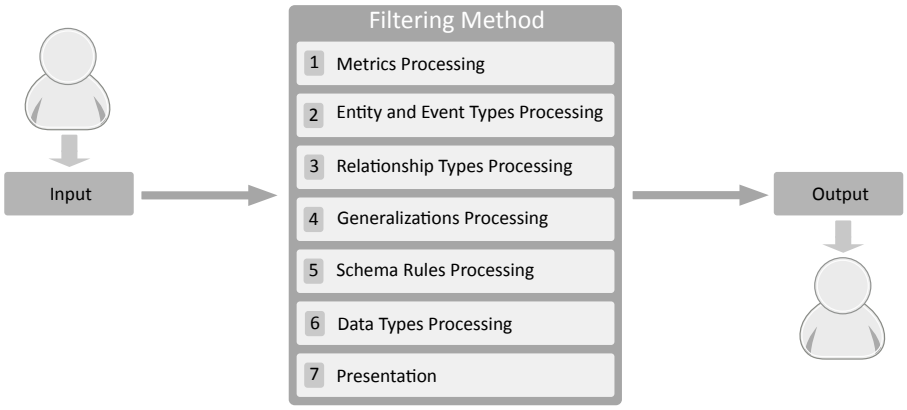


Figure 5.6. The 7 stages of the filtering method.

The filtering method is divided into seven ordered stages that sequentially process the input specified by a user in order to obtain a particular output. Figure 5.6 presents the different stages of the filtering method.

1. **Metrics Processing** The first stage applies the metrics of Ch. 4 to the elements of the original large schema out of the focus set in order to discover which are the most relevant ones for the user.
2. **Entity and Event Types Processing** The second stage selects the entity and event types from the large schema that will appear in the resulting filtered schema. Such set will contain at least the entity and event types included in the user's focus set.
3. **Relationship Types Processing** This stage selects the relationship types from the large schema that will appear in the resulting filtered schema. This process makes use of projection and redefinition to align the relationships with the user's interest.
4. **Generalizations Processing** This stage selects the generalization relationships whose members belong to the filtered schema, processes them to avoid redundancies, and includes them in the output.
5. **Schema Rules Processing** This stage processes the schema rules of the original schema in order to include into the output those that affect elements of interest to the user.
6. **Data Types Processing** This stage selects those data types referenced by elements of the filtered schema in order to add them into it.
7. **Presentation** The last stage deals with the representation of the filtered schema to the user in order to maximize its understandability.

Next subsections present a detailed description of the main characteristics of each stage of the filtering method.

A running example: Magento

We illustrate the different stages of the filtering methodology by using a simple example to be extended in later sections. We use the conceptual schema of the Magento e-commerce system [94]. According to its website[1], Magento is a feature-rich e-commerce platform built on open-source technology that provides online merchants with unprecedented flexibility and control over the look, content, and functionality of their e-commerce store. Magento e-commerce system provides two different components: an e-commerce site or online store, to which customers interact to make their purchases; and the administration of the store, designed for the internal management of the site.

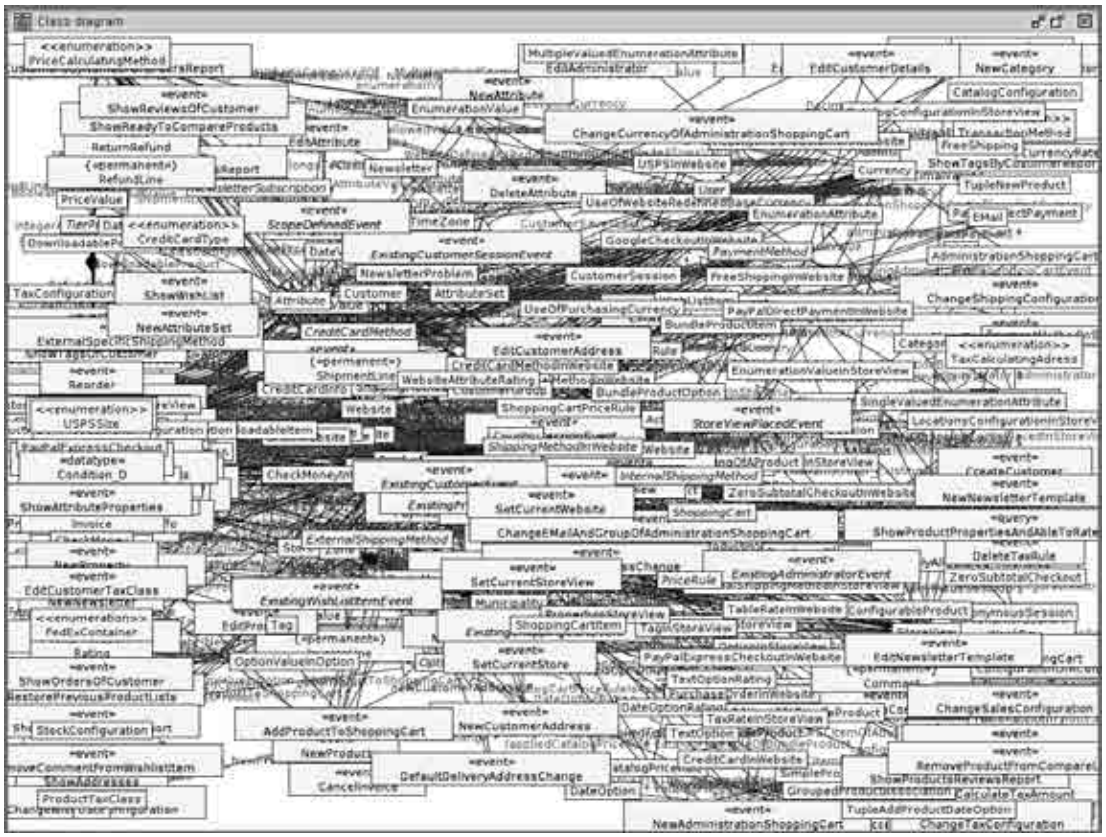


Figure 5.7. Conceptual schema of the Magento e-commerce system.

Magento's conceptual schema (see Fig. 5.7) has been specified in UML/OCL. The structural schema consists of a taxonomy of 218 entity types (with their generalization/specialization relationships and the taxonomic constraints), a set of relationship types (either attributes or associations), the cardinality constraints of the relationship types, and a set of other constraints formally defined in OCL. The behavioral schema consists of a set of 187 event types modelized as UML classes with the «event» stereotype. Magento's schema has a total of 983 attributes, 165 generalizations, 319 associations, 893 general constraints, and 69 pre- and postconditions. Therefore, it can be considered a large schema.

[1]Magento official site `http://www.magentocommerce.com`

5.4.1 Stage 1: Metrics Processing

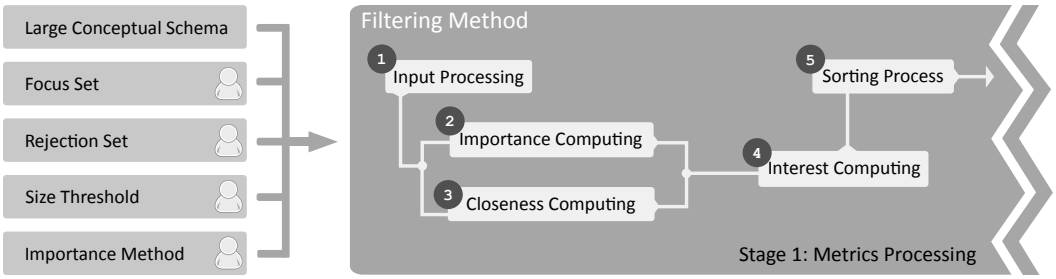


Figure 5.8. Stage 1: Metrics Processing.

As aforementioned, the proposed filtering method starts by processing the necessary metrics from Ch. 4 that enable the extraction of those elements that are of interest to the user who constructs the input. The metrics-processing stage contains five mandatory steps to perform its function. Figure 5.8 shows the steps and their sequential order of application.

Input processing

First step consists of processing the different components of the input. As seen in Sec. 5.2.1 the input includes the focus and rejection sets, the size threshold, the importance method, and the large conceptual schema:

$$
\begin{aligned}
&\mathcal{CS} \ : \texttt{Large Conceptual Schema}\\
&\mathcal{FS} \ : \texttt{Focus Set}\\
&\mathcal{RS} \ : \texttt{Rejection Set}\\
&\mathcal{K} \ : \texttt{Size Threshold}\\
&\mathcal{I} \ : \texttt{Importance Method}
\end{aligned}
$$

Next, some sets of elements are extracted from the focus set $\mathcal{FS}$ in order to prepare the process to compute the relevance metrics. These sets gather the entity, event and relationship types that are present in $\mathcal{FS}$:

$$
\begin{aligned}
\mathcal{E}_{\mathcal{FS}} &= \{e \in \mathcal{E} \mid e \in \mathcal{FS} \vee e \rightleftharpoons \mathcal{FS}\}^2\\
\mathcal{E}_{b\mathcal{FS}} &= \{ev \in \mathcal{E}_b \mid ev \in \mathcal{FS} \vee ev \rightleftharpoons \mathcal{FS}\}\\
\mathcal{R}_{\mathcal{FS}} &= \{r \in \mathcal{R} \mid r \in \mathcal{FS} \vee r \rightleftharpoons \mathcal{FS}\}\\
\mathcal{S}_{\mathcal{FS}} &= \mathcal{E}_{\mathcal{FS}} \cup \mathcal{E}_{b\mathcal{FS}} \cup \mathcal{R}_{\mathcal{FS}}
\end{aligned}
$$

$\mathcal{E}_{\mathcal{FS}}$ contains the entity types of focus, $\mathcal{E}_{b\mathcal{FS}}$ the corresponding event types, and $\mathcal{R}_{\mathcal{FS}}$ the relationship types. Finally, $\mathcal{S}_{\mathcal{FS}}$ is the union of all the previous elements of user focus. Note that a single element to be included in $\mathcal{E}_{\mathcal{FS}}$, $\mathcal{E}_{b\mathcal{FS}}$, or $\mathcal{R}_{\mathcal{FS}}$ must belong to the focus set $\mathcal{FS}$, or it must be referenced in at least one of the schema rules included in $\mathcal{FS}$.

[2]If s is an entity, event or relationship type and $\mathcal{A}$ is a set containing schema rules, we say that $s \rightleftharpoons \mathcal{A}$ if and only if s is referenced by any of the schema rules of $\mathcal{A}$

Importance computing

Our approach is based on the concept of importance. Second step consists in computing the importance of the entity and event types from the input schema with the importance method $\mathcal{I}$ selected in the input. Our approach can be used in connection with any of the existing importance-computing methods from Ch. 4.

Algorithm 5.1. Compute Importance Ψ.

```
for each e ∈ {E ∪ E_b} do
    Ψ(e) = I(e)
end
```

As indicated in Alg. 5.1, the filtering method computes the global importance Ψ of any entity or event type of the large schema $\mathcal{CS}$. The most relevant entity and event types will be the basis of the resulting filtered conceptual schema, in conjunction to the elements in the focus set $\mathcal{FS}$.

Closeness computing

The importance metric is useful when a user wants to know which are the most important entity and event types, but it is of little use when the user is interested in a specific subset of elements of focus, independently from their importance. This step computes the measure of closeness of the entity and event types of the original schema $\mathcal{CS}$ that are candidates to be included in the resulting filtered schema, with respect to the elements of the focus set $\mathcal{FS}$. A candidate entity or event type belongs to the original schema and is neither in the focus set nor in the rejection set.

Algorithm 5.2. Compute Closeness Ω.

```
for each e ∈ {{E ∪ E_b} \ {S_FS ∪ R_S}} do
    Ω(e, S_FS) = |S_FS| / Σ_{s∈S_FS} δ(e, s)
end
```

Intuitively, the closeness of e should be directly related to the inverse of the distance δ of e to the elements of the focus set, which are in $\mathcal{S}_{\mathcal{FS}}$. Those entity and event types that are closer to more elements of $\mathcal{S}_{\mathcal{FS}}$ will have a greater closeness $\Omega(e, \mathcal{S}_{\mathcal{FS}})$.

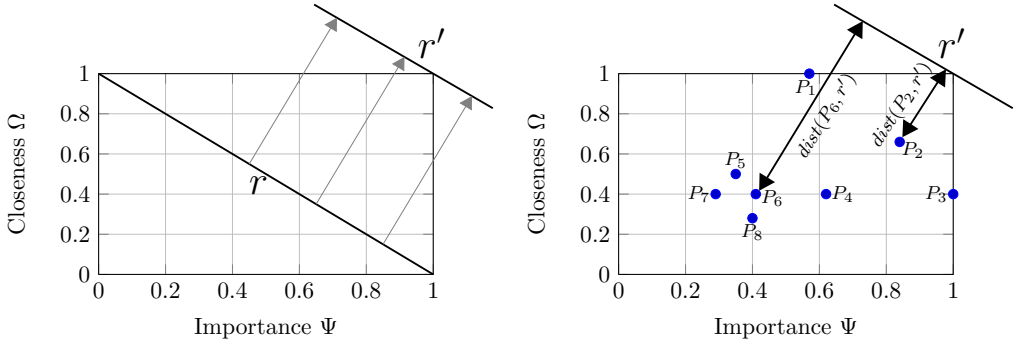
Interest computing

Our method requires a metric that measures the interest of a candidate entity or event type e with respect to the focus set $\mathcal{FS}$. This metric should take into account both the absolute importance of e and the closeness measure of e with regard to the elements in $\mathcal{FS}$ (as explained in Ch. 4). For this reason, the filtering method computes such measure as shown in Alg. 5.3.

Algorithm 5.3. Compute Interest Φ.

```
for each e ∈ {{E ∪ E_b} \ {S_FS ∪ R_S}} do
    Φ(e) = α × Ψ(e) + (1 − α) × Ω(e, S_FS)
end
```

Note that α is a balancing parameter in the range [0,1] to set the preference between closeness and importance for the retrieved knowledge. An $\alpha > 0.5$ benefits importance against closeness while an $\alpha < 0.5$ does the opposite. The default α value is set to 0.5 and can be modified by the user.



(a) Movement of r to obtain r'.

(b) Distances to the interest indicator r'.

Figure 5.9. Geometrical foundation of the concept of Interest of entity and event types $\Phi(e)$.

Each candidate entity type or event type e of the conceptual schema $\mathcal{CS}$ can be seen, in a geometrically sense, as a point in a bidimensional space with the axis being the measures of *importance* $\Psi(e)$ and *closeness* $\Omega(e, \mathcal{FS})$. Figure 5.9(a) shows such bidimensional space with the corresponding axis. Let r be a straight line between the points $(0, \Omega_{max})$ and $(\Psi_{max}, 0)$ of the maximum values of closeness and importance ($\Omega_{max} = \Psi_{max} = 1$ in Fig. 5.9(a)). We choose r in order to maintain the same proportion between closeness and importance ($\alpha = 0.5$). A straight line r' parallel to r traversing the point $(\Psi_{max}, \Omega_{max})$ indicates the interest line to the user (see Fig. 5.9(b)).

Taking the importance and the closeness measures to a set of entity and event types, we obtain the coordinates to place them as bidimensional points in the plane, as shown in Fig. 5.9(b). The distance between each point in the plane and the straight line r' is inversely proportional to the interest of the entity or event type the point represents. Figure 5.9(b) shows that the element placed at point P2=(0.84, 0.66) is of more interest than the element at point P6=(0.41, 0.4) due to its smaller distance to r'. Note that the balancing parameter α in can be seen as a modifier of the slope of the straight line r' of Fig. 5.9(b), in order to prioritize the closeness or importance components. In particular, if we choose $\alpha = 1$ then we only take into account the importance $\Psi(e)$, and the element at point P3 would be ranked the first.

Sorting process

The last step sorts the entity and event types e that are candidates to be included into the final filtered conceptual schema according to their interest $\Phi(e)$ obtained from the previous steps. Next stage will start straight afterwards this sorting process.

Algorithm 5.4. Compute ordered list $\mathcal{L}$.

```
1 L = sort e ∈ {{E ∪ E_b} \ {S_FS ∪ RS}} with Φ(e)
```

Example: Magento - Stage 1

In order to show an example of the first stage of the filtering methodology, we propose the following scenario where a user of the conceptual schema of the Magento e-commerce system wants to explore the knowledge about a particular portion of the schema. Concretely, the user is interested in the functionalities related to the log in and log out of customers within an online store developed with the Magento system. The user may construct the input of our proposed method as follows:

```
CS = Magento
FS = {LogIn, LogOut, Customer}
RS = ∅
K  = 6
I  = CEntityRank Extended
```

The focus set of interest contains the event types LogIn and LogOut, and the entity type Customer, all of them defined within the schema of Magento. Also, the user indicates that the resulting filtered conceptual schema may contain at most 6 entity/event types, and that the algorithm to compute the relevance of the elements in the schema must be the extended version of the CEntityRank introduced in Ch. 4. We can assume that the user has no previous experience with the conceptual schema of Magento and, therefore, she maintains the rejection set empty.

With this information, our method process the input as aforementioned and computes the importance, closeness, and interest metrics. The candidate entity types are sorted according to their final value of the interest metric, as shown in Tab. 5.1. Note that the ranking contains elements of high importance (as the entity type Product, which is the most important element in the schema), and also elements that are close to all the elements within the selected focus set (as in the case of the event type ExistingCustomerEvent, which is directly connected to LogIn, LogOut, and Customer).

Table 5.1. Top-10 entity and event types of interest with regard to $\mathcal{FS} = \{LogIn,\ LogOut,\ Customer\}$.

Rank	Element Type (e)		Importance $\Psi(e)$	Closeness $\Omega(e, \mathcal{FS})$	Interest $\Phi(e, \mathcal{FS})$
1	*Entity*	StoreView	0.972	0.375	0.673
2	*Entity*	Website	0.915	0.429	0.672
3	*Entity*	**Product**	**1**	0.333	0.667
4	*Event*	**ExistingCustomerEvent**	0.172	**1**	0.586
5	*Entity*	ProductInStoreView	0.691	0.273	0.482
6	*Entity*	Order	0.571	0.333	0.452
7	*Entity*	Session	0.37	0.429	0.399
8	*Entity*	Store	0.414	0.3	0.357
9	*Entity*	Address	0.267	0.429	0.348
10	*Entity*	CustomerSession	0.193	0.5	0.346

$\mathcal{FS} = (LogIn,\ LogOut,\ Customer)$

5.4.2 Stage 2: Entity and Event Types Processing

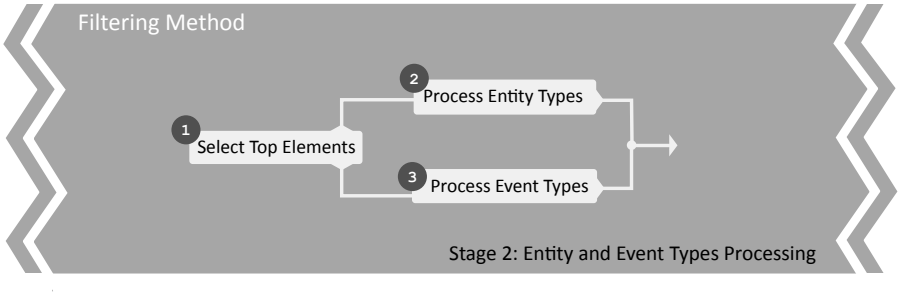


Figure 5.10. Stage 2: Entity and Event Types Processing.

The second stage of the filtering method deals with the process of selecting the entity and event types that are included in the resulting filtered conceptual schema. This stage contains three steps to perform that function. Figure 5.10 shows the steps and their sequential order of application.

Select Top Elements

The last step of the previous stage constructed a list $\mathcal{L}$ with the candidate entity and event types e to be included in the filtered schema in order of decreasing interest value $\Phi(e)$. This step selects the top entity and event types from $\mathcal{L}$ until reaching the size threshold $\mathcal{K}$ of the input. Such selected schema elements are included in $\mathcal{E}_\Phi$ as shown in Alg 5.5.

Algorithm 5.5. Compute interest set $\mathcal{E}_\Phi$.

```
1 E_Φ = ∅
2 k = |{E_FS ∪ E_bFS}|
3 while k < K do
4   e = top(L)[3]
5   E_Φ = E_Φ ∪ {e}
6   k = k + 1
7 end
```

Since the resulting filtered conceptual schema must always contain the entity and event types —$\mathcal{E}_{\mathcal{FS}}$ and $\mathcal{E}_{b\mathcal{FS}}$— from the focus set $\mathcal{FS}$, the amount of elements of interest $|\mathcal{E}_\Phi|$ that are selected equals to $\mathcal{K} - |\{\mathcal{E}_{\mathcal{FS}} \cup \mathcal{E}_{b\mathcal{FS}}\}|$. Figure 5.11 presents a Venn diagram of the entity and event types that are included in the filtered conceptual schema. The set $\mathcal{E}_{\mathcal{F}}\mathcal{S}$ of filtered entity types contains the entity types $\mathcal{E}_{\mathcal{FS}}$ from the focus set, the entity types from the interest set $\mathcal{E}_\Phi$, and a set $\mathcal{E}_{aux}$ of auxiliary entity types that help with the projection of relationship types (see Sect. 5.4.3). The case of the filtered event types is analogous to the entity types. Note that the size threshold $\mathcal{K}$ from the user input does not take into account the auxiliary sets $\mathcal{E}_{aux}$ and $\mathcal{E}_{baux}$.

[3] If $\mathcal{A}$ is an ordered list s.t. $\mathcal{A} = [a, b, c]$, we assume that $\texttt{top}(\mathcal{A}) = a$, and after that $\mathcal{A} = [b, c]$.

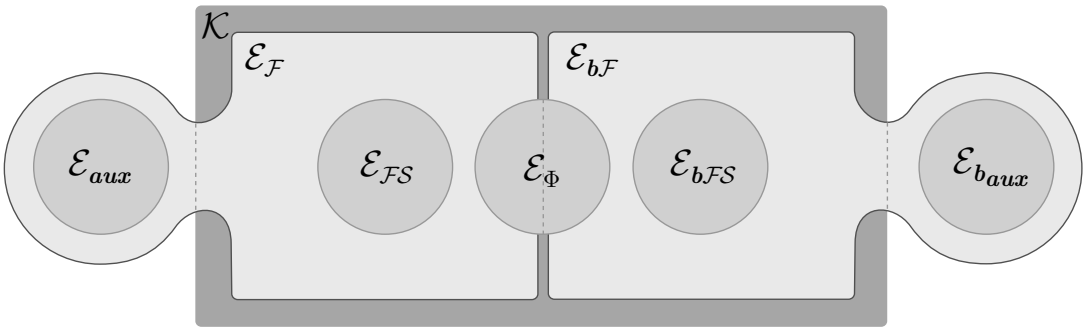


Figure 5.11. Venn diagram of the entity and event types in the filtered conceptual schema.

Process Entity Types

Next, the following step constructs the set of entity types $\mathcal{E}_{\mathcal{F}}$ of the filtered conceptual schema $\mathcal{CS}_{\mathcal{F}}$. This set includes both the entity types $\mathcal{E}_{\mathcal{FS}}$ from the focus set $\mathcal{FS}$ and the entity types of interest from the set $\mathcal{E}_{\Phi}$.

Algorithm 5.6. Compute entity types of $\mathcal{CS}_{\mathcal{F}}$.

```
1 E_F = ∅
2 for each e ∈ {E_FS ∪ {E ∩ E_Φ}} do
3   E_F = E_F ∪ {e}
4 end
```

Process Event Types

Correspondingly, this step constructs the set of event types $\mathcal{E}_{b\mathcal{FS}}$ of the filtered conceptual schema $\mathcal{CS}_{\mathcal{F}}$. This set includes both the event types $\mathcal{E}_{b\mathcal{FS}}$ from the focus set $\mathcal{FS}$ and the event types of interest from the set $\mathcal{E}_{\Phi}$.

Algorithm 5.7. Compute event types of $\mathcal{CS}_{\mathcal{F}}$.

```
1 E_bF = ∅
2 for each ev ∈ {E_bFS ∪ {E_b ∩ E_Φ}} do
3   E_bF = E_bF ∪ {ev}
4 end
```

Example: Magento - Stage 2

To continue with our example, we need to select the top elements from the previous ranking shown in Tab. 5.1. Concretely, since the user selected the event types LogIn and LogOut, and the entity type Customer, and she indicated that the size of the output schema may be of 6 elements, the method completes it with three more elements. The selected elements are the ones in the top-3 of Tab. 5.1, i.e., the entity types StoreView, Website, and Product. Therefore, the filtered conceptual schema will contain the following elements:

```
E_F  = {Customer, StoreView, Website, Product} -- Filtered Entity Types
E_bF = {LogIn, LogOut} -- Filtered Event Types
```

5.4.3 Stage 3: Relationship Types Processing

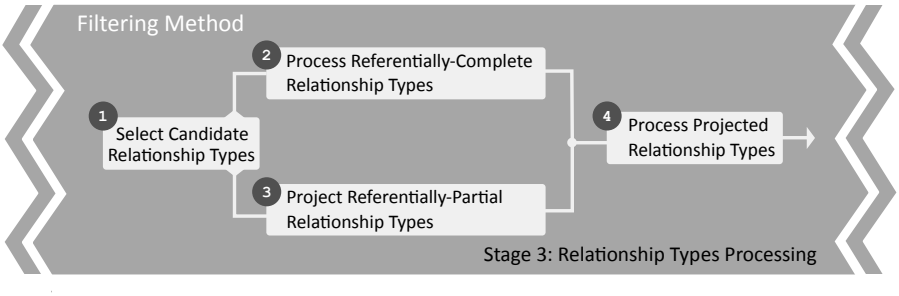


Figure 5.12. Stage 3: Relationship Types Processing.

The third stage of the filtering method deals with the process of selecting the relationship types that are included in the resulting filtered conceptual schema. This stage contains three steps to perform that function. Figure 5.12 shows the steps and their sequential order of application.

Select Candidate Relationship Types

The first step in this stage selects those relationship types from the original schema that can be included in the filtered schema and classifies them according to their participants. As a result of this process, the relationship types are divided into referentially-complete ($\mathcal{R}_{rc}$) and referentially-partial ($\mathcal{R}_{rp}$) relationships as shown in Alg. 5.8.

Algorithm 5.8. Classification of candidate relationship types.

```
1  R_rc = ∅ -- set of referentially-complete relationships
2  R_rp = ∅ -- set of referentially-partial relationships
3  R_d  = ∅ -- set of selected redefinition relationships
4  partial = false
5  incomplete = false
6  for each r ∈ {{R ∪ R_b} \ RS} do
7    for each e ∈participants(r) do
8      if e ∉ CS_F then
9        if ∃e' s.t. e' ∈descendants(e) and e' ∈ CS_F then
10         partial = true
11       else
12         incomplete = true
13         break
14       endif
15     endif
16   end
17   if not incomplete then
18     if partial then R_rp = R_rp ∪ {r} else R_rc = R_rc ∪ {r} endif
19     if isRedefinition(r) then R_d = R_d ∪ {r} endif
20   endif
21 end
```

A referentially-complete relationship is a relationship type whose participant entity or event types are all included in the filtered conceptual schema $\mathcal{CS}_\mathcal{F}$. On the other hand, a referentially-partial relationship is a relationship type with some participants inside of $\mathcal{CS}_\mathcal{F}$ and others outside, but all of those who are outside have descendants that are included in the filtered schema. Finally, those relationships with none of its participants nor its descendants inside of $\mathcal{CS}_\mathcal{F}$ are identified as referentially-incomplete relationship types and, therefore, all of them are not processed in the next steps.

The process iterates over all the participants of a relationship. When it founds a participant outside of the filtered schema (line 8 of Alg. refalg:candidate), the process marks the relationship as partial if such participant have at least one descendant inside $\mathcal{CS}_\mathcal{F}$ (lines 9-10). Alternatively, if the participant does not have descendants in the filtered schema, the process marks the relationship as incomplete and finishes the analysis of its participants (lines 11-13).

Finally, the process classifies the relationship types according to the previous explanation (lines 17-20). It also includes the relationship types that are redefinitions of other relationship types in another set in order to use it in further steps. Note that the `isRedefinition` operation returns `true` or `false` depending on whether a relationship type redefines another relationship or not.

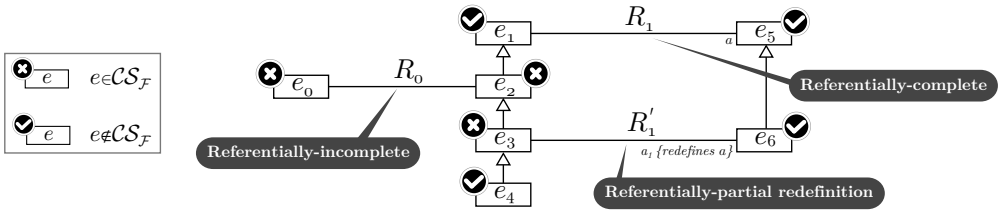


Figure 5.13. Classification of relationship types.

Figure 5.13 shows an example of classification of relationship types. The relationship $\mathcal{R}_0$ is a referentially-incomplete relationship because e_0 does not belong to the filtered conceptual schema. On the other hand, $\mathcal{R}_1$ is a referentially-complete relationship type that must be included into $\mathcal{R}_{rc}$ since all its participants (e_1 and e_5) belong to $\mathcal{CS}_\mathcal{F}$. Finally, $\mathcal{R}'_1$ is a referentially-partial relationship given that e_6 is a member of $\mathcal{CS}_\mathcal{F}$, and e_3 has e_4 as a descendant. Note that $\mathcal{R}'_1$ must be included into $\mathcal{R}_{rp}$ and also into the set $\mathcal{R}_d$ of selected redefinitions.

Process Referentially-Complete Relationship Types

After the previous classification, the set of referentially-complete relationship types must be included in the filtered conceptual schema since all its participants were already included in the previous stage of the filtering method (see Sect. 5.4.2).

Algorithm 5.9. Process referentially-complete relationship types.

```
1 for each r s.t. r ∈ R_rc and r ∉ R_d do
2   if r ∈ R then R_F = R_F ∪ {r} endif
3   if r ∈ R_b then R_bF = R_bF ∪ {r} endif
4 end
```

Note that the process does not include those referentially-complete relationships that redefine another relationship. The inclusion of a redefinition depends on the inclusion of the relationship that it redefines. Consequently, those redefinitions are processed in next steps.

Project Referentially-Partial Relationship Types

The next step projects the referentially-partial relationship types according to the entity and event types of the filtered conceptual schema in order to convert those relationships to referentially-complete ones.

Algorithm 5.10. Project referentially-partial relationship types.

```
1  R' = ∅ -- set of projected relationships
2  for each r ∈ R_rp do
3    r' = projectionOf(r)
4    R' = R' ∪ {r'}
5  end
```

As aforementioned, a referentially-partial relationship contains participants in its relationship ends that are not included in $\mathcal{CS}_{\mathcal{F}}$. However, the filtered conceptual schema contains at least one descendant of each of these participants. The `projectionOf` operation descends each relationship end whose owner entity or event type does not belong to $\mathcal{CS}_{\mathcal{F}}$ to its descendants that are in $\mathcal{CS}_{\mathcal{F}}$. Concretely, the relationship descends up to the lowest common ancestor (`LCA`) of all the descendants of the owner of the relationship end.

Algorithm 5.11. Function `projectionOf`.

```
1  function projectionOf(Relationship r)
2    r' = new Relationship
3    r'.name = name(r)
4    for each re ∈relationshipEnds(r) do
5      re' =copy(re)
6      e = owner(re)
7      if e ∉ CS_F then
8        e' = LCA(e, CS_F)
9        owner(re') = e'
10     endif
11     relationshipEnds(r') = relationshipEnds(r') ∪ {re'}
12   end
13   return r'
14 end
```

Note that `relationshipEnds(`r`)` returns the set of relationship ends of a relationship type r. Each relationship end contains its owner participant entity or event type, its multiplicity, and a role name of the end in the relationship.

The lowest common ancestor (`LCA`) is a well-known concept in graph theory and computer science [11]. Let $\mathcal{T}$ be a tree with n nodes. The lowest common ancestor is defined between a set of nodes $v_1, ..., v_n$ as the lowest node in $\mathcal{T}$ that has $v_1, ..., v_n$ as descendants (where we allow a node to be a descendant of itself). We define the `LCA` function as shown in Alg. 5.12.

Algorithm 5.12. Function `LCA`.

```
1  function LCA(e, CS_F)
2    e_LCA = e
3    C = descendants(e) ∩ CS_F
4    if size(C) = 1 then e_LCA = first(C)
5    else
6      V = directDescendants(e) -- V is an ordered set
7      while size(V) > 0 do
8        e' = first(V)
9        if {C \ {e'}} ⊆ descendants(e') then
10         if e' ∉ RS then e_LCA = e' endif
11         V = V ∪ directDescendants(e')
12       endif
13       V = V \ {e'}
14     end
15   endif
16   return e_LCA
17 end
```

Figure 5.14 shows an example of referentially-partial relationship type and its required projection. Since e_1 is not a member of the filtered conceptual schema, we need to project the relationship type $\mathcal{R}_1$ to the selected descendants of e_1 in order to transform $\mathcal{R}_1$ into a referentially-complete relationship type. To do so, we compute the lowest common ancestor of e_5 and e_6, which are the descendants of e_1. Therefore, the `LCS` for that pair of elements is e_3, which is not a member of the filtered schema. Consequently, $\mathcal{R}_1$ is projected to e_3. If such projected relationship type is finally included in $\mathcal{CS}_\mathcal{F}$, e_3 must be also added to it as an auxiliary entity or event type.

Note that the generalization relationships between e_3 and the pair e_5 and e_6 will be processed in the next stage of the filtering method (see Sect. 5.4.4).

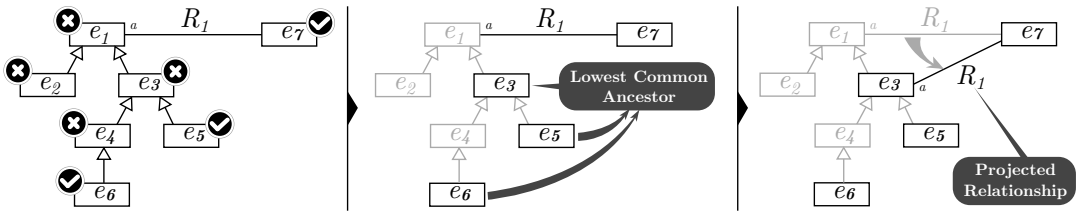


Figure 5.14. Projection of a referentially-partial relationship type.

Process Projected Relationship Types

The last step of this stage processes the projected relationship types and the redefinition relationships in order to include them in the filtered conceptual schema. This step is divided into three tasks. Firstly, we need to delete those redefinitions that are subsumed by others when projected, from the set of relationship types to include in $\mathcal{CS}_\mathcal{F}$.

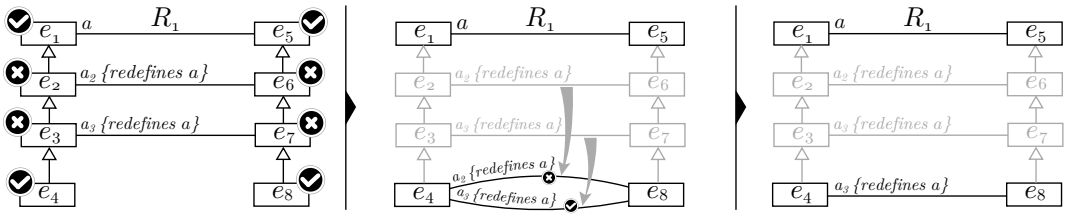


Figure 5.15. Subsumed redefinitions projected to the same set of participants.

The example in Fig. 5.15 presents two redefinitions of the same relationship type $\mathcal{R}_1$. Both redefinitions are projected to the same set of participants —their projections match. In this situation the projection of the deeper redefinition subsumes the projections of other redefinitions. Formally, we proceed as shown in Alg. 5.13.

Algorithm 5.13. Task 1: Delete subsumed redefinitions.

```
1  R_rcd = {R_d ∩ R_rc}  -- set of referentially-complete redefinitions
2  R'    = R' ∪ R_rcd    -- set of resulting relationships
3  R'_d  = {R_d ∩ R'}    -- set of projected redefinitions
4
5  -- Task 1: delete subsumed redefinitions
6  for each r_i, r_j ∈ {R_rcd ∪ R'_d} do
7    if match(r_i, r_j) then
8      if r_i isDeeperThan r_j then R' = R' \ {r_j} endif
9      if r_j isDeeperThan r_i then R' = R' \ {r_i} endif
10   endif
11 end
```

Note that the `isDeeperThan` boolean operator returns `true` whenever the second operand is a redefinition closer to the redefined relationship than the first operand. For the case of the example in Fig. 5.15 we have that the redefinition of $\mathcal{R}_1$ between e_3 and e_7 `isDeeperThan` the one between e_2 and e_6. Consequently, we delete the projection of the redefinition between e_2 and e_6 because it is subsumed by the projection of the redefinition between e_3 and e_7.

The second task we need to do is the combination of a relationship type with its redefinition relationship whenever both were referentially-partial relationships and are projected to the same set of participants. In this case, we are in the situation represented by the example in Fig. 5.16. Formally, we combine the two projected relationships as shown in Alg. 5.14.

Algorithm 5.14. Task 2: Combine redefinition with its redefined relationship.

```
1  for each r_d ∈ {{R_rcd ∪ R'_d} ∩ R'} do
2    r =redefinedRelationshipOf(r_d)
3    if r ∈ R' and match(r_d, r) then
4      r' = combinationOf(r, r_d)
5      R' = R' \ {r_d, r}
6      R' = R' ∪ {r'}
7    endif
8  end
```

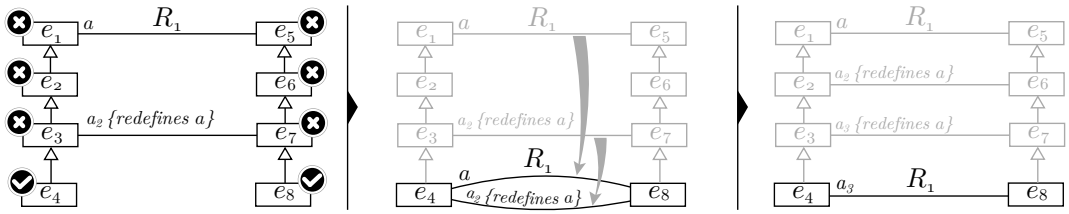


Figure 5.16. Combination of a projected relationship and its projected redefinition.

The process iterates over all the redefinition relationships that belong to the set $\mathcal{R}'$ of resulting relationships —those that were not deleted in previous steps. We obtain the redefined relationship for each of those redefinition relationships through the `redefinedRelationshipOf` operation. Then, we create a new relationship that substitutes the redefinition and redefined relationships when both share the same participants. The `match` operation checks whether the pair of relationships are defined with the same relationship ends under the same participants. The `combinationOf` operation returns the substitute relationship type as shown in Alg. 5.15.

Algorithm 5.15. Function `combinationOf`.

```
1  function combinationOf(r, r_d)
2    r' = new Relationship
3    r'.name = name(r)
4    for each re, re_d s.t. re ∈relationshipEnds(r) and
5        re_d ∈relationshipEnds(r_d) and match(re, re_d)
6    do
7      re' = new RelationshipEnd
8      re'.owner =owner(re_d)
9      re'.rolename =rolename(re_d)
10     re'.multiplicity =multiplicity(re_d)
11     relationshipEnds(r') = relationshipEnds(r') ∪ {re'}
12   end
13   return r'
14 end
```

The `relationshipEnds` operation returns the set of relationship ends of a relationship type. The `owner` operation returns the entity or event type that participates in the relationship type under a relationship end. The `rolename` operation returns the role name a relationship end plays in a relationship type. And the `multiplicity` operation returns the minimum and maximum number of instances of the owner type that participate in the relationship type where the relationship end belongs.

Basically, each relationship end of the new substitute relationship the process creates has the same characteristics than the equivalent relationship end of the actual redefinition relationship. The unique exception to this is the name of the substitute relationship, which takes the name of the redefined relationship.

For the case depicted in Fig. 5.16 we have a redefinition relationship between e_3 and e_7 that redefines the $\mathcal{R}_1$ relationship type between e_1 and e_5 —$\mathcal{R}_1$ is the redefined relationship. Since e_4 and e_8 are the unique participants that belong to the filtered schema, $\mathcal{R}_1$ and its redefinition relationship are both projected to the same set of participants. As explained, to avoid collision conflicts like this one, we combine the redefinition and the redefined relationships into a new relationship that substitutes them. In the case of Fig. 5.16, the new relationship type takes the name of the redefined relationship. The relationship ends of the substitute relationship are the same as the ones of the redefinition relationship (without the `redefines` keyword that indicates the redefinition of a relationship).

Finally, when we apply a projection to a relationship type, it is possible to require the inclusion of the entity or event types that were computed as lowest common ascendants into the filtered conceptual schema. To this end, we go through all the final relationships and before including them in $\mathcal{CS}_\mathcal{F}$ we include those participants that were auxiliary for the projection of referentially-partial relationship types. We proceed as shown in Alg. 5.16.

Algorithm 5.16. Task 3: Add auxiliary entity and event types.

```
1  for each r ∈ R' do
2    for each e ∈participants(r) do
3      if e ∉ CS_F then
4        if e ∈ E   then E_aux  = E_aux ∪ {e}   endif
5        if e ∈ E_b then E_baux = E_baux ∪ {e} endif
6      endif
7    end
8    -- add filtered relationships
9    if r ∈ R   then R_F  = R_F ∪ {r}   endif
10   if r ∈ R_b then R_bF = R_bF ∪ {r} endif
11 end
```

It is important to note that the set of auxiliary entity types $\mathcal{E}_{aux}$ is contained in $\mathcal{E}_\mathcal{F}$ and the set of auxiliary event types $\mathcal{E}_{baux}$ is contained in $\mathcal{E}_{b\mathcal{F}}$ (see Sect. 5.4.2). For the case of Fig. 5.14, the lowest common ascendant e_3 needs to be included in $\mathcal{CS}_\mathcal{F}$ as an auxiliary element that allows the connection of the elements in the filtered conceptual schema. Alternatively, the examples of Fig. 5.15 and Fig. 5.16 show projections of relationships to the elements e_4 and e_8, which are lowest common ascendants of themselves. In this case, since both e_4 and e_8 are already included in $\mathcal{CS}_\mathcal{F}$, no further actions are required. Note that the size threshold $\mathcal{K}$ of the input does not take into account those auxiliary elements.

Example: Magento - Stage 3

In the previous stage, our methodology selected the entity and event types that are included into the resulting filtered conceptual schema. Concretely, we obtained the entity types Customer, StoreView, Website, and Product, and the event types LogIn and LogOut.

Then, our method explores the relationship types in the conceptual schema of the Magento e-commerce system, and selects the referentially-complete ones to be part of the filtered schema. Those relationships have as participants, the previously selected entity or event types. In our

running example, the method obtained 16 referentially-complete relationships types. Of these, 4 are association classes. These relationships types are:

```
association IsCreatedInTheWebsite between
  Customer[*] role associatedCustomer
  Website[1] role websiteWhereIsAssociated

association accountVisibleInWebsite between
  Website[1..*] role websiteWhereIsVisible
  Customer[*] role visibleCustomer

association crossSellProductReflective between
  Product[*] role crossSellProduct
  Product[*] role productOfCrossSell

association isCreatedIn between
  StoreView[0..1] role storeViewWhereIsCreated
  Customer[*]

association relatedProductReflective between
  Product[*] role relatedProduct
  Product[*] role productOfRelated

association upSellProductReflective between
  Product[*] role upSellProduct
  Product[*] role productOfUpSell

associationclass ActivityInfoOfCustomerInStoreView between
  Customer[*] role customerWithInfoAbout
  StoreView[*] role storeViewWithInfoAbout

associationclass ActivityInfoOfCustomerInWebsite between
  Customer[*] role customerWithInfoAbout
  Website[*] role websiteWithInfoAbout

associationclass ProductInStoreView between
  Product[*]
  StoreView[*]

associationclass ProductInWebsite between
  Product[*]
  Website[*]

association CustomerWebsiteDefinesReadyToCompare between
  ActivityInfoOfCustomerInWebsite[*] role
      activityInfoOfCustomerInWebsiteOfReadyToCompareProduct
  Product[*] role readyToCompareProduct
```

```
association CustomerWebsiteDefinesRecentlyCompared between
  ActivityInfoOfCustomerInWebsite[*] role
      activityInfoOfCustomerInWebsiteOfRecentlyComparedProduct
  Product[*] role recentlyComparedProduct

association CustomerWebsiteDefinesRecentlyViewed between
  ActivityInfoOfCustomerInWebsite[*] role
      activityInfoOfCustomerInWebsiteOfRecentlyViewedProduct
  Product[*] role recentlyViewedProduct

association CustomerStoreViewDefinesReadyToCompare between
  ActivityInfoOfCustomerInStoreView[*] role
      activityInfoOfCustomerInStoreViewOfReadyToCompareProduct
  Product[*] role readyToCompareProduct

association CustomerStoreViewDefinesRecentlyCompared between
  ActivityInfoOfCustomerInStoreView[*] role
      activityInfoOfCustomerInStoreViewOfRecentlyComparedProduct
  Product[*] role recentlyComparedProduct

association CustomerStoreViewDefinesRecentlyViewed between
  ActivityInfoOfCustomerInStoreView[*] role
      activityInfoOfCustomerInStoreViewOfRecentlyViewedProduct
  Product[*] role recentlyViewedProduct
```

Since the conceptual schema of the Magento does not contain redefinitions of relationship types, the previous relationships are all included in the filtered conceptual schema.

In addition to it, our filtering method obtains a referentially-partial relationship type that must be projected in order to be included inside the filtered schema. This referentially-partial relationship type connects the entity type Customer with the event type ExistingCustomerEvent as follows:

```
association Customer_EexistingCustomerEvent between
  Customer[1] role customer
  ExistingCustomerEvent[*] role existingCustomerEvent
```

Since ExistingCustomerEvent was not selected in the second stage, but LogIn and LogOut are event types that are descendants of it, our method projects this relationship to such descendants in order to be included inside the filtered conceptual schema.

Figure 5.17 presents the projection of the aforementioned relationship to the descendants repeating the relationship type. Therefore, the original projection is transformed into two relationship types connecting Customer directly with LogIn and LogOut. However, the repetition of relationships may confuse the user, and decreases the direct traceability with the original schema.

To solve this situation, our methodology computes the lowest common ancestor of the pro-

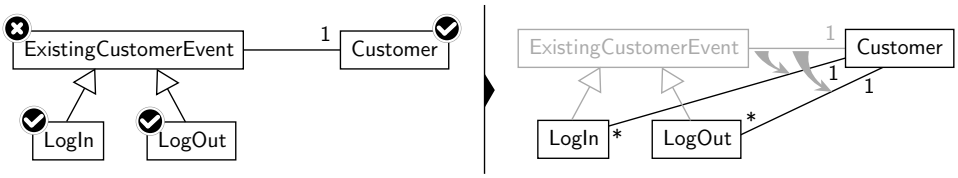


Figure 5.17. Projection of a relationship type that produces repeated relationship types.

jected participants in order to avoid repetition of relationship types. Figure 5.18 indicates that the lowest common ancestor of LogIn and LogOut for the relationship type that connects with Customer is the event type ExistingCustomerEvent. Therefore, we include ExistingCustomerEvent inside the filtered schema as an auxiliary event type and project the relationship type to it. In this example, the projection does not changes the relationship because lowest common ancestor ExistingCustomerEvent is the original participant.

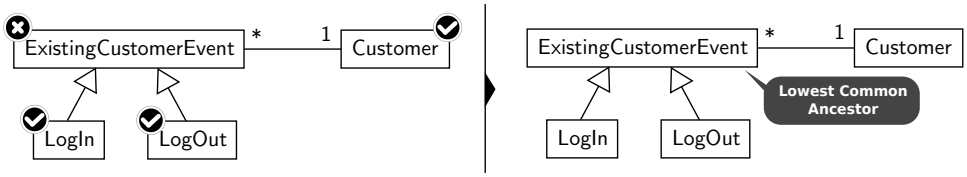


Figure 5.18. Projection of a relationship type to the lowest common ancestor of LogIn and LogOut.

Then, the relationship type is transformed into a referentially-complete relationship and is included inside the filtered conceptual schema:

```
association Customer_ExistingCustomerEvent between
  Customer[1] role customer
  ExistingCustomerEvent[*] role existingCustomerEvent -- auxiliary event
```

5.4.4 Stage 4: Generalizations Processing

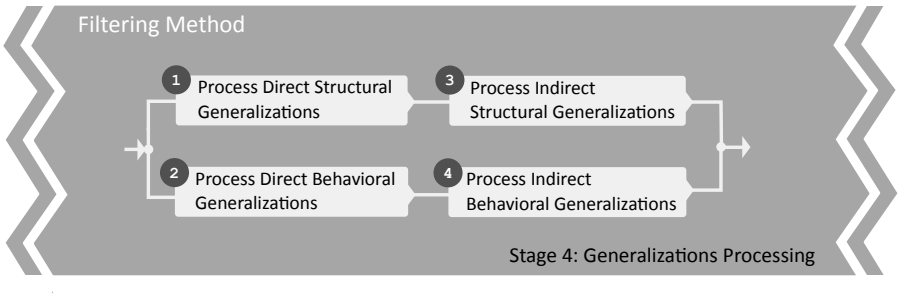


Figure 5.19. Stage 4: Generalizations Processing.

The fourth stage of the filtering method deals with the process of selecting the generalization relationships from the original schema that must be part of the resulting filtered schema. The inclusion of generalization relationships will help in the task of maintaining the semantics of the original schema in the filtered one.

This stage contains four steps to perform that function according to whether the generalization relationships are direct or indirect, or belong to the structural or behavioral subschemas of a large conceptual schema. Figure 5.19 shows the four steps and their sequential order of application.

Process Direct Structural Generalizations

At this point of the filtering process, the resulting filtered schema contains its entity, event, and relationship types. To keep the original semantics, a filtering method also needs to include the generalization relationships into the output schema. The first step in this stage deals with those generalizations whose general and specific elements are entity types included in the filtered schema. This process constructs the set $\mathcal{G}_\mathcal{F}$ of $\mathcal{CS}_\mathcal{F}$ as shown in Alg. 5.17.

Algorithm 5.17. Compute direct generalizations of $\mathcal{G}_\mathcal{F}$.

```
1 G_F = ∅
2 for each g ∈ G do
3   if general(g) ∈ E_F and specific(g) ∈ E_F then
4     G_F = G_F ∪ {g}
5   endif
6 end
```

The `general` and `specific` operations return the general and specific elements of a generalization relationship, respectively. Note that if e_i, e_j are entity types and we have a generalization relationship g so that e_j is a descendant of e_i ($e_i \leftarrow e_j$), then `general(`g`)`$= e_i$ and `specific(`g`)`$= e_j$. Basically, we add into the set $\mathcal{G}_\mathcal{F}$ of filtered generalization relationships of the structural subschema those generalizations whose general and specific elements are entity types that belong to the filtered schema $\mathcal{CS}_\mathcal{F}$.

Process Direct Behavioral Generalizations

The following step deals with those generalizations whose general and specific elements are event types included in the filtered schema. This process constructs the set $\mathcal{G}_{b\mathcal{F}}$ of $\mathcal{CS}_{\mathcal{F}}$ in the same way the method did with the generalization relationships between entity types of the structural subschema. Note that the `general` and `specific` operations have the same behavior and return the direct ascendant or descendant event type in a generalization relationship, respectively.

Algorithm 5.18. Compute direct generalizations of $\mathcal{G}_{b\mathcal{F}}$.

```
1 G_bF = ∅
2 for each g ∈ G do
3   if general(g) ∈ E_bF and specific(g) ∈ E_bF then
4     G_bF = G_bF ∪ {g}
5   endif
6 end
```

Process Indirect Structural Generalizations

The previous steps add into the filtered schema those generalization relationships of the original schema whose members are already included into the filtered schema. However, it is possible to have a pair of entity types e_i, e_j of $\mathcal{E}_{\mathcal{F}}$ so that e_i is an indirect ascendant of e_j in $\mathcal{CS}$ but they are not connected through generalization relationships $g \in \mathcal{G}_{\mathcal{F}}$ of $\mathcal{CS}_{\mathcal{F}}$.

The projection of relationship types from the third stage of the filtering method (see Sect. 5.4.3) is an operation that connects distant entity and event types in order to obtain a fully-connected schema. However, at this point of the method there are pairs of schema elements in the filtered schema that were members of the same hierarchy in the original schema but need to be directly connected through generalization relationships that are direct in $\mathcal{CS}_{\mathcal{F}}$ but a path of generalizations in $\mathcal{CS}$.

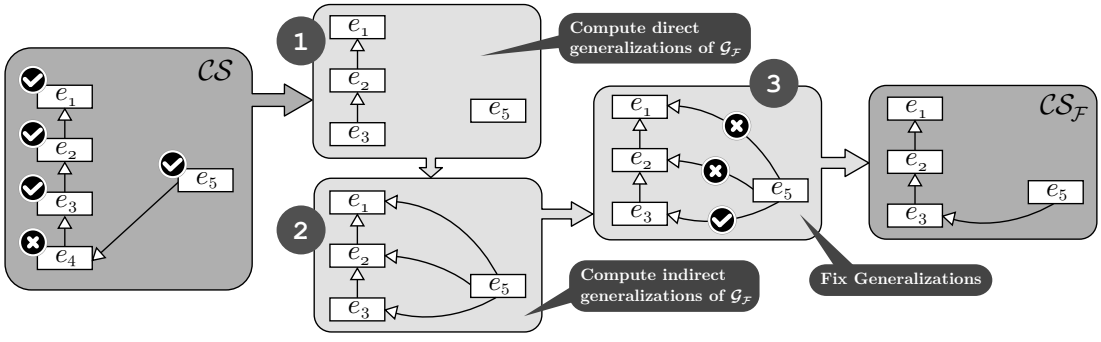


Figure 5.20. Process to filter generalization relationships.

Figure 5.20 presents an example of the overall process to filter the generalization relationships. It shows five entity types in the original schema $\mathcal{CS}$, and four of them —e_1, e_2, e_3 and e_5— are selected to be part of the filtered schema. The first step computes all direct generalization

relationships that apply to the selected entity types and adds them into $\mathcal{G}_\mathcal{F}$. As a result, the entity type e_5 that was a descendant in the hierarchy of generalizations $e_1 \leftarrow e_2 \leftarrow e_3 \leftarrow e_4 \leftarrow e_5$ of the original schema, it is now an isolated entity type in the filtered schema, which also contains e_1, e_2, and e_3. To avoid this inconsistency, we create indirect generalizations as shown in Alg. 5.19.

Algorithm 5.19. Compute indirect generalizations of $\mathcal{G}_\mathcal{F}$.

```
1  for each e_i, e_j ∈ E_F do
2    if (∄ g_F ∈ G_F s.t. general(g_F) = e_i and specific(g_F) = e_j)
3    and
4    e_i ∈ ascendants(e_j, G)
5    then
6      g_ij = new Generalization e_i ← e_j
7      G_F = G_F ∪ {g_ij}
8    endif
9  end
10 G_F = fixGeneralizations(G_F)
```

It is necessary to check each pair of entity types from the filtered schema. The method creates a new generalization between those entity types and adds it into the filtered schema whenever there is no generalization relationship between them in the filtered schema but one of the entity types was an ascendant of the other in the original schema. Note that the `ascendants`$(e,\ \mathcal{G})$ operation returns the set of all the entity types that are upper-level members in a hierarchy of generalizations from $\mathcal{G}$ with respect to the entity type e.

Following this process, Fig. 5.20 shows that there are three new generalization relationships connecting e_5 to e_1, e_2, and e_3 because these three entity types are ascendants of e_5 in the original schema —in fact, the process also creates a generalization relationship connecting e_3 with e_1, but it has been skipped for the sake of simplicity. However, only one of the three new generalizations is necessary to maintain the original semantics in the filtered schema. The generalization $e_1 \leftarrow e_3$ subsumes the generalizations $e_1 \leftarrow e_5$ and $e_2 \leftarrow e_5$. The `fixGeneralizations` operation deletes all the generalization relationships that are not necessary.

Algorithm 5.20. Function `fixGeneralizations`.

```
1  function fixGeneralizations(G)
2    G' = copy(G)
3    for each g_i, g_j ∈ G do
4      if specific(g_i) = specific(g_j) then
5        if general(g_i) ∈ ascendants(general(g_j, G)) then
6          G' = G' \ {g_i}
7        endif
8        if general(g_j) ∈ ascendants(general(g_i, G)) then
9          G' = G' \ {g_j}
10       endif
11     endif
12   end
13   return G'
14 end
```

Basically, a generalization g is unnecessary —and therefore, we delete it— whenever exists another generalization g' that share the same specific entity type with g but the general entity type of g is an ascendant of the general entity type of g'. For the case of Fig. 5.20, the `fixGeneralizations` operation deletes the generalizations $e_1 \leftarrow e_5$ and $e_2 \leftarrow e_5$ because the generalization $e_3 \leftarrow e_5$ shares the specific entity type e_5 with the two others and then the general entity types e_1 and e_2 are ascendants of e_3. At the end of the process we have a consistent filtered conceptual schema $\mathcal{CS}_{\mathcal{F}}$ as indicated in the right side of Fig. 5.20.

Process Indirect Behavioral Generalizations

The last step to complete the process of filtering generalization relationships deals with the indirect generalizations between event types. The methodology we follow is the same as we presented for the case of entity types. It is possible to have a pair of event types ev_i, ev_j of $\mathcal{E}_{b\mathcal{F}}$ so that ev_i is an indirect ascendant of ev_j in $\mathcal{CS}$ but they are not connected through generalization relationships $g \in \mathcal{G}_{b\mathcal{F}}$ of $\mathcal{CS}_{\mathcal{F}}$. To avoid this inconsistency, we create indirect generalizations.

Algorithm 5.21. Compute indirect generalizations of $\mathcal{G}_{b\mathcal{F}}$.

```
1  for each ev_i, ev_j ∈ E_bF do
2    if (∄ g_F ∈ G_bF s.t. general(g_F) = ev_i and specific(g_F) = ev_j)
3    and
4    ev_i ∈ ascendants(ev_j, G_b)
5    then
6      g_ij = new Generalization ev_i ← ev_j
7      G_bF = G_bF ∪ {g_ij}
8    endif
9  end
10 G_bF = fixGeneralizations(G_bF)
```

It is necessary to check each pair of event types from the filtered schema. The method creates a new generalization between those event types and adds it into the filtered schema whenever there is no generalization relationship between them in the filtered schema but one of the event types was an ascendant of the other in the original schema. Similarly to the case of entity types, this process generates unnecessary generalization relationships that the `fixGeneralizations` operation deletes.

Example: Magento - Stage 4

To continue with the example of Magento, our filtering method explores the schema and obtains two direct behavioral generalization relationships:

```
LogIn  directly inherits from ExistingCustomerEvent
LogOut directly inherits from ExistingCustomerEvent
```

Since the entity types selected in the second stage are not members of the same hierarchy, there are no more indirect generalization relationships to process.

5.4.5 Stage 5: Schema Rules Processing

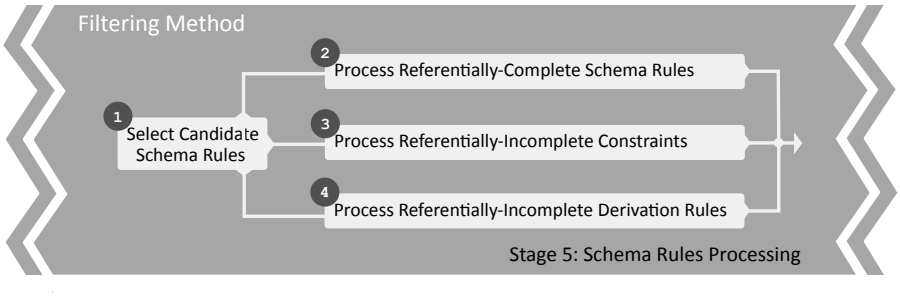


Figure 5.21. Stage 5: Schema Rules Processing.

The fifth stage of the filtering method deals with the process of selecting the schema rules from the original schema that must be part of the resulting filtered schema. This stage contains four steps to perform that function. Figure 5.21 shows the steps and their sequential order of application.

Select Candidate Schema Rules

The first step of this stage selects those schema rules that will be part of the output of the method. Concretely, we are interested in integrity constraints and derivations rules whose context is an element that belongs to the filtered schema. To this end, the process puts the candidate schema rules into the set $\mathcal{S}$ as shown in Alg. 5.22.

Algorithm 5.22. Compute candidate schema rules.

```
1 S = ∅
2 for each r ∈ {C ∪ D ∪ Cb} do
3   if context(r) ∈ CS_F then S = S ∪ {r}
4 end
```

Note that the `context` operation returns the attribute, operation, entity, event, or relationship type where the constraint or derivation rule is defined. If that element was included in $\mathcal{CS}_{\mathcal{F}}$ then the associated rule is a candidate rule.

Process Referentially-Complete Schema Rules

We say that a schema rule is referentially-complete whether all its participants belong to the filtered conceptual schema. The participants of a schema rule are the set of attributes, operations, relationship types, and entity and event types that are referenced in the expressions that conform the rule.

This step of the filtering method explores every candidate schema rule from the previous step in order to check whether all its participants are members of the filtered conceptual schema $\mathcal{CS}_{\mathcal{F}}$. In that case, each referentially-complete schema rule is included into $\mathcal{C}_{\mathcal{F}}$, $\mathcal{D}_{\mathcal{F}}$, or $\mathcal{C}_{b\mathcal{F}}$ according to its category —constraint, derivation rule, or constraint from the behavioral subschema.

On the other side, if any of the participants of a candidate rule does not belong to the filtered schema, the process includes the rule into the set $\mathcal{S}_{ri}$ of referentially-incomplete rules. These rules are analyzed in the next two steps inside this stage.

Algorithm 5.23. Compute referentially-complete schema rules.

```
1  C_F = ∅
2  D_F = ∅
3  C_bF = ∅
4  S_ri = ∅ -- set of referentially-incomplete rules
5  for each r ∈ S do
6    for each e ∈participants(r) do
7      if e ∉ CS_F
8      then
9        S_ri = S_ri ∪ {r}
10       break
11     endif
12   end
13   if r ∉ S_ri then
14     if r ∈ C then C_F = C_F ∪ {r} endif
15     if r ∈ D then D_F = D_F ∪ {r} endif
16     if r ∈ C_b then C_bF = C_bF ∪ {r} endif
17   endif
18 end
```

Process Referentially-Incomplete Constraints

This step deals with the integrity constraints from the structural and behavioral subschemas that are referentially-incomplete. These constraints cannot be directly included into the filtered conceptual schema in order to maintain the semantics —they contain participants which are not in $\mathcal{CS}_{\mathcal{F}}$. Consequently, we propose to indicate their existence by including an empty constraint into the filtered conceptual schema for each of these constraints, maintaining its original name and context element. We believe that the header of a constraint serves as the indicator of its semantics, and may be used as starting point for a new execution of the filtering method whenever the user has interest in it.

Algorithm 5.24. Compute referentially-incomplete constraints.

```
1  for each r ∈ S_ri do
2    r' = new Constraint
3    r'.context = context(r)
4    r'.name = name(r)
5    r'.expression = empty
6    if r ∈ C then C_F = C_F ∪ {r'} endif
7    if r ∈ C_b then C_bF = C_bF ∪ {r'} endif
8  end
```

An entity or event type can be referenced by means of its attributes, its participations in relationship types or by referencing the type itself. As an example, the integrity constraint `ic1` in Fig. 5.22 has A and B as its participants. A is referenced as the context of the constraint and by means of its attribute *a1* in the OCL expression `self.a1`. Also, B is referenced by means of its attribute *b1* in the OCL expression `self.b.b1`. Our method only includes `ic1` into $\mathcal{C}_{\mathcal{F}}$ or $\mathcal{C}_{b\mathcal{F}}$ if both A and B are entity or event types in $\mathcal{E}_{\mathcal{F}}$. In case that only A, which is the context of `ic1`, belongs to the filtered schema then we include the header of the constraint in the output as "**context A inv** ic1".

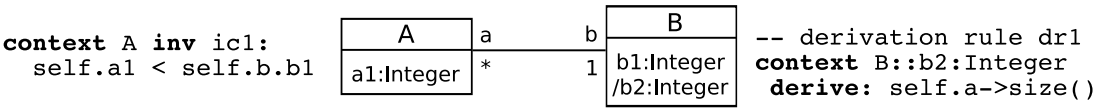


Figure 5.22. Example of integrity constraint (`ic1`) and derivation rule (`dr1`).

Process Referentially-Incomplete Derivation Rules

Finally, the last step processes the derivation rules that were selected as referentially-incomplete. As in the case of those constraints with participants out of the filtered schema, we cannot directly include these derivation rules in the output of the method. Rather than that, we propose to mark as materialized the context element of each referentially-incomplete derivation rule as shown in Alg. 5.25. The context element may be an entity type, an event type, a relationship type, an attribute, or an association end of a relationship type.

Algorithm 5.25. Compute referentially-incomplete derivation rules.

```
1 for each r ∈ S_ri do
2   c = context(r)
3   markAsMaterilized(c in CS_F)
4 end
```

The operation `markAsMaterialized` takes the context of a derivation rule as a parameter and indicates that such element is not derived in $\mathcal{CS}_{\mathcal{F}}$ because the derivation rule is not included in the final output due to it references elements that are not included to be part of the filtered conceptual schema.

The derivation rule `dr1` in Fig. 5.22 is included in $\mathcal{D}_{\mathcal{F}}$ if both A and B are entity or event types in $\mathcal{CS}_{\mathcal{F}}$. If only $B \in \mathcal{CS}_{\mathcal{F}}$, our method marks the derived attribute *b2* as materialized (hiding the "/" symbol that precedes the attribute definition in B) and does not include `dr1` in $\mathcal{D}_{\mathcal{F}}$ because that derivation rule also references A which is not included in the filtered schema.

Example: Magento - Stage 5

To continue with the example of Magento, our filtering method obtains the schema rules that are referentially-complete and includes them into the filtered conceptual schema. Those referentially-complete rules only reference entity, event, and relationship types (including association classes) that were selected to be part of the filtered schema, and are defined in the context of any of these schema elements. Concretely, the method selects those rules that are referentially-complete and are defined in the context of the entity types Customer, StoreView, Website, and Product; the event types LogIn, LogOut, and ExistingCustomerEvent; and the association classes ActivityInfoOfCustomerInStoreView, ActivityInfoOfCustomerInWebsite, ProductInStoreView, and ProductInWebsite. The selected schema rules for this example are:

```
context ActivityInfoOfCustomerInStoreView
  inv hasNotMatchingComparedAndRecentlyComparedProducts:
    self.recentlyComparedProduct
    ->excludesAll(self.readyToCompareProduct)

context ActivityInfoOfCustomerInWebsite
  inv hasNotMatchingComparedAndRecentlyComparedProducts:
    self.recentlyComparedProduct
    ->excludesAll (self.readyToCompareProduct)

context Product
  inv isIdentifiedBySku:
    Product.allInstances()->isUnique(sku)
  inv isSpecifiedInAllStoreViews:
    self.storeView->includesAll(StoreView.allInstances())
  inv isSpecifiedInAllWebsites:
    self.website->includesAll(Website.allInstances())
```

Additionally, the filtering method selects all the schema rules defined in the context of schema elements included in the filtered conceptual schema that reference other elements out of the filtered schema. These rules are referentially-partial schema rules, and our method includes their context and name into the filtered schema in order to inform the user of their existence in the original schema. The contexts and names selected for the referentially-partial rules in this example are:

```
context ActivityInfoOfCustomerInStoreView
  inv hasNotUsedTellToAFriendTooMuch

context ActivityInfoOfCustomerInWebsite
  inv definesWishedProductsOnlyIfAllowed

context Customer
 inv isCreatedInACorrectStoreView
 inv isIdentifiedByItsEmailAndTheWebsitesWhereIsVisible
 inv isNotSubscribedTwiceInAStoreView
```

```
context LogIn
 inv CustomerIsNotLoggedIn
 inv CustomerIsVisibleInWebsite

context Product
  inv globallyRatedAttributeSubsetsAbleToRateAttribute
  inv hasAGlobalWebsiteAndStoreViewRatingForAllAbleToRateProducts
  inv hasARatingForAllRequiredAttributes
  inv hasInventoryPropertiesOnlyIfNeeded
  inv hasOptionsOnlyIfNeeded
  inv hasQuantityPropertyWhenNeeded
  inv isOnlyRatedForItsAbleToRateProducts

context ProductInStoreView
  inv baseImagePathIsDefinedOnlyForAssociatedProductTypes
  inv baseImagePathIsMandatoryIfDefined
  inv descriptionIsDefinedOnlyForAssociatedProductTypes
  inv descriptionIsMandatoryIfDefined
  inv giftMessageAllowedIsDefinedOnlyForAssociatedProductTypes
  inv giftMessageAllowedIsMandatoryIfDefined
  inv imageGalleryPathIsDefinedOnlyForAssociatedProductTypes
  inv imageGalleryPathIsMandatoryIfDefined
  inv isAvailableForGoogleCheckoutIsDefinedOnlyForAssociatedProductTypes
  inv isAvailableForGoogleCheckoutIsMandatoryIfDefined
  inv isNewFromIsDefinedOnlyForAssociatedProductTypes
  inv isNewFromIsMandatoryIfDefined
  inv isNewUntilIsDefinedOnlyForAssociatedProductTypes
  inv isNewUntilIsMandatoryIfDefined
  inv metaDescriptionIsDefinedOnlyForAssociatedProductTypes
  inv metaDescriptionIsMandatoryIfDefined
  inv metaKeywordIsDefinedOnlyForAssociatedProductTypes
  inv metaKeywordIsMandatoryIfDefined
  inv metaTitleIsDefinedOnlyForAssociatedProductTypes
  inv metaTitleIsMandatoryIfDefined
  inv shortDescriptionIsDefinedOnlyForAssociatedProductTypes
  inv shortDescriptionIsMandatoryIfDefined
  inv smallImagePathIsDefinedOnlyForAssociatedProductTypes
  inv smallImagePathIsMandatoryIfDefined
  inv specialNetPriceFromIsDefinedOnlyForAssociatedProductTypes
  inv specialNetPriceFromIsMandatoryIfDefined
  inv specialNetPriceUntilIsDefinedOnlyForAssociatedProductTypes
  inv specialNetPriceUntilIsMandatoryIfDefined
  inv thumbnailPathIsDefinedOnlyForAssociatedProductTypes
  inv thumbnailPathIsMandatoryIfDefined
  inv urlKeyIsDefinedOnlyForAssociatedProductTypes
  inv urlKeyIsMandatoryIfDefined
  inv visibleOnCatalogIsDefinedOnlyForAssociatedProductTypes
  inv visibleOnCatalogIsMandatoryIfDefined
```

```
  inv visibleOnSearchIsDefinedOnlyForAssociatedProductTypes
  inv visibleOnSearchIsMandatoryIfDefined
  inv weightIsDefinedOnlyForAssociatedProductTypes

context ProductInWebsite
  inv netPriceIsDefinedOnlyForAssociatedProductTypes
  inv specialNetPriceIsDefinedOnlyForAssociatedProductTypes
  inv specialNetPriceIsMandatoryIfDefined

context StoreView
  inv doesNotHaveTwoCategoriesWithTheSameRedefinedName
  inv doesNotHaveTwoPaymentMethodsWithTheSameRedefinedName

context Website
  inv hasOnePaymentMethodEnabledAtLeast
  inv hasOneShippingMethodEnabledAtLeast

context LogIn::effect() post

context LogOut::effect() post
```

Note that the previous referentially-incomplete schema rules help to realize the complexity of each element in the filtered schema. In this example, it is easy to see that the association class ProductInStoreView is a complex schema element because there are 37 referentially-incomplete integrity constraints defined in its context. With this information, a user interested in ProductInStoreView may start a new iteration of the filtering method focusing in that element.

5.4.6 Stage 6: Data Types Processing

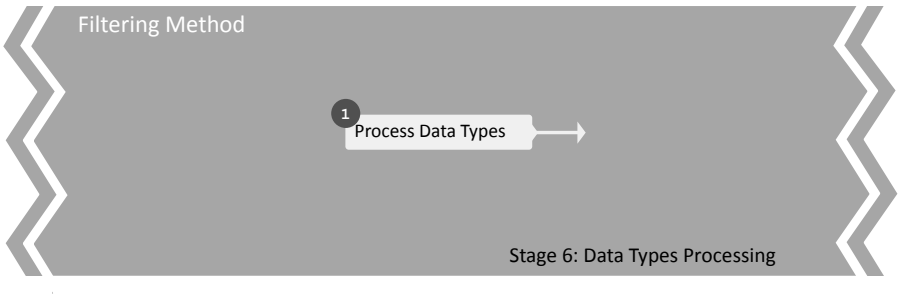


Figure 5.23. Stage 6: Data Types Processing.

The next stage of the filtering method deals with the process of selecting the data types that are included in the resulting filtered conceptual schema. The main step inside this stage performs the selection of such data types from the original schema that are referenced by the specific elements already included in the filtered schema.

Process Data Types

At this point of the filtering process, the resulting filtered schema contains its entity, event, and relationship types, as well as its generalizations and schema rules. To complete the overall process, it is mandatory to include in the result all the required data types referenced by those schema elements in order to maintain the consistency of the filtered schema. Formally, we proceed as shown in Alg. 5.26.

Algorithm 5.26. Compute data types of $\mathcal{CS}_\mathcal{F}$.

```
1 T_F = ∅
2 for each t ∈ T do
3   if t ⇌ CS_F[4] then T_F = T_F ∪ {t} endif
4 end
```

The data types included within the set of data types $\mathcal{T}_\mathcal{F}$ of the filtered schema $\mathcal{CS}_\mathcal{F}$ are a subset of the original data types $\mathcal{T}$ of $\mathcal{CS}$. Concretely, we include inside $\mathcal{T}_\mathcal{F}$ all those data types that define the types of attributes of entity or event types of $\mathcal{CS}_\mathcal{F}$, and also those data types that are used and referenced in integrity constraints or derivation rules of $\mathcal{CS}_\mathcal{F}$. Note that a data type may be a primitive type (e.g. `Integer`, `Boolean`, `String`), or an enumeration with enumeration literals (e.g. `Gender{Male, Female, Unknown}`).

It is important to include into the filtered conceptual schema those data types that are not primitive in order to help the user to understand the semantics of the different attributes of entity and event types.

[4]If t is a data type and $\mathcal{C}$ is a conceptual schema, we say that $t \rightleftharpoons \mathcal{C}$ if and only if t is used in or referenced by any of the schema elements (e.g. attributes, schema rules) of $\mathcal{C}$.

Example: Magento - Stage 6

Taking into account the elements selected in the previous stages of our filtering method, the data types that are referenced by them, and need to be included in the filtered schema are:

```
enumeration WeekDay
  {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}

enumeration Status
  {Enabled, Disabled}

enumeration BackOrderPolicy
  {NoBackorders, AllowQtyBelowZero, AllowQtyBelowZeroAndNotifyCustomer}

enumeration ProductStatus
  {InStock, OutOfStock}

enumeration ProductType
  {Simple, Grouped, Configurable, Downloadable, Virtual, Bundle}

datatype PhoneNumber

datatype DateTime
 attributes
  date: Date
  time: Time

datatype Date
 attributes
  day: Integer
  month: Integer
  year: Integer

datatype Time
 attributes
  hour: Integer
  min: Integer
  sec: Integer

datatype Address
 attributes
  firstName: String[1]
  middleName: String[0..1]
  lastName: String[1]
  namePrefix: String [0..1]
  nameSuffix: String[0..1]
  company: String[0..1]
  streetAddress: String[1]
  telephone: String[1]
  fax: String[0..1]
```

5.4.7 Stage 7: Presentation

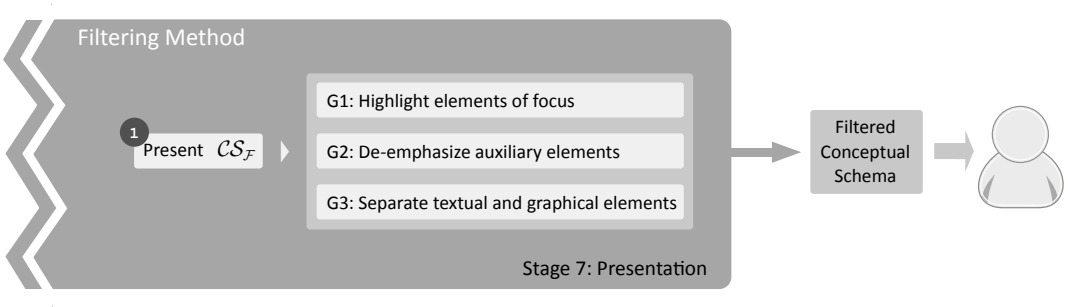


Figure 5.24. Stage 7: Presentation.

The next stage of the filtering method deals with the process of graphically presenting to the user the elements included in the resulting filtered conceptual schema. The main step inside this stage indicates some guidelines that must be followed in order to increase the understandability of the filtered schema by its intended user.

Present Filtered Conceptual Schema $\mathcal{CS}_\mathcal{F}$

The last step of the filtering method post-processes the filtered conceptual schema obtained through the previous steps in order to make it more attractive to the end user. The modification of the visual aspect must follow a set of guidelines to highlight those elements in the schema that are more interesting to the user and to help to reduce the comprehension effort. Figure 5.25 shows an example of a filtered conceptual schema. We can assume that such schema was obtained by the execution of the filtering method over a large conceptual schema. We will use this plain schema to explain the required modifications we introduce by following the next presentation guidelines.

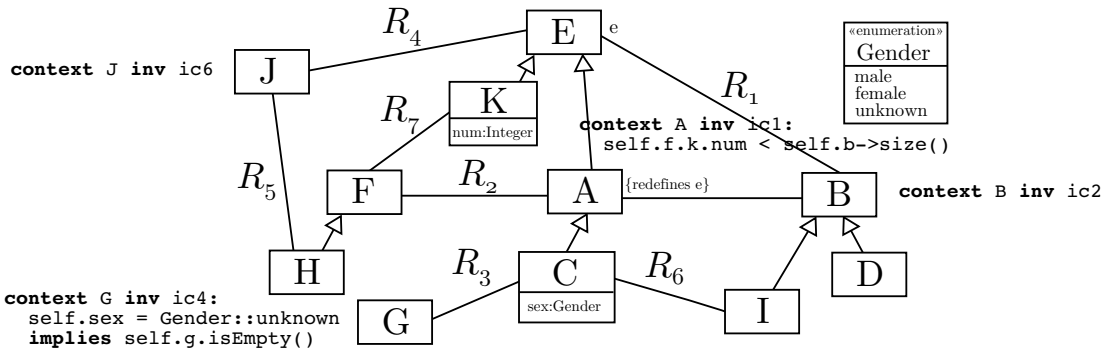


Figure 5.25. Example of filtered conceptual schema without presentation enhancement.

Guideline 1: Highlight elements of focus

The conceptual schema that the filtering method produces contains the elements in the user focus. Therefore, it is of great importance to highlight those elements in order to help the user to quickly identify them. Consequently, we draw them with a darker color or thicker shape as indicated in Fig. 5.26 where the entity types A, D, F, and I conformed an hypothetical focus set. In case of a focus set containing schema rules, the proposed guideline to highlight them consists of putting the schema rules of interest inside of an square box filled with a darker color.

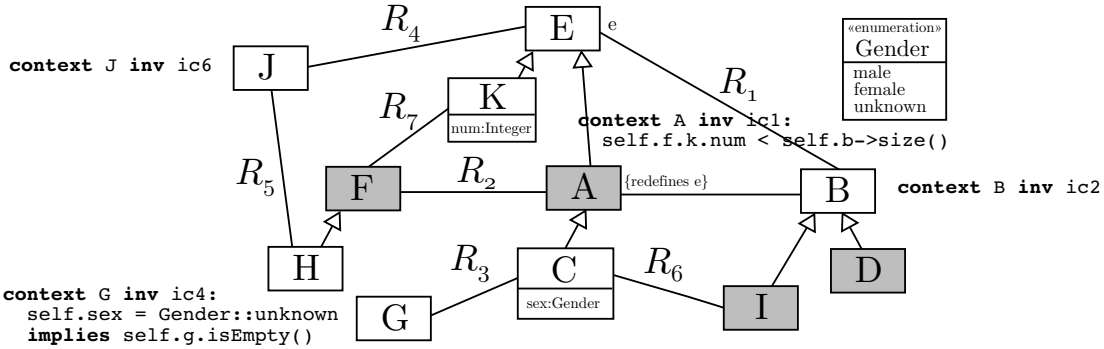


Figure 5.26. Example of filtered conceptual schema highlighting elements of focus.

Comparing this schema with the one in Fig. 5.25, it is easier to focus on the highlighted elements and therefore keep the user in context. By highlighting the elements in the focus set, we maintain the traceability between the input and the output of the filtering method.

Guideline 2: De-emphasize auxiliary elements

In addition to the previous highlighting of important elements in the schema we need to do the inverse operation to those elements that are of less relevance to the user according to the filtering method. We should de-emphasize the auxiliary entity and event types introduced when projecting relationship types, the relationships between them, and the generalization relationships that indicate indirect generalizations between a pair of elements. Thus, we draw those elements with a lighter color to reduce their presence in the final schema.

Figure 5.27 shows that E and B are auxiliary entity types, and that the generalizations between E and A, A and C, and B and I are indirect in the original schema. Additionally, the relationship type $\mathcal{R}_1$ is de-emphasized because both E and B, which are its participants, are auxiliary entity types —they belong to $\mathcal{E}_{aux}$.

The auxiliary elements in the filtered conceptual schema help the user by connecting the resulting elements that are not directly connected in the original schema. The indirect generalization relationships are key constructions that allow the user to figure out the knowledge that the schema contain. By following the principles of focus+context techniques, we simplify the required understandability effort and increase the manageability of the schema.

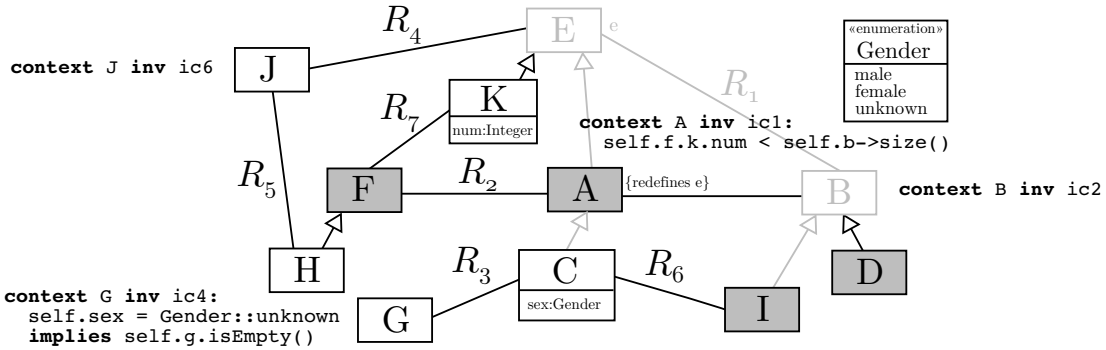


Figure 5.27. Example of filtered conceptual schema de-emphasizing auxiliary elements.

Guideline 3: Separate textual and graphical elements

Finally, it is also important to divide the resulting schema in two parts. The first one contains the graphical elements, whereas the second one presents the schema rules that are written in OCL. Such division helps the user to focus on the semantics of the graphical part at first, and then complete the information with additional semantics from the textual rules that depend on the previous understandability of the schema. Figure 5.28 shows that division.

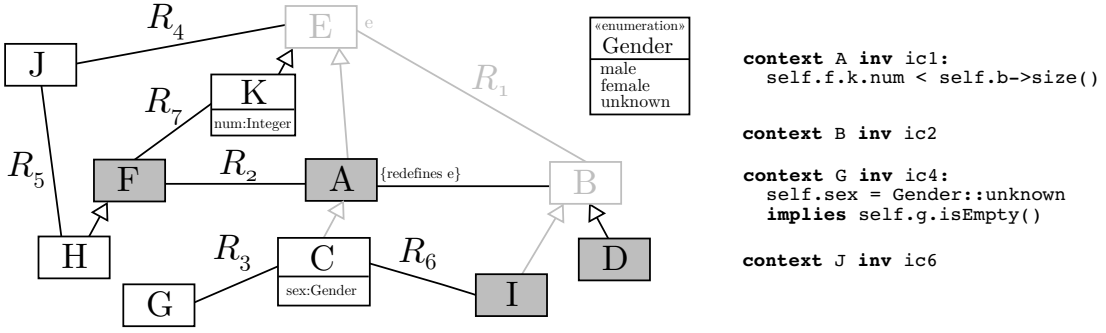


Figure 5.28. Example of filtered conceptual schema following presentation guidelines.

In addition, it may be possible to obtain a long list of referentially-incomplete schema rules to show in the graphical representation of the schema. In situations like this one, a filtering system that implements our filtering methodology may hide such list of context and names of integrity constraints and derivation rules. A better interactive user interface may provide the list of referentially-incomplete schema rules of each schema element when clicking on it, in a separate view. This focus+context alternative helps the user on his task of understanding the filtered conceptual schema.

Example: Magento - Stage 7

The last step of our example takes all the schema elements previously selected to be part of the result and constructs a graphical representation of the filtered conceptual schema that includes them. Going back to the first stage, the filtering scenario proposed in this example describes a user of the conceptual schema of the Magento e-commerce system that wanted

to explore the knowledge about a particular portion of the schema. Concretely, the user was interested in the functionalities related to the log in and log out of customers within an online store developed with the Magento system. The user constructed the input of the method as follows:

```
CS  = Magento
FS  = {LogIn, LogOut, Customer}
RS  = ∅
K   = 6
I   = CEntityRank Extended
```

Through the different stages of the filtering method, we have explored the elements that were filtered from the large schema to be included in the resulting filtered schema. Figures 5.29 and 5.30 presents the graphical representation of the filtered conceptual schema that corresponds to the user requirements of the previous input.

The user obtains a filtered schema of small size in comparison to the original schema but with knowledge of interest with regard to the functionalities that the user wanted to explore. It is easy to see the event types LogIn and LogOut, and the entity type Customer, as the elements of focus. These event types are both direct descendants of the ExistingCustomerEvent, which is an auxiliary event type selected by the filtering method to avoid the repetition of the relationship type that connects with Customer.

Exploring the filtered schema, the user realizes that both LogIn and LogOut are related to a Customer, which may be defined in the context of a store view, and is associated to a website, although its account can be visible in several websites. Also, there are two association classes that keep the activity information of the customer in store views and websites within the Magento system. These association classes —ActivityInfoOfCustomerInStoreView and ActivityInfoOfCustomerInWebsite— are in charge of maintaining the interaction of the Customer with Products. Concretely, these association classes are connected to the recently viewed products, the recently compared ones, and the products that are ready to be compared. Finally, there are two more association classes —ProductInWebsite and ProductInStoreView, in Fig. 5.30— that connect the different store views and websites with the products that they contain.

Figure 5.30(right) includes the different datatypes and enumeration types that are necessary to keep the semantics of the attributes of the elements in the filtered conceptual schema. Note that the schema shows the attributes of the datatypes and the enumeration literals of the enumeration types. The last part of this figure indicates the referentially-complete schema rules for the filtered schema. The user can review them and realize that the set of recently compared products and the products that are ready to be compared by a particular customer must be disjoint. Also, the user may discover additional business rules for the entity type Product, which is identified by its stock-keeping unit (SKU) code and must be specified in all store views and websites. All these knowledge allows the user to understand the fragments of interest of the schema without requiring to explore all the elements specified within it.

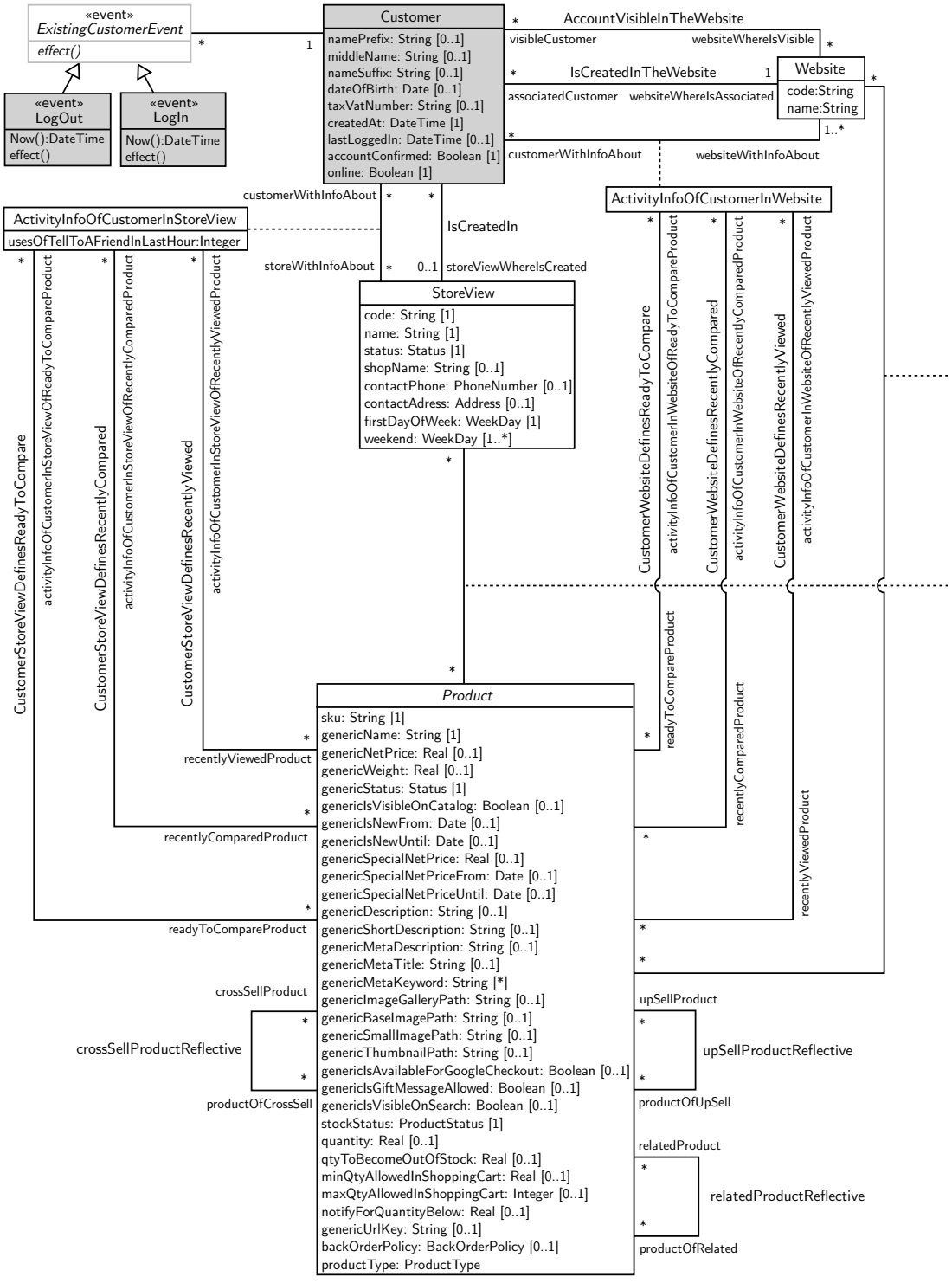
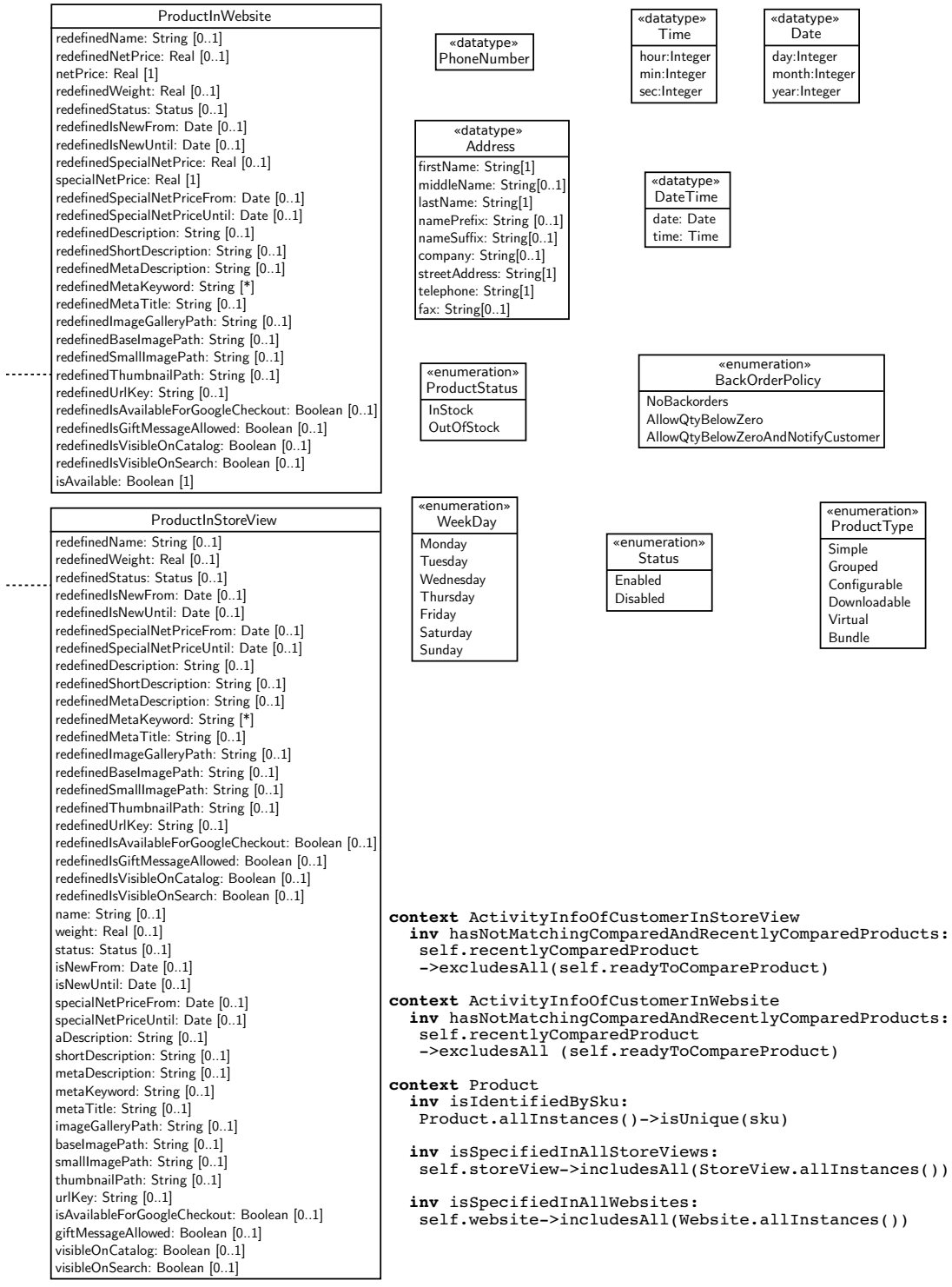


Figure 5.29. Presentation of the filtered conceptual schema for the example of Magento (I).




```
context ActivityInfoOfCustomerInStoreView
  inv hasNotMatchingComparedAndRecentlyComparedProducts:
   self.recentlyComparedProduct
   ->excludesAll(self.readyToCompareProduct)

context ActivityInfoOfCustomerInWebsite
  inv hasNotMatchingComparedAndRecentlyComparedProducts:
   self.recentlyComparedProduct
   ->excludesAll (self.readyToCompareProduct)

context Product
  inv isIdentifiedBySku:
   Product.allInstances()->isUnique(sku)

  inv isSpecifiedInAllStoreViews:
   self.storeView->includesAll(StoreView.allInstances())

  inv isSpecifiedInAllWebsites:
   self.website->includesAll(Website.allInstances())
```

Figure 5.30. Presentation of the filtered conceptual schema for the example of Magento (II).

From the perspective of a conceptual modeler that is interested in the fragment of the Magento concerning LogIn, LogOut, and Customer, the output of our method shows some characteristics about this schema that are worth to mention. There is a large amount of attributes and relationships that extend the knowledge about concepts and interconnect them. By using our method the modeler realizes that there is some kind of relation between the attributes of entity type Product whose name starts with *generic* and the attributes of association classes ProductInWebsite and ProductInStoreView whose name starts with *redefined*.

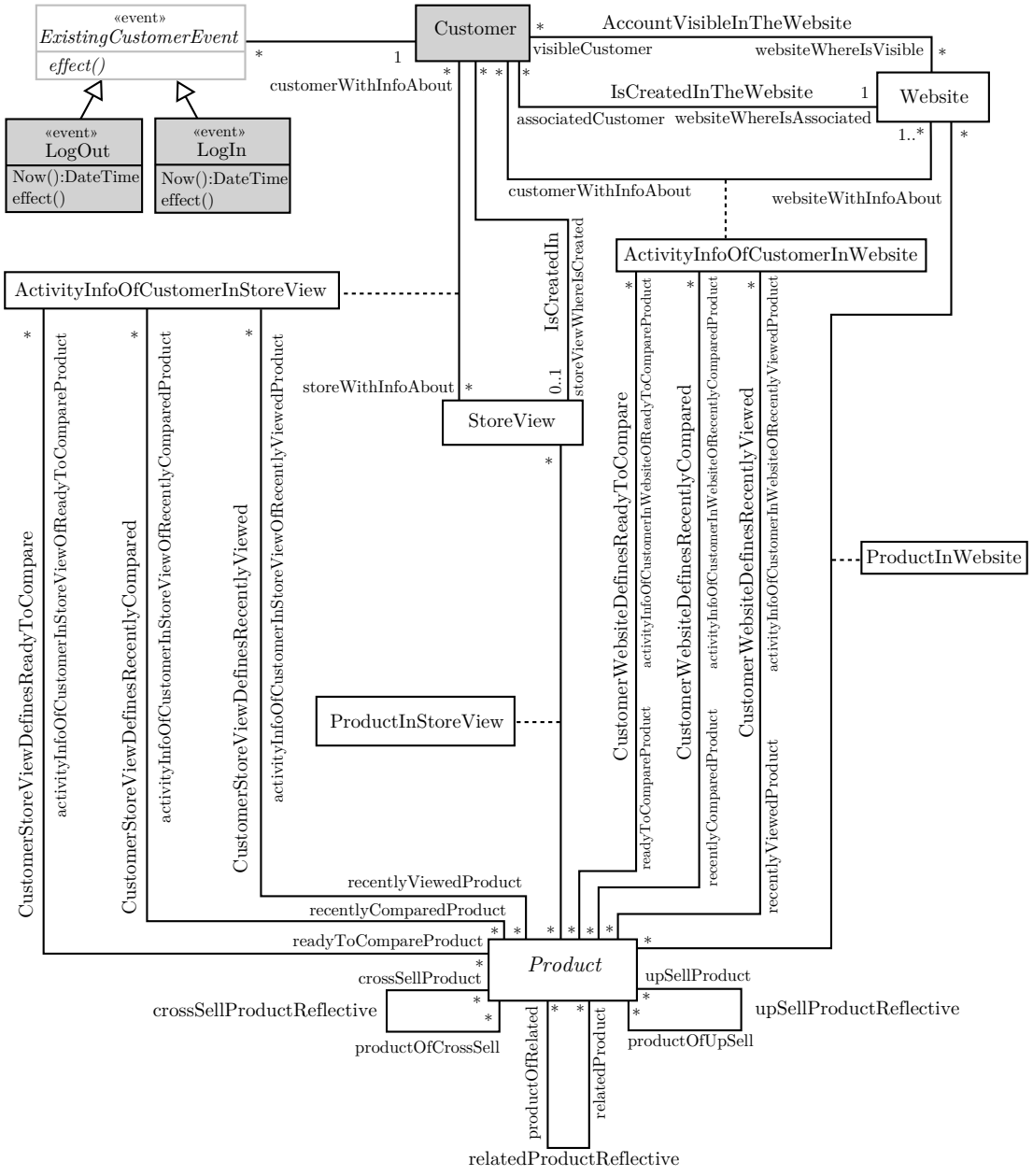


Figure 5.31. Presentation of the filtered conceptual schema for the example of Magento (simplified).

The modeler may check with a domain expert whether this situation is correctly modeled or not, or even suggest a new nomenclature for the attributes of ProductInWebsite and ProductInStoreView to change the *redefined* particle in the names with *specific* in order to avoid any further confusion with the formal UML redefinition of attributes, which uses the tag {*redefines*} as explained in [84].

Nowadays, there are several tools [69, 30] to visualize conceptual schemas that allow users to hide the declaration of attributes inside entity types in order to focus on the general structure at first, and then explore the different details on demand. Figure 5.31 presents a simplified version of the filtered schema depicted in Fig. 5.29 and Fig. 5.30. In such schema the attributes are hidden and the datatypes and enumerations are not shown. The final complexity of the simplified schema is cleary reduced in comparison to the filtered schema with all the details.

5.5 Summary

The chapter has presented a methodology to automatically extract knowledge of interest from a large conceptual schema. For this purpose, we have defined the characteristics of a filtering method. Our proposal is a passive filtering system that determines the relevance of schema elements according to the user preferences. Our filtering method operates at the information source. It maintains the knowledge about the large conceptual schema and filters it whenever the user posts the specific input of a filtering request. We follow a cognitive process based on the relevance of the content of the schema with respect to the user preferences, user goals and needs. The proposed method performs an explicit process of acquiring knowledge on users based on user interrogation. The method expects the intervention of the user to obtain the filtering preferences that conform the input of the filtering process, which contains as core element the focus set of schema elements of interest.

By means of the input described in Sect. 5.2.1, the filtering method produces a filtered conceptual schema as output of its filtering process. The characteristics of such filtered schema depend on the particular user information needs represented in the input of the filtering method. Such input contains a focus set with the schema elements the user wants to focus on, a rejection set that specifies the schema elements the user denotes as not interesting for her knowledge request, a size threshold to limit the final size of the filtered schema, the importance method to compute the relevance of the schema elements, and the large conceptual schema to filter. The filtered conceptual schema presented in Sect. 5.3 contains a subset of the knowledge in the large schema, being a valid instance of the UML metaschema of reduced size with respect to the original schema, and with an interest-driven approach which implies that its contents are of high relevance to the user.

The filtering method is divided into seven ordered stages that sequentially process the input specified by a user in order to obtain a particular output. Section 5.4.1 presents the first stage, which processes the necessary metrics from Ch. 4 which enable the extraction of those elements that are of interest to the user who constructs the input. Section 5.4.2 describes the second stage which deals with the process of selecting the entity and event types that are included in the resulting filtered conceptual schema. Section 5.4.3 explains the characteristics of the third

stage, which processes the relationship types of the filtered schema by selecting and projecting those candidate relationship types from the large schema. Section 5.4.4 shows the fourth stage, which performs the analysis and filtering of generalization relationships. Section 5.4.5 reviews the fifth stage, which extracts the schema rules that must be part of the resulting schema. The data types in the output are processed in sixth stage, as described in Sect. 5.4.6. Finally, the last stage deals with the graphical representation of the filtered conceptual schema in Sect. 5.4.7 and presents the output to the user of the filtering method.

Chapter 6 continues the explanation of the filtering method when applied to specific filtering circumstances. We present a set of particular filtering requests with differences in the input that will produce differences in the filtered conceptual schema that results from their application. The distinct approaches will benefit different information needs of a broad number of users, which may have several expertise degrees when dealing with large conceptual schemas.

But what is it good for?

Engineer at the Advanced Computing Systems Division
of IBM, commenting on the microchip (1968)

6

Catalog of Filtering Requests for Large Conceptual Schemas

A methodology without one or more real implementations that instantiate it to solve the specific problems of a particular situation is worthless. This chapter studies the requirements of users that need to extract knowledge from large conceptual schemas, and the schema elements within them that may be the focus of these users to start specific filtering requests. The main contribution on this part is a catalog of filtering requests as well as the rationale behind their existence. According to the precise information needs of a concrete user, the filtering request to use may be one or another, just like its produced output. Our proposed catalog of filtering requests covers all major knowledge extraction requirements a common user that works with large schemas may need for performing his/her tasks.

The chapter is structured as follows. Section 6.1 summarizes our filtering activity and describes the need for specific filtering requests. In Sect. 6.2 the structure of a general filtering request is introduced. It presents the input, the output, and the different stages of the filtering process that each specific filtering request requires to achieve its purpose. Section 6.3 presents a catalog of six specific filtering requests based on the filtering methodology presented in Ch. 5. Each filtering request contains its application scenario, the characteristics of its inputs and output, and a detailed description of its different phases to filter a large conceptual schema. Finally, Sect. 6.4 describes the relationships between these filtering requests and their intended interaction to satisfy a user's information need over a large conceptual schema.

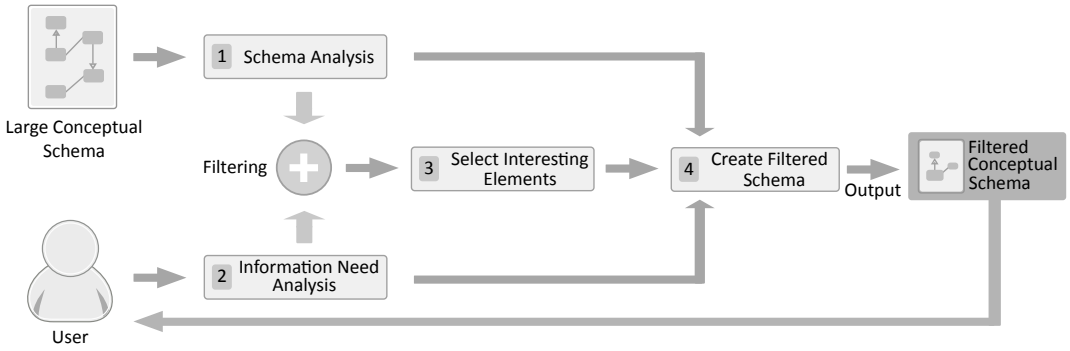


Figure 6.1. Structure of the Filtering Activity.

6.1 Filtering Activity

Large conceptual schemas contain an abundant amount of knowledge about a domain of interest. The main goal of the filtering activity is the management of that information overload to expose users to only information that is relevant to them. Following the filtering methodology of Ch. 5, users of large conceptual schemas obtain a conceptual schema of small size with a structure containing elements that include the knowledge they are interested in.

The activity of filtering large conceptual schemas is represented in Fig. 6.1. On one side, it is necessary to explore the original large schema to know the structure and knowledge contained within it. Usually, such exploration has been done manually by all the users working on the schema. This task is a time-consuming process that sometimes may worsen the impression one gets from the large schema, rather than to improve it. Our methodology presents an automatic approach to explore and analyze large conceptual schemas using relevance metrics.

On the other side, the activity of filtering schemas requires the identification and proper representation of the information need of a user. In general, the requirements of a user evolve as the knowledge she has about the schema increases. Therefore, the analysis of the user's need is a key task in the extraction of the relevant fragments of the schema that match with the user requirements.

Once both the schema and the user information are analyzed, the filtering starts. What is needed is an automatic process to align the knowledge gained from the schema with the representation of the needs of the user, and therefore select the most interesting elements from the large schema correspondingly. That selection has to follow some size limitations according to the user's demand.

The last step of the process binds the selected elements in order to create a valid conceptual schema of reduced size and maximum interest to the user. This task requires the original large schema in order to search for links between elements, and also the analysis of the user to correctly present the resulting filtered conceptual schema. It is important to note that the overall process requires to be executed each time a user makes a specific request. Also, the expected time of each response must be acceptable to allow the process to be dynamic.

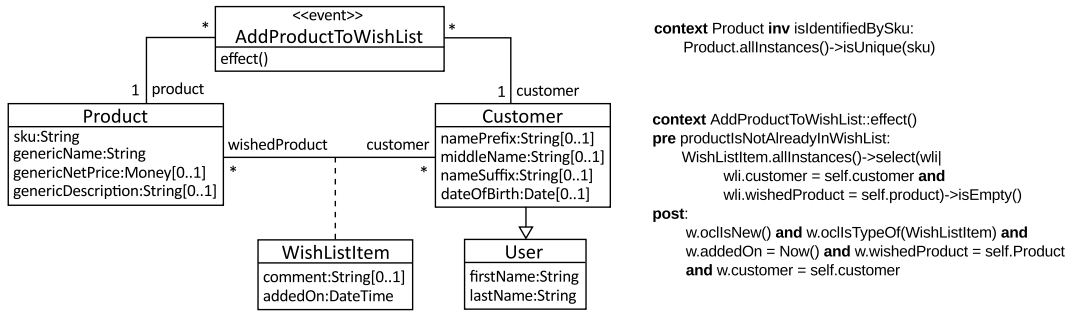


Figure 6.2. Fragment of a conceptual schema of Magento [94] to add products into a wish list.

6.1.1 The Need for Specific Filtering Requests

One of the main problems of using filtering techniques is to deal with and represent the information need of a particular user. Different filtering approaches use different methods to acquire knowledge about the users. This knowledge forms a user representation, which is usually kept in the form of user profiles or rules. According to Hanani et al. in [55], the kind of methods for acquiring knowledge about users include an *explicit approach*, which is based on user interrogation, an *implicit approach*, which infers the user representation automatically by recording user behavior, or a mixed approach.

User interrogation is the most popular technique of the *explicit approach*. Systems utilizing this method usually require their users to fill out a form describing their areas of interest or other relevant parameters. On the other hand, recording user behavior is an *implicit approach* that does not require active user involvement in the knowledge acquisition task. Instead, the user's reaction to each incoming interaction is recorded, in order to learn from it about the actual relevancy of elements to the user. Finally, the element space approach lies between the *explicit* and the *implicit* approaches, as it requires minimum user involvement. This method creates a field of elements that the user has previously judged as relevant. Any new element is tested for its similarity to the elements existing in that space. If similarity of the new element is above a certain relevance threshold, it is considered relevant.

Our approach follows the guidelines of the filtering methodology presented in Ch. 5. It is an *explicit approach* where the user selects the elements in the large schema that she is interested in and wants to obtain more information about. That selection is called focus set and may contain several kinds of elements inside. Our filtering approach automatically selects the most interesting fragment of the schema according to the user selection and creates a small and well-formed schema with it, as aforementioned.

Since different kinds of elements must be part of the focus set, it is necessary to have specific filtering requests to provide different filtering behaviors. As an example, Fig. 6.2 presents a small fragment of a conceptual schema of the Magento [94] e-commerce system. The fragment describes the structure and behavior that provides the functionality of adding products into a wish list to customers of an on-line store. It contains the entity types `Product`, `Customer`, and

`User`, which is the superclass of `Customer`. The `isIdentifiedBySku` invariant indicates that each product is identified by its stock-keeping unit code. There is a relationship type between `Product` and `Customer` with the `WishListItem` association class to represent the items within the wish list of a customer. Finally, the event type `AddProductToWishList` represents the creation event of a particular instance of `WishListItem` between a pair of specific product and customer, as indicated in the precondition and postcondition of the `effect()` operation of `AddProductToWishList`. All of these elements may be the focus of a specific filtering request of a user that wants to obtain more knowledge from the schema about any of them. We summarize those elements as follows:

Entity Types They are one of the most important elements in conceptual schemas. Entity types play a fundamental role defining the concepts that are relevant to a particular information system. A user that wants to know a conceptual schema, needs to know the entity types that are defined within it. As an example, a user that does not have experience with an e-commerce system like Magento (Fig. 6.2) may select a focus set containing the entity types `Product` and `Customer` in an initial filtering request. As a result, the user will obtain a filtered conceptual schema with attributes, relationship types, generalizations, event types and schema rules of interest to him and with a higher relation to `Product` and `Customer`. Consequently, the user is able to manage such reduced amount of information and obtains knowledge about the semantics of the schema.

Relationship Types They represent how two or more entity or event types are related to one another. To focus on relationship types is specially useful when the participants in a relationship are subtypes or supertypes of other entity or event types. In that case, the relationship type in the filtered schema may be the original, a redefinition, or a projection of it, according to the other elements of the focus set. As a result, a user may discover other aspects of a relationship while she incrementally obtains knowledge from the schema.

Event Types They represent the changes in the information base of a conceptual schema that are permissible. Are similar to entity types but in the behavioral part. A user that wants to know the participants in an event type may select the event as the focus set of a specific filtering request. In the example of Magento (Fig. 6.2) a focus set containing `AddProductToWishList` will produce a filtered schema with such event as main element.

Schema Rules They represent the constraints, invariants, derivation rules, and pre- and post-conditions that affect the schema, like the ones in Fig. 6.2. A user may focus on a schema rule in order to obtain a filtered schema with the schema elements that are affected by such rule.

Fragment of a Conceptual Schema An entire fragment of a large schema as the one in Fig. 6.2 may also be a selected focus set for a user that wants to surround such fragment with additional knowledge.

We believe that the filtering requests that are described in Sec. 6.3 are sufficient for most of the users that need to explore the knowledge represented in a large conceptual schema.

6.2 General Structure of a Filtering Request

In order to fulfill the present requirements of the filtering activity, we need a filtering methodology, as the one presented in Ch. 5. However, our filtering methodology describes the general characteristics of a regular and systematic way of accomplishing the user-driven process to extract knowledge from a large conceptual schema. What is needed is to define a set of concrete filtering requests that inherit the guidelines and main organization proposed by the previous methodology but adapting it to the particular filtering needs a user may have when working with a large schema. A filtering request for a large conceptual schema is a specific knowledge extraction request that automatically obtains a portion of the entire knowledge of small size and high relevance to the user in relation to the schema elements in the user focus of interest. The specific output depends on the input and the specific filtering request.

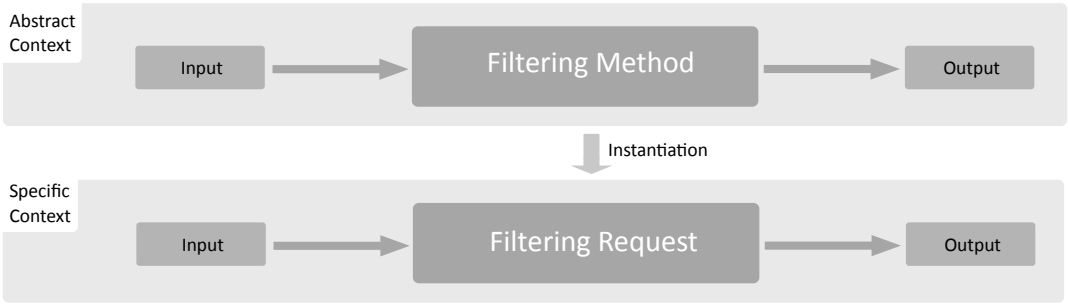


Figure 6.3. Relationship between the filtering methodology and the specific filtering requests.

As depicted in Fig. 6.3, we consider each filtering request as a concrete instantiation of our filtering method to be effectively used under certain filtering circumstances. A user that wants to extract knowledge from a large conceptual schema may focus on an entity type and then obtain the most interesting entity types with relation to that one. Or maybe the user wants to obtain the event types where the entity type of focus participates. Or its schema rules containing integrity constraints and derivation rules. Or even the user prefers to focus on a relationship type of interest. Or, instead of a relationship, the user wants to focus on a small subset of the large schema. It is clear that the filtering activity has many faces, and to effectively provide help on all of these situations is not an easy task. Accordingly, we define the filtering method as a general template or abstract representation from which the concrete filtering requests to cover all the particular filtering situations are derived.

The filtering method and the specific filtering requests that conform our proposed catalog are processes that require an input to produce their output. The structure of a specific filtering request follows the same template defined by the filtering method in Ch. 5. There are common characteristics shared by the input and output of all the specific filtering requests that are worth to mention in relation to the general input and output of the filtering method. The following subsections review the elements that conform the input, the output, and the different stages that are part of the internal composition of the filtering requests.

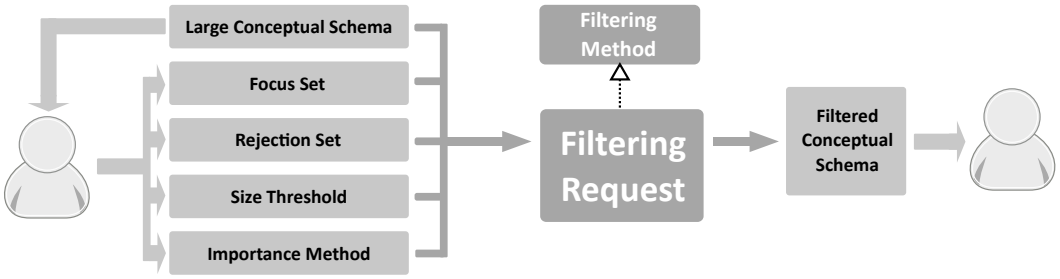


Figure 6.4. The input and output of a filtering request.

6.2.1 Specific Input of a Filtering Request

The input for a filtering request contains the same elements as in the filtering method of Ch. 5. Concretely, the common characteristics of a specific input are:

- **Large conceptual schema:** the source schema $\mathcal{CS} = \langle \mathcal{SS}, \mathcal{BS} \rangle$ where $\mathcal{SS} = \langle \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ is the structural subschema, and $\mathcal{BS} = \langle \mathcal{E}_b, \mathcal{R}_b, \mathcal{G}_b, \mathcal{C}_b \rangle$ is the behavioral subschema, as described in Ch. 2.
- **Focus Set:** the conceptual schema viewpoint of the user. Formally, the focus set $\mathcal{FS}$ contains a small subset of schema elements from the large schema, from which the user wants to know more about.
- **Size threshold:** the maximum expected number of schema elements in the output. It ensures that the size of the output does not exceed the requirements of the user.
- **Rejection Set:** the set with schema elements from the large schema that the user does not want to obtain in the output. Note that it is disjoint with the focus set.
- **Importance method:** the algorithm to compute the importance of schema elements as described in Ch. 4.

Figure 6.4 depicts the previous elements as the input of a filtering request. It is important to note that these elements conform the minimum input for a filtering request. It is possible to add new elements in order to describe new filtering behaviors, as in the case of the filtering request for contextualized event types in the catalog of Sect. 6.3.1.

6.2.2 Specific Output of a Filtering Request

The output for a filtering request is a filtered conceptual schema $\mathcal{CS}_\mathcal{F} = \langle \mathcal{SS}_\mathcal{F}, \mathcal{BS}_\mathcal{F} \rangle$, where $\mathcal{SS}_\mathcal{F} = \langle \mathcal{E}_\mathcal{F}, \mathcal{R}_\mathcal{F}, \mathcal{T}_\mathcal{F}, \mathcal{G}_\mathcal{F}, \mathcal{C}_\mathcal{F}, \mathcal{D}_\mathcal{F} \rangle$ is the structural subschema, and $\mathcal{BS}_\mathcal{F} = \langle \mathcal{E}_{b\mathcal{F}}, \mathcal{R}_{b\mathcal{F}}, \mathcal{G}_{b\mathcal{F}}, \mathcal{C}_{b\mathcal{F}} \rangle$ is the behavioral subschema, as described in Sect. 5.3 of Ch. 2.

The filtered conceptual schema fulfills the requirements of the user as indicated in the input of the filtering request and contains a fragment of knowledge of high interest and small size from the large conceptual schema. Figure 6.4 depicts the filtered conceptual schema as the output of a filtering request.

6.2.3 The 7 Stages of a Filtering Request

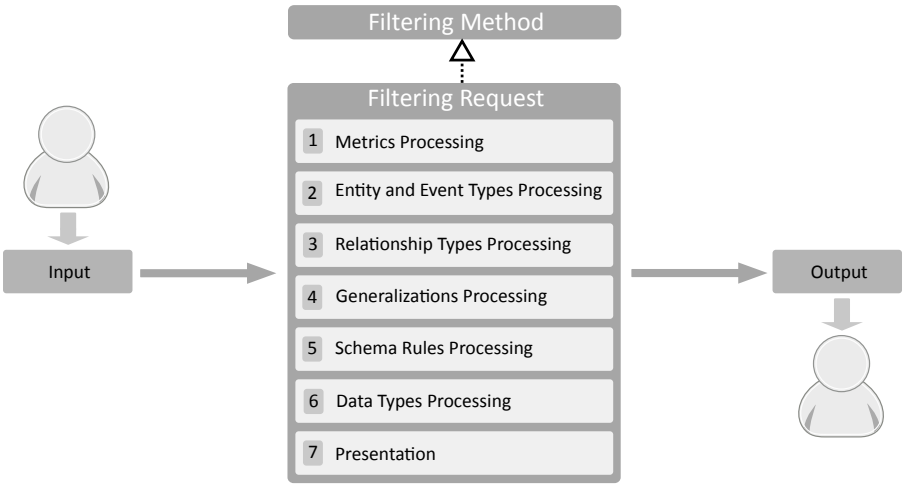


Figure 6.5. The 7 stages of a filtering request.

A specific filtering request is divided into seven ordered stages that sequentially process the input specified by a user in order to obtain a particular output. Figure 6.5 presents the different stages of a filtering request.

1. **Metrics Processing** The first stage applies the metrics of Ch. 4 to the elements of the original large schema out of the focus set in order to discover which are the most relevant ones for the user. This stage follows the same steps as presented in Sect. 5.4.1 of Ch. 5.
2. **Entity and Event Types Processing** The second stage selects the entity and event types from the large schema that will appear in the resulting filtered schema. This stage follows the same steps as presented in Sect. 5.4.2 of Ch. 5.
3. **Relationship Types Processing** This stage selects the relationship types from the large schema that will appear in the resulting filtered schema. This stage follows the same steps as presented in Sect. 5.4.3 of Ch. 5.
4. **Generalizations Processing** This stage selects the generalization relationships that will appear in the resulting filtered schema. This stage follows the same steps as presented in Sect. 5.4.4 of Ch. 5.
5. **Schema Rules Processing** This stage processes the schema rules of the original schema that will appear in the resulting filtered schema. This stage follows the same steps as presented in Sect. 5.4.5 of Ch. 5.
6. **Data Types Processing** This stage selects the data types that belong to the resulting filtered schema. This stage follows the same steps as presented in Sect. 5.4.6 of Ch. 5.
7. **Presentation** The last stage deals with the presentation of the filtered schema to the user. This stage follows the same steps as presented in Sect. 5.4.7 of Ch. 5.

6.3 Catalog of Filtering Requests

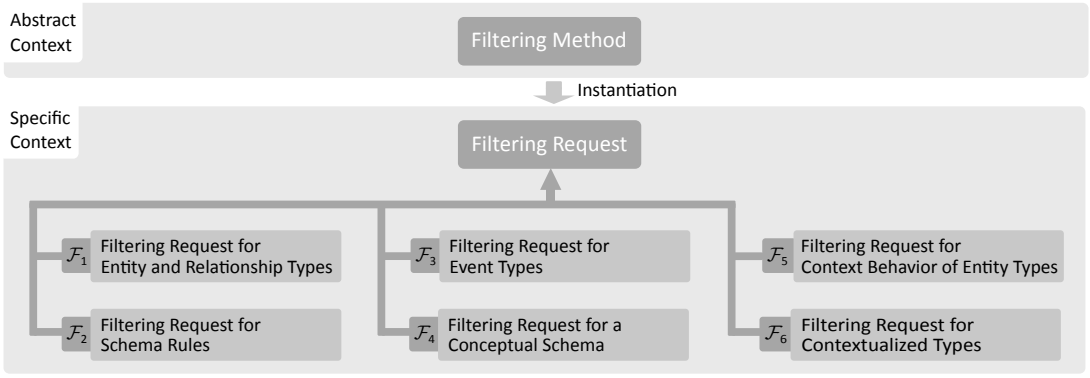


Figure 6.6. Catalog of Filtering Requests.

This section presents a catalog of filtering requests applicable to large conceptual schemas specified in UML/OCL. This catalog presents six filtering requests and describes the characteristics of each of them in comparison to the general stages presented along Sec. 5.4 of Ch. 5. Figure 6.6 depicts the taxonomy of the catalog of filtering requests.

- **Filtering Request** $\mathcal{F}_1$: The user focuses on a set of entity and relationship types from a large schema. The request obtains a filtered conceptual schema that includes the combination of the initial entity and relationship types with the elements of interest.
- **Filtering Request** $\mathcal{F}_2$: The user focuses on a set of schema rules from a large schema. As output, the user obtains a filtered conceptual schema that includes the combination of the elements referenced by the selected schema rules with the elements of interest.
- **Filtering Request** $\mathcal{F}_3$: The user focuses on a set of event types from a large schema. The request produces a reduced conceptual schema that includes the combination of the selected event types with the elements of interest.
- **Filtering Request** $\mathcal{F}_4$: The user focuses on a small fragment from the large schema. The user is aware of the elements that conform such fragment or she has accessed them via previous requests. As output, the user obtains a filtered schema that includes the combination of the elements of the fragment surrounded with elements of interest.
- **Filtering Request** $\mathcal{F}_5$: The user focuses on a set of entity types from a large schema. The request obtains those event types of interest that reference the selected entity types. The resulting schema includes the combination of those entity and event types.
- **Filtering Request** $\mathcal{F}_6$: The user focuses on a set of entity and event types. The user contextualize them by means of a function to reduce or limit the characteristics defined over such types. As output, the user obtains a filtered schema with the selected entity and event types and the elements of interest taking into account the contextualization.

Next subsections present a detailed description of the characteristics of each filtering request.

6.3.1 $\mathcal{F}_1$: Filtering Request for Entity and Relationship Types

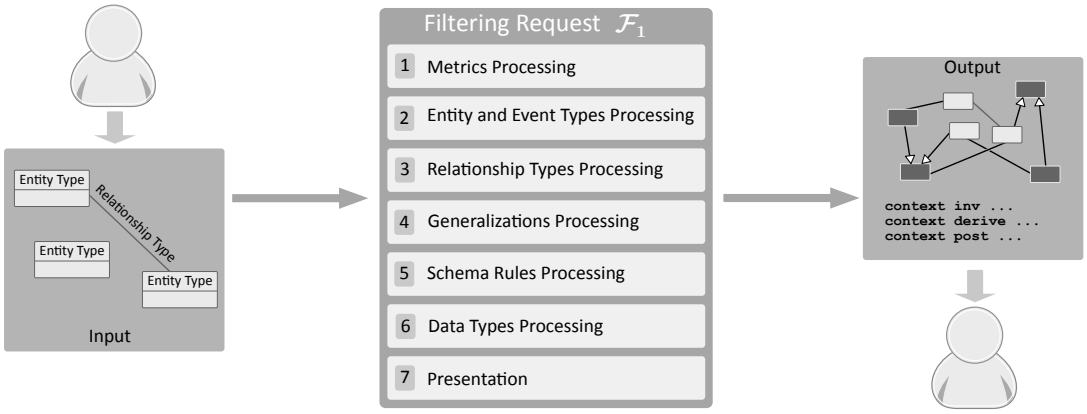


Figure 6.7. $\mathcal{F}_1$: Filtering request for entity and relationship types.

The first filtering request deals with a filtering interaction centered on the entity and relationship types from a large schema. In the following we describe this filtering request according to its application scenario, the specific inputs and outputs, the particularities within its filtering stages, and the correctness of the overall process.

Application Scenario

$$\mathcal{F}_1\colon \mathcal{P}_{\geq 1}(\mathcal{E} \cup \mathcal{R}) \stackrel{1}{\rightarrow} \mathcal{CS}_{\mathcal{F}}$$

The user focuses on a set of entity and relationship types from a large conceptual schema. The user is aware of those types or she has accessed them via previous filtering requests. The information need consists in obtaining more knowledge from the schema with relation to the entity and relationship types in the user focus. The method obtains the elements of interest to the user according to the initial selection and the characteristics represented in the large schema. As output, the user obtains a small-size filtered conceptual schema that includes the combination of the initial elements of focus with the elements of interest gathered by our methodology.

Specific Input

The input for the filtering request $\mathcal{F}_1$ contains the same elements as indicated in Sect. 6.2. In addition, we present here a detailed description of the particularities of such elements for the specific input of $\mathcal{F}_1$.

[1] $\mathcal{P}(\mathcal{S})$ denotes the power set of a set $\mathcal{S}$. If $\mathcal{S} = \{a, b\}$ Then $\mathcal{P}(\mathcal{S}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$ and $\mathcal{P}_{\geq 1}(\mathcal{S}) = \{\{a\}, \{b\}, \{a, b\}\}$.

- **Large conceptual schema:** the source schema $\mathcal{CS} = \langle \mathcal{SS}, \mathcal{BS} \rangle$ where $\mathcal{SS} = \langle \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ is the structural subschema, and $\mathcal{BS} = \langle \mathcal{E}_b, \mathcal{R}_b, \mathcal{G}_b, \mathcal{C}_b \rangle$ is the behavioral subschema. The amount of knowledge represented in $\mathcal{CS}$ is large and makes it very difficult to manually extract fragments of interest to a user.

- **Focus Set:** it works as the conceptual schema viewpoint of the user. Therefore, a focus set is an initial point that should be extended with more knowledge. Formally, the focus set $\mathcal{FS}$ contains a small subset of the entity types of $\mathcal{E}$ and the relationship types of $\mathcal{R}$, from which the user wants to know more about. Note that the size of the focus set is reduced with respect to the amount of entity and relationship types from the large schema ($|\mathcal{FS}| \ll |\mathcal{E} \cup \mathcal{R}|$). Also, it is mandatory for the user to select a non-empty focus set ($\mathcal{FS} \neq \emptyset$).

- **Size threshold:** it denotes the maximum expected number $\mathcal{K}$ of entity and event types in the output. Note that $|\mathcal{E}_{\mathcal{FS}}| \leq \mathcal{K} \leq |\mathcal{E} \cup \mathcal{E}_b|$, where $\mathcal{E}_{\mathcal{FS}}$ is the set that includes the entity types in the focus set $\mathcal{FS}$. Note that $\mathcal{E}_{\mathcal{FS}}$ also contains those entity types that participate in relationship types within the focus set.

- **Rejection Set:** the set $\mathcal{RS}$ with entity types of $\mathcal{E}$ and event types of $\mathcal{E}_b$ that the user does not want to obtain in the output. Note that it is disjoint with the focus set ($\mathcal{RS} \cap \mathcal{FS} = \emptyset$). By default, the rejection set is empty ($|\mathcal{RS}| \geq 0$).

- **Importance method:** the algorithm $\mathcal{I}$ to compute the importance of entity types from $\mathcal{E}$ and event types from $\mathcal{E}_b$. By default $\mathcal{I} = I_{SM}$, the Simple Method described in Chap. 4.

Specific Output

The output of this filtering request is a filtered conceptual schema $\mathcal{CS}_{\mathcal{F}} = \langle \mathcal{SS}_{\mathcal{F}}, \mathcal{BS}_{\mathcal{F}} \rangle$, where $\mathcal{SS}_{\mathcal{F}} = \langle \mathcal{E}_{\mathcal{F}}, \mathcal{R}_{\mathcal{F}}, \mathcal{T}_{\mathcal{F}}, \mathcal{G}_{\mathcal{F}}, \mathcal{C}_{\mathcal{F}}, \mathcal{D}_{\mathcal{F}} \rangle$ is the structural subschema, and $\mathcal{BS}_{\mathcal{F}} = \langle \mathcal{E}_{b\mathcal{F}}, \mathcal{R}_{b\mathcal{F}}, \mathcal{G}_{b\mathcal{F}}, \mathcal{C}_{b\mathcal{F}} \rangle$ is the behavioral subschema. The specific constraints that $\mathcal{CS}_{\mathcal{F}}$ must satisfy are described as follows:

[C1] $\mathcal{E}_{\mathcal{F}}$ contains the entity types $\mathcal{E}_{\mathcal{FS}}$ from the focus set $\mathcal{FS}$.

[C2] $\mathcal{E}_{\mathcal{F}}$ does not contain the entity types from the rejection set $\mathcal{RS}$.

[C3] $\mathcal{R}_{\mathcal{F}}$ contains the relationship types $\mathcal{R}_{\mathcal{FS}}$ from the focus set $\mathcal{FS}$.

[C4] $\mathcal{R}_{\mathcal{F}}$ does not contain the relationship types from the rejection set $\mathcal{RS}$.

[C5] $\mathcal{G}_{\mathcal{F}}$ contains direct generalization relationships between entity types of $\mathcal{E}_{\mathcal{F}}$.

[C6] $\mathcal{G}_{b\mathcal{F}}$ contains direct generalization relationships between event types of $\mathcal{E}_{b\mathcal{F}}$.

[C7] If c is an integrity constraint or derivation rule of $\mathcal{C}$, $\mathcal{D}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_{\mathcal{F}}$ and any of the schema elements referenced by c belong to $\mathcal{CS}_{\mathcal{F}}$, then c is included in $\mathcal{C}_{\mathcal{F}}$, $\mathcal{D}_{\mathcal{F}}$ or $\mathcal{C}_{b\mathcal{F}}$.

[**C8**] If r is a relationship type of $\mathcal{R}$ and its participant entity types belong to $\mathcal{E}_\mathcal{F}$, or are ascendants of entity types of $\mathcal{E}_\mathcal{F}$ (in which case a projection is needed), then r is included in $\mathcal{R}_\mathcal{F}$. The same behavior applies to relationship types of $\mathcal{R}_b$ to be included in $\mathcal{R}_{b\mathcal{F}}$.

[**C9**] If d is a data type of $\mathcal{T}$ and it is used by attributes of entity types of $\mathcal{E}_\mathcal{F}$, event types of $\mathcal{E}_{b\mathcal{F}}$, or schema rules of $\mathcal{C}_\mathcal{F}$, $\mathcal{C}_{b\mathcal{F}}$ or $\mathcal{D}_\mathcal{F}$, then d is included in $\mathcal{T}_\mathcal{F}$ of $\mathcal{CS}_\mathcal{F}$.

[**C10**] If e_1 and e_2 are entity types of $\mathcal{E}_\mathcal{F}$ and does not exist a direct generalization between them in $\mathcal{G}_\mathcal{F}$ nor a path of direct generalizations of $\mathcal{G}_\mathcal{F}$ traversing entity types e_i of $\mathcal{E}_\mathcal{F}$, but both e_1 and e_2 belong to different levels of the same hierarchy in $\mathcal{G}$ of $\mathcal{CS}$, a direct generalization g' is included between e_1 and e_2 in $\mathcal{G}_\mathcal{F}$ but is marked as derived. The same behavior applies to pairs of event types of $\mathcal{E}_{b\mathcal{F}}$.

[**C11**] If c is a constraint of $\mathcal{C}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_\mathcal{F}$ and references schema elements out of $\mathcal{CS}_\mathcal{F}$ only the header of such constraint is included in $\mathcal{C}_\mathcal{F}$ or $\mathcal{C}_{b\mathcal{F}}$. If d is a derivation rule of $\mathcal{D}$ whose context belongs to $\mathcal{CS}_\mathcal{F}$ and references schema elements out of $\mathcal{CS}_\mathcal{F}$, the context element of the rule is marked as materialized in $\mathcal{CS}_\mathcal{F}$ and d is not included in $\mathcal{D}_\mathcal{F}$.

Filtering Stages

The filtering request for entity and relationship types follows the specific methodology presented in Sect. 5.4 of Ch. 5. In the following, we present a brief summary of the different stages and steps (see Fig. 6.8) that belong to this filtering request.

- **Stage 1: Metrics.** The first step in this stage (see Activity 1.1 in Fig. 6.8) processes the input of the filtering request. It creates some auxiliary sets to gather the entity, relationship, and event types from the focus set. In this particular case, the auxiliary set $\mathcal{E}_{\mathcal{FS}}$ contains the union between the entity types from the focus set, and the entity types that are participants of the relationship types selected in the focus set. Also, the set $\mathcal{E}_{b\mathcal{FS}}$ that contains event types from the focus set is empty since this filtering request focus on the structural part to construct the input. The rest of the steps follow the same indications presented in Sect. 5.4.1 of Ch. 5. The method computes the importance, closeness, and interest metrics, and sorts the entity and event types in a ranking.
- **Stage 2: Entity and Event Types.** This stage follows the same steps as presented in Sect. 5.4.2 of Ch. 5. The method selects the top entity and event types from the interest ranking taking into account the size threshold of the input and includes them in the resulting filtered conceptual schema.
- **Stage 3: Relationship Types.** This stage follows the same steps as presented in Sect. 5.4.3 of Ch. 5. The method classifies the relationship types according to their participants as referentially-complete or referentially-partial relationship types. Note that those relationship types in the focus set are always referentially-complete since their participants are members on the set of entity types $\mathcal{E}_{\mathcal{FS}}$ from the focus set. Then, the method projects and selects the final relationship types that are part of the resulting filtered conceptual schema.

- **Stage 4: Generalizations.** This stage follows the same steps as presented in Sect. 5.4.4 of Ch. 5. The method processes the direct and creates indirect generalizations to construct the hierarchies of the resulting filtered conceptual schema.
- **Stage 5: Schema Rules.** This stage follows the same steps as presented in Sect. 5.4.5 of Ch. 5. The method selects the schema rules defined in the context of elements from the filtered schema and processes the referentially-incomplete ones in order to construct the rules of the resulting filtered conceptual schema.
- **Stage 6: Data Types.** This stage follows the same steps as presented in Sect. 5.4.6 of Ch. 5. The method includes in the filtered schema those data types referenced or used by other elements within such schema.
- **Stage 7: Presentation.** This stage follows the same steps as presented in Sect. 5.4.7 of Ch. 5. The method presents the filtered schema to the user. The elements to highlight are the entity types and relationship types from the focus set $\mathcal{FS}$.

Method Correctness

The proposed activities in the stages of the filtering request transform the input into a valid output in the form of a filtered conceptual schema that satisfies a set constraints over such schema. In the following we verify the correctness of the method according to those constraints and the activities that satisfy each of them:

- Constraint [**C1**] is satisfied by the activity 2.2 of the filtering request.
- Constraint [**C2**] is satisfied by the activity 2.2 of the filtering request.
- Constraint [**C3**] is satisfied by the activities 3.1, 3.2, and 3.4 of the filtering request.
- Constraint [**C4**] is satisfied by the activity 3.1 of the filtering request.
- Constraint [**C5**] is satisfied by the activity 4.1 of the filtering request.
- Constraint [**C6**] is satisfied by the activity 4.2 of the filtering request.
- Constraint [**C7**] is satisfied by the activities 5.1 and 5.2 of the filtering request.
- Constraint [**C8**] is satisfied by the activities 3.1, 3.3, and 3.4 of the filtering request.
- Constraint [**C9**] is satisfied by the activity 6.1 of filtering request method.
- Constraint [**C10**] is satisfied by the activities 4.3 and 4.4 of the filtering request.
- Constraint [**C11**] is satisfied by the activities 5.1, 5.3, and 5.4 of the filtering request.

The activities 1.1, 1.2, 1.3, 1.5, and 1.5 of the filtering request process the input and deal with the computation of relevance metrics to filter the large schema. The activity 2.1 selects the additional entity and event types to complete the knowledge from the focus set, and the activity 2.3 includes the event types from 2.1 into the filtered schema. The activity 7.1 presents the resulting filtered conceptual schema to the user.

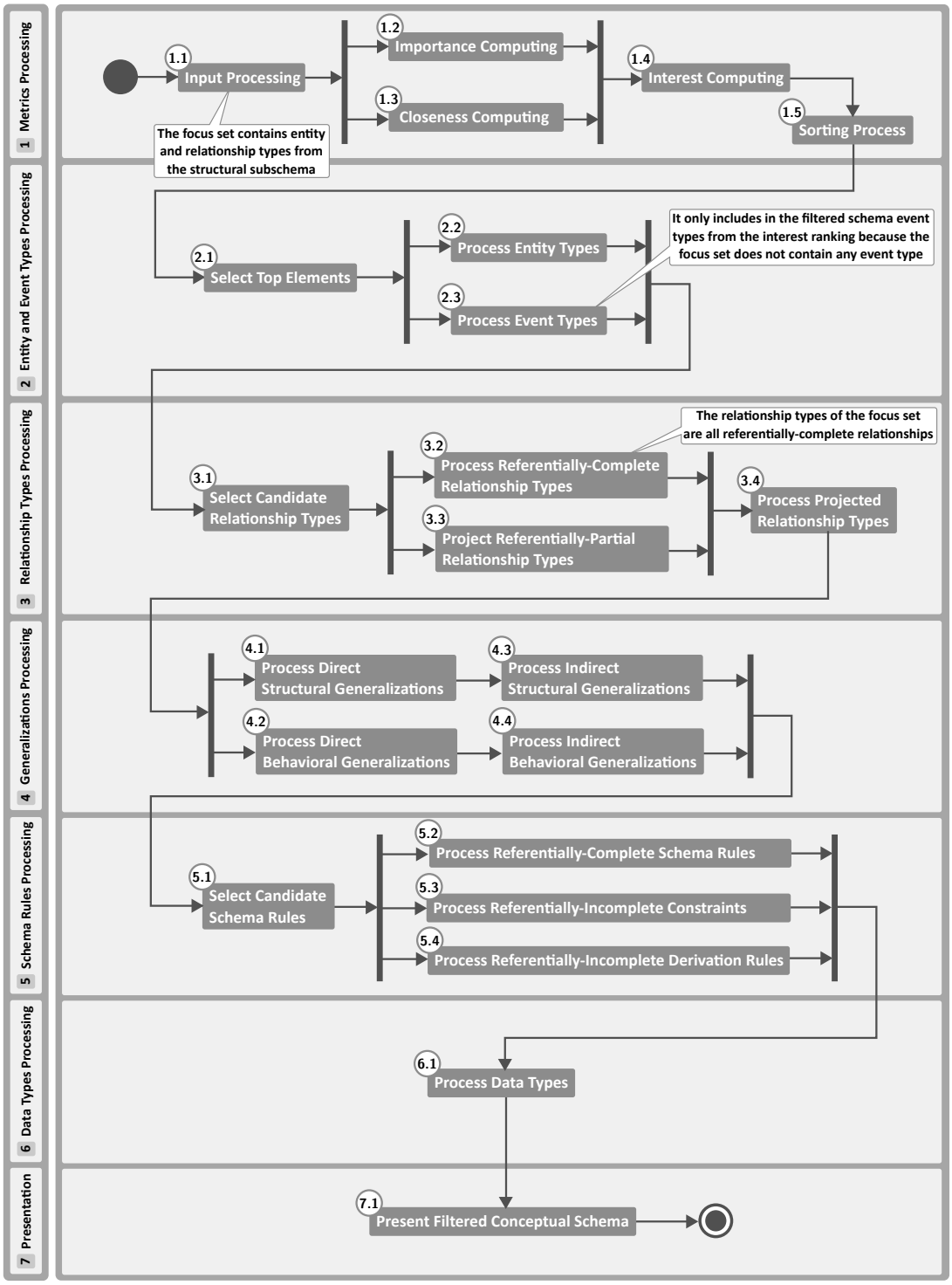


Figure 6.8. Activity diagram for the filtering request $\mathcal{F}_1$.

Example of $\mathcal{F}_1$

The filtering scenario proposed in this example describes a user of the conceptual schema of the Magento e-commerce system that wants to know the schema elements of interest with respect to a set of entity and relationship types. Concretely, the user focuses in the entity type PriceAttribute, and the relationship type AttributeInStoreView. The input of the filtering request is as follows:

```
CS = Magento
FS = {PriceAttribute, AttributeInStoreView}
RS = ∅
K = 5
I  = CEntityRank Extended
```

Figure 6.9 and Fig. 6.10 depict the resulting filtered conceptual schema our method produces for the previous request. Note that the first important feedback the user obtains is the definition of the relationship type of focus AttributeInStoreView as an association class between PriceAttribute and StoreView. Such relationship type was originally specified in the context of the entity type Attribute and has been projected to PriceAttribute, which is a descendant of Attribute. The projection of this relationship is transparent for the user who focus on PriceAttribute since our filtering method maintains the semantics of the original schema. The rest of relationship types from the filtered schema in which PriceAttribute participates are also projected relationships from the context of Attribute.

From the filtered schema of Fig. 6.9, the user discovers that products are related to attributes that can be rated (globally or not) in a store view or a website inside the Magento system. There may be an instance of the association class StoreViewAttributeRating for each tuple of PriceAttribute, StoreView, and Product. In the same way, there may be an instance of the association class WebsiteAttributeRating for each tuple of PriceAttribute, Website, and Product. Also, there is a lot of information about products in the context of a website or a store view where the product is located. Such information is defined as a set of attributes of the association classes ProductInWebsite and ProductInStoreView. These attributes redefine specific properties about the general definition of product where it is presented in a store view or website.

The fragment of the filtered schema depicted in Fig. 6.10 contains the graphical representation of the data types and enumeration types that are necessary to understand the semantics of the result. The attributes contactPhone, contactAddress, and firstDayOfWeek of the entity type StoreView are of type PhoneNumber, Address, and WeekDay, respectively. Also, the attributes genericStatus, stockStatus, backOrderPolicy, and productType are of type Status, ProductStatus, BackOrderPolicy, and ProductType, respectively. Furthermore, we include in the filtered schema those schema rules that are referentially complete. Concretely, we show in Fig. 6.10 the OCL specification to indicate that products are identified by their SKU (stock-keeping unit) value, and that products must be specified in all the websites and store views of the system. Additionally, there are two more schema rules to indicate that store views are identified by their code and name, and that websites are identified by their code.

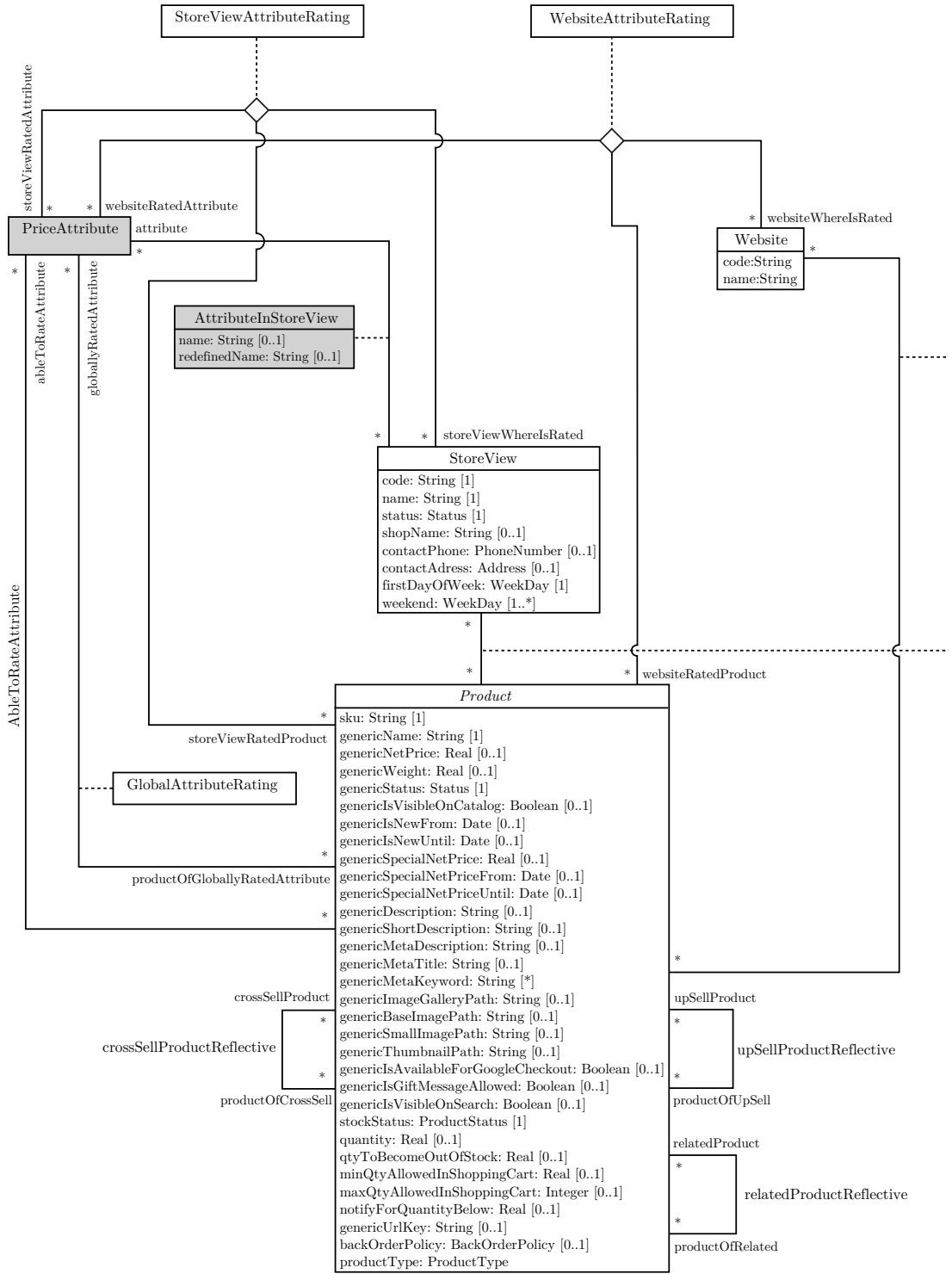


Figure 6.9. Filtered schema for the entity PriceAttribute and association AttributeInStoreView (I).

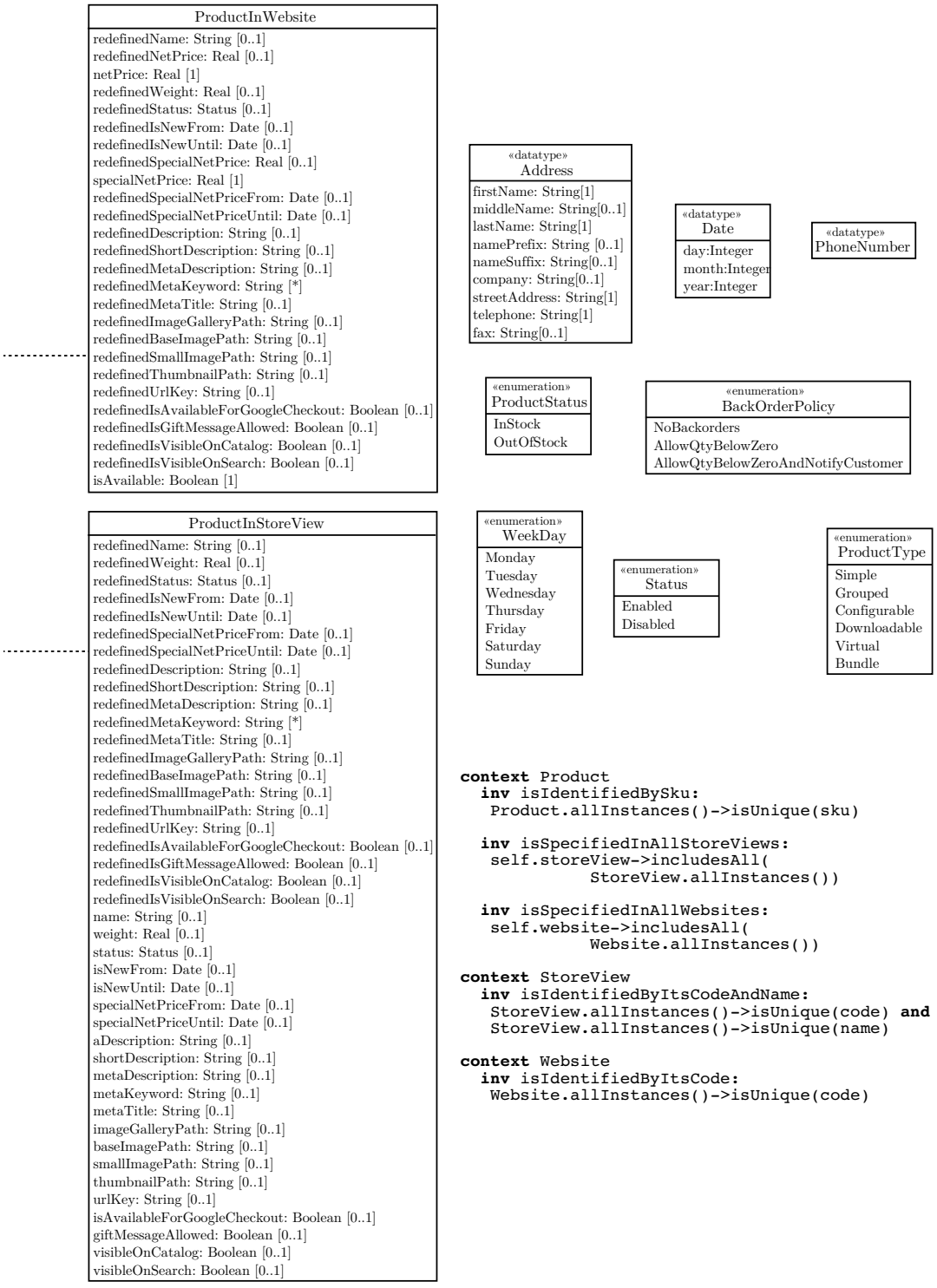


Figure 6.10. Filtered schema for the entity PriceAttribute and association AttributeInStoreView (II).

6.3.2 $\mathcal{F}_2$: Filtering Request for Schema Rules

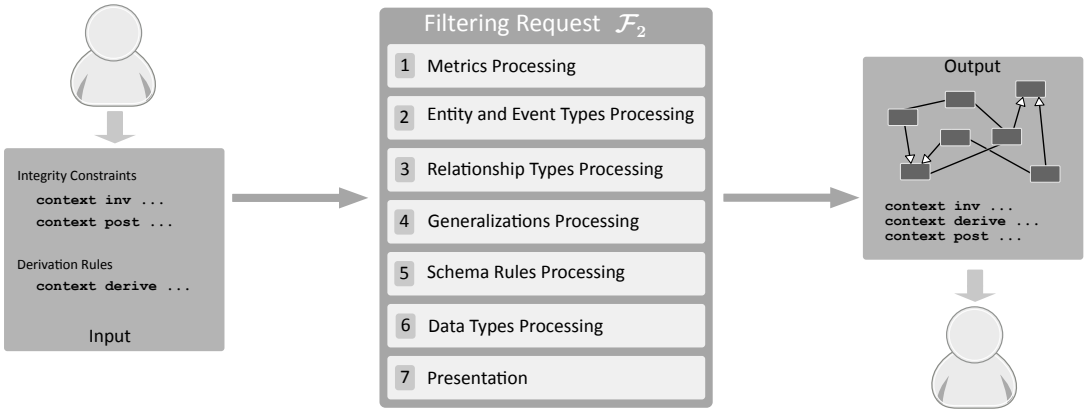


Figure 6.11. $\mathcal{F}_2$: Filtering request for schema rules.

Application Scenario

$$\mathcal{F}_2\colon \mathcal{P}_{\geq 1}(\mathcal{C} \cup \mathcal{D} \cup \mathcal{C}_b) \rightarrow \mathcal{CS}_{\mathcal{F}}$$

The user focuses on a set of integrity constraints and derivation rules from a large conceptual schema. The user is aware of those schema rules or she has accessed them via previous filtering requests. The information need consists in obtaining knowledge from the schema with relation to the schema rules in the user focus. The method obtains the elements of interest to the user according to the initial selection and the characteristics represented in the large schema. As output, the user obtains a small-sized filtered conceptual schema that includes the combination of the initial schema rules of focus with the elements of interest gathered by our methodology.

Specific Input

The input for the filtering request $\mathcal{F}_2$ contains the same elements as indicated in Sect. 6.2 with an extra element to indicate the scope of the results. In addition, we present here a detailed description of the particularities of such elements for the specific input of $\mathcal{F}_2$.

- **Large conceptual schema:** the source schema $\mathcal{CS} = \langle \mathcal{SS}, \mathcal{BS} \rangle$ where $\mathcal{SS} = \langle \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ is the structural subschema, and $\mathcal{BS} = \langle \mathcal{E}_b, \mathcal{R}_b, \mathcal{G}_b, \mathcal{C}_b \rangle$ is the behavioral subschema. The amount of knowledge represented in $\mathcal{CS}$ is large and makes it very difficult to manually extract fragments of interest to a user.
- **Focus Set:** it works as the conceptual schema viewpoint of the user. Therefore, a focus set is an initial point that should be extended with more knowledge. Formally, the focus set $\mathcal{FS}$ contains a small subset of the integrity constraints of $\mathcal{C}$ and $\mathcal{C}_b$, and the derivation

rules of $\mathcal{D}$, from which the user wants to know more about. Note that the size of the focus set is reduced with respect to the amount of schema rules from the large schema ($|\mathcal{FS}| \ll |\mathcal{C} \cup \mathcal{D} \cup \mathcal{C}_b|$). Also, it is mandatory for the user to select a non-empty focus set ($\mathcal{FS} \neq \emptyset$).

- **Size threshold:** it denotes the maximum expected number $\mathcal{K}$ of entity and event types in the output. Note that $|\mathcal{E}_{\mathcal{FS}} \cup \mathcal{E}_{b\mathcal{FS}}| \leq \mathcal{K} \leq |\mathcal{E} \cup \mathcal{E}_b|$, where $\mathcal{E}_{\mathcal{FS}}$ and $\mathcal{E}_{b\mathcal{FS}}$ are the set that include the entity and event types in the focus set $\mathcal{FS}$, respectively. Since the focus set contains schema rules, $\mathcal{E}_{\mathcal{FS}}$ and $\mathcal{E}_{b\mathcal{FS}}$ contain the entity and event types used and referenced inside the expressions within those rules.

- **Scope:** it indicates the kind of filtered schema the user wants to obtain. A *local* value for the scope implies that the filtered conceptual schema will only contain those elements referenced by the schema rules of focus. The *local* value of the scope forces $\mathcal{K} = |\mathcal{E}_{\mathcal{FS}}| + |\mathcal{E}_{b\mathcal{FS}}|$, in order to limit the size of the filtered conceptual schema to the referenced elements of the input schema rules. Any other value for the size threshold will be ignored. On the other hand, a *global* value for the scope will include additional knowledge of interest to the user until reaching the size threshold $\mathcal{K}$. In this case, the size threshold must satisfy $\mathcal{K} >= |\mathcal{E}_{\mathcal{FS}}| + |\mathcal{E}_{b\mathcal{FS}}|$.

- **Rejection Set:** the set $\mathcal{RS}$ with entity types of $\mathcal{E}$ and event types of $\mathcal{E}_b$ that the user does not want to obtain in the output. Note that it is disjoint with the focus set ($\mathcal{RS} \cap \mathcal{FS} = \emptyset$). Therefore, the rejection set must contain only elements that are not referenced by the schema rules of the focus set. By default, the rejection set is empty ($|\mathcal{RS}| \geq 0$). The rejection set is ignored when the scope of the request is set to local.

- **Importance method:** the algorithm $\mathcal{I}$ to compute the importance of entity types from $\mathcal{E}$ and event types from $\mathcal{E}_b$. By default $\mathcal{I} = I_{SM}$, the Simple Method described in Chap. 4.

Specific Output

The output of this filtering request is a filtered conceptual schema $\mathcal{CS}_\mathcal{F} = \langle \mathcal{SS}_\mathcal{F}, \mathcal{BS}_\mathcal{F} \rangle$, where $\mathcal{SS}_\mathcal{F} = \langle \mathcal{E}_\mathcal{F}, \mathcal{R}_\mathcal{F}, \mathcal{T}_\mathcal{F}, \mathcal{G}_\mathcal{F}, \mathcal{C}_\mathcal{F}, \mathcal{D}_\mathcal{F} \rangle$ is the structural subschema, and $\mathcal{BS}_\mathcal{F} = \langle \mathcal{E}_{b\mathcal{F}}, \mathcal{R}_{b\mathcal{F}}, \mathcal{G}_{b\mathcal{F}}, \mathcal{C}_{b\mathcal{F}} \rangle$ is the behavioral subschema. The specific constraints that $\mathcal{CS}_\mathcal{F}$ must satisfy are described as follows:

[C1] $\mathcal{E}_\mathcal{F}$ contains the entity types $\mathcal{E}_{\mathcal{FS}}$ referenced by schema rules from the focus set $\mathcal{FS}$. If the scope is set to *local*, the entity types of $\mathcal{E}_\mathcal{F}$ only contain the projection of those attributes referenced by schema rules from the focus set.

[C2] If the scope is set to global, $\mathcal{E}_\mathcal{F}$ does not contain the entity types from the rejection set $\mathcal{RS}$.

[C3] $\mathcal{G}_\mathcal{F}$ contains direct generalization relationships between entity types of $\mathcal{E}_\mathcal{F}$.

[C4] $\mathcal{G}_{b\mathcal{F}}$ contains direct generalization relationships between event types of $\mathcal{E}_{b\mathcal{F}}$.

[**C5**] $\mathcal{E}_{b\mathcal{F}}$ contains the event types $\mathcal{E}_{b\mathcal{FS}}$ referenced by schema rules from the focus set $\mathcal{FS}$. If the scope is set to *local*, the event types of $\mathcal{E}_{b\mathcal{FS}}$ only contain the projection of those attributes referenced by schema rules from the focus set.

[**C6**] If the scope is set to global, $\mathcal{E}_{b\mathcal{F}}$ does not contain the event types from the rejection set $\mathcal{RS}$.

[**C7**] $\mathcal{C}_{\mathcal{F}}$, $\mathcal{D}_{\mathcal{F}}$, and $\mathcal{C}_{b\mathcal{F}}$ contain the integrity constraints and derivation rules from the focus set $\mathcal{FS}$.

[**C8**] If c is an integrity constraint or derivation rule of $\mathcal{C}$, $\mathcal{D}$ or $\mathcal{C}_b$ out of the focus set but defined in the context of an schema element of $\mathcal{CS}_{\mathcal{F}}$ and all of the schema elements referenced by c belong to $\mathcal{CS}_{\mathcal{F}}$, then c is included in $\mathcal{C}_{\mathcal{F}}$, $\mathcal{D}_{\mathcal{F}}$ or $\mathcal{C}_{b\mathcal{F}}$.

[**C9**] If r is a relationship type of $\mathcal{R}$ and its participant entity types belong to $\mathcal{E}_{\mathcal{F}}$, or are ascendants of entity types of $\mathcal{E}_{\mathcal{F}}$ (in which case a projection is needed), then r is included in $\mathcal{R}_{\mathcal{F}}$. The same behavior applies to relationship types of $\mathcal{R}_b$ to be included in $\mathcal{R}_{b\mathcal{F}}$. If the scope was set to *local*, only those relationship types referenced by schema rules from the focus set are included into $\mathcal{R}_{\mathcal{F}}$ and $\mathcal{R}_{b\mathcal{F}}$.

[**C10**] If d is a data type of $\mathcal{T}$ and it is used by attributes of entity types of $\mathcal{E}_{\mathcal{F}}$, event types of $\mathcal{E}_{b\mathcal{F}}$, or schema rules of $\mathcal{C}_{\mathcal{F}}$, $\mathcal{C}_{b\mathcal{F}}$ or $\mathcal{D}_{\mathcal{F}}$, then d is included in $\mathcal{T}_{\mathcal{F}}$ of $\mathcal{CS}_{\mathcal{F}}$. If the scope was set to local, only those data types referenced by schema rules from the focus set are included into $\mathcal{T}_{\mathcal{F}}$.

[**C11**] If e_1 and e_2 are entity types of $\mathcal{E}_{\mathcal{F}}$ and does not exist a direct generalization between them in $\mathcal{G}_{\mathcal{F}}$ nor a path of direct generalizations of $\mathcal{G}_{\mathcal{F}}$ traversing entity types e_i of $\mathcal{E}_{\mathcal{F}}$, but both e_1 and e_2 belong to different levels of the same hierarchy in $\mathcal{G}$ of $\mathcal{CS}$, a direct generalization g' is included between e_1 and e_2 in $\mathcal{G}_{\mathcal{F}}$ but is marked as derived. The same behavior applies to pairs of event types of $\mathcal{E}_{b\mathcal{F}}$.

[**C12**] If the scope is set to *global* and c is a constraint of $\mathcal{C}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_{\mathcal{F}}$ and references schema elements out of $\mathcal{CS}_{\mathcal{F}}$ only the header of such constraint is included in $\mathcal{C}_{\mathcal{F}}$ or $\mathcal{C}_{b\mathcal{F}}$.

[**C13**] If the scope is set to *global* and d is a derivation rule of $\mathcal{D}$ whose context belongs to $\mathcal{CS}_{\mathcal{F}}$ and references schema elements out of $\mathcal{CS}_{\mathcal{F}}$, the context element of the rule is marked as materialized in $\mathcal{CS}_{\mathcal{F}}$ and d is not included in $\mathcal{D}_{\mathcal{F}}$.

Filtering Stages

The filtering request for schema rules follows the specific methodology presented in Sect. 5.4 of Ch. 5. In the following, we present a brief summary of the different stages and steps that belong to this filtering request. Figure 6.12 indicates the steps when the scope is *local* while Fig. 6.13 shows the steps when the scope is *global*.

- **Stage 1: Metrics.** The first step in this stage (see Activity 1.1 in Fig. 6.13) processes the input of the filtering request. It creates some auxiliary sets to gather the entity and

event types referenced by the schema rules from the focus set. In this particular case, the auxiliary set $\mathcal{E}_{\mathcal{FS}}$ and $\mathcal{E}_{b\mathcal{FS}}$ contains the entity and event types referenced by the schema rules from the focus set. Also, the set $\mathcal{R}_{\mathcal{FS}}$ that contains relationship types from the focus set is empty since this filtering request focus on schema rules to construct the input. In addition to it, if the scope is set to *local* the method creates an auxiliary set with the attributes of entity and event types referenced by the schema rules from the focus set, and ignores the values of the rejection set $\mathcal{RS}$ and size threshold $\mathcal{K}$. The rest of the steps follow the same indications presented in Sect. 5.4.1 of Ch. 5. If the scope is set to *global*, the method computes the importance, closeness, and interest metrics, and sorts the entity and event types in a ranking. Otherwise, the method only requires the entity, event, and relationship types referenced by schema rules from the focus set.

- **Stage 2: Entity and Event Types.** This stage follows the same steps as presented in Sect. 5.4.2 of Ch. 5. If the scope is set to *global*, the method selects the top entity and event types from the interest ranking taking into account the size threshold of the input and includes them in the resulting filtered conceptual schema. Otherwise, if the scope is set to *local* the method selects the entity and event types referenced by the schema rules of focus and includes them in the resulting filtered conceptual schema only with the projection of those attributes that also were referenced in the schema rules of focus. Note that since we see attributes as binary relationship types between an entity or event type and a datatype, the projection of attributes is similar than the projection of relationship types presented in Sect. 5.4.3 of Ch. 5

- **Stage 3: Relationship Types.** This stage follows the same steps as presented in Sect. 5.4.3 of Ch. 5. The method classifies the relationship types according to their participants as referentially-complete or referentially-partial relationship types. Note that those relationship types used in expressions of the schema rules from the focus set are always referentially-complete since their participants are members of the sets of entity and event types $\mathcal{E}_{\mathcal{FS}}$ and $\mathcal{E}_{b\mathcal{FS}}$ referenced from the focus set. Therefore, such relationship types will be part of the filtered schema. Finally, the method projects and selects the final relationship types that are part of the resulting filtered conceptual schema. If the scope is set to *local*, the method only includes into the filtered schema those relationship types that are referenced by the schema rules from the focus set.

- **Stage 4: Generalizations.** This stage follows the same steps as presented in Sect. 5.4.4 of Ch. 5. The method processes the direct and creates indirect generalizations to construct the hierarchies of the resulting filtered conceptual schema.

- **Stage 5: Schema Rules.** This stage follows the same steps as presented in Sect. 5.4.5 of Ch. 5. The method selects the schema rules defined in the context of elements from the filtered schema and processes the referentially-incomplete ones in order to construct the rules of the resulting filtered conceptual schema. The schema rules from the focus set are all referentially-complete since the entity and event types they reference are included in the filtered conceptual schema.

- **Stage 6: Data Types.** This stage follows the same steps as presented in Sect. 5.4.6 of Ch. 5. The method includes in the filtered schema those data types referenced or used by other elements within such schema.

- **Stage 7: Presentation.** This stage follows the same steps as presented in Sect. 5.4.7 of Ch. 5. The method presents the filtered schema to the user. The elements to highlight are the schema rules from the focus set $\mathcal{FS}$.

Method Correctness

The proposed activities in the stages of the filtering request transform the input into a valid output in the form of a filtered conceptual schema that satisfies a set constraints over such schema. In the following we verify the correctness of the method according to those constraints and the activities that satisfy each of them:

- Constraint **[C1]** is satisfied by the activity 2.2 of the filtering request.
- Constraint **[C2]** is satisfied by the activity 2.2 of the filtering request.
- Constraint **[C3]** is satisfied by the activity 4.1 of the filtering request.
- Constraint **[C4]** is satisfied by the activity 4.2 of the filtering request.
- Constraint **[C5]** is satisfied by the activity 2.3 of the filtering request.
- Constraint **[C6]** is satisfied by the activity 2.3 of the filtering request.
- Constraint **[C7]** is satisfied by the activities 5.1 and 5.2 of the filtering request.
- Constraint **[C8]** is satisfied by the activities 5.1 and 5.2 of the filtering request.
- Constraint **[C9]** is satisfied by the activities 3.1, 3.2, 3.3, and 3.4 of filtering request.
- Constraint **[C10]** is satisfied by the activity 6.1 of the filtering request.
- Constraint **[C11]** is satisfied by the activities 4.3 and 4.4 of the filtering request.
- Constraint **[C12]** is satisfied by the activities 5.1 and 5.3 of the filtering request.
- Constraint **[C13]** is satisfied by the activities 5.1 and 5.4 of the filtering request.

The activities 1.1, 1.2, 1.3, 1.5, and 1.5 of the filtering request process the input and deal with the computation of relevance metrics to filter the large schema. The activity 2.1 selects the additional entity and event types to complete the knowledge from the focus set. The activity 7.1 presents the resulting filtered conceptual schema to the user.

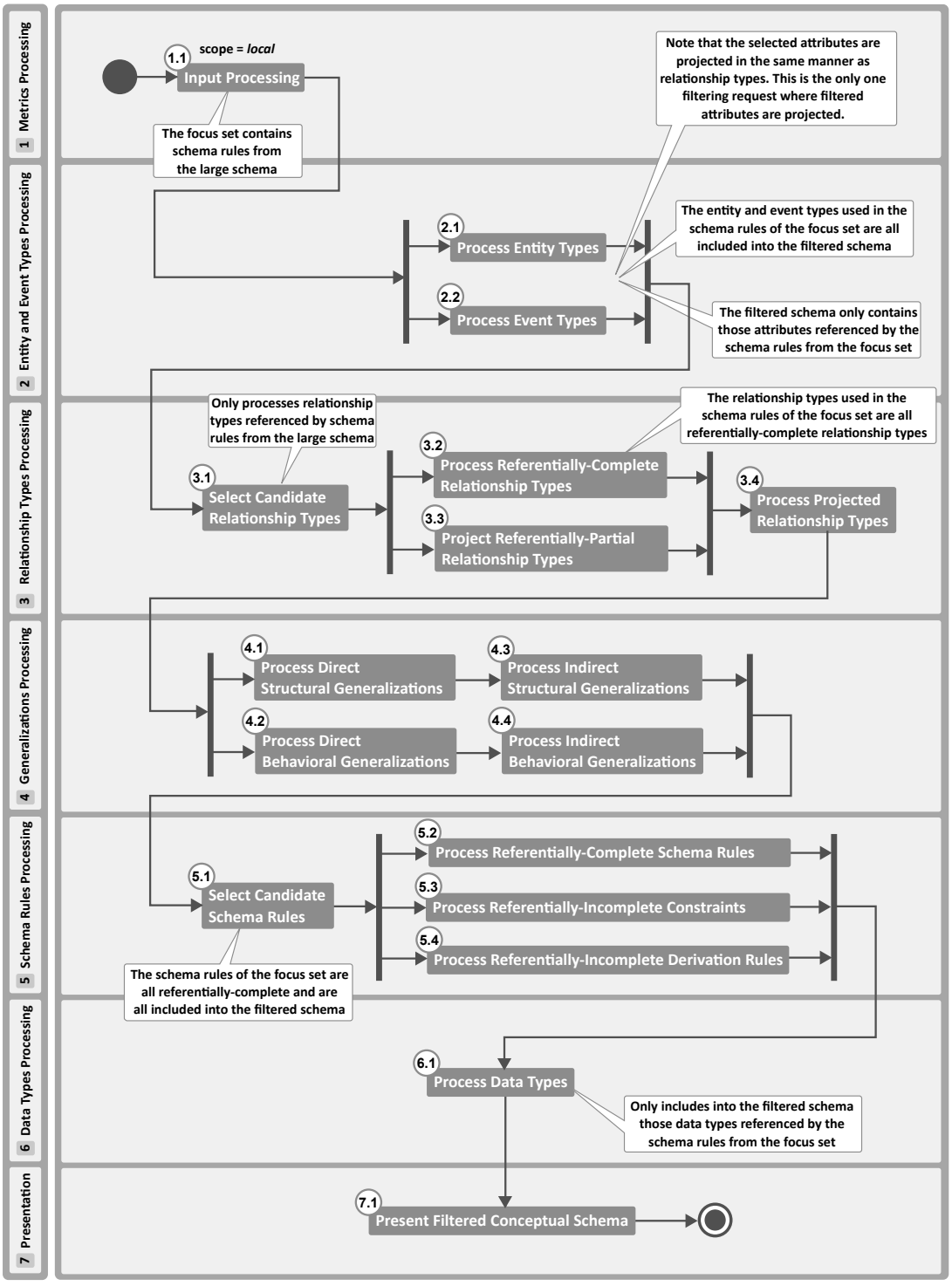


Figure 6.12. Activity diagram for the filtering request $\mathcal{F}_2$ when the scope is *local*.

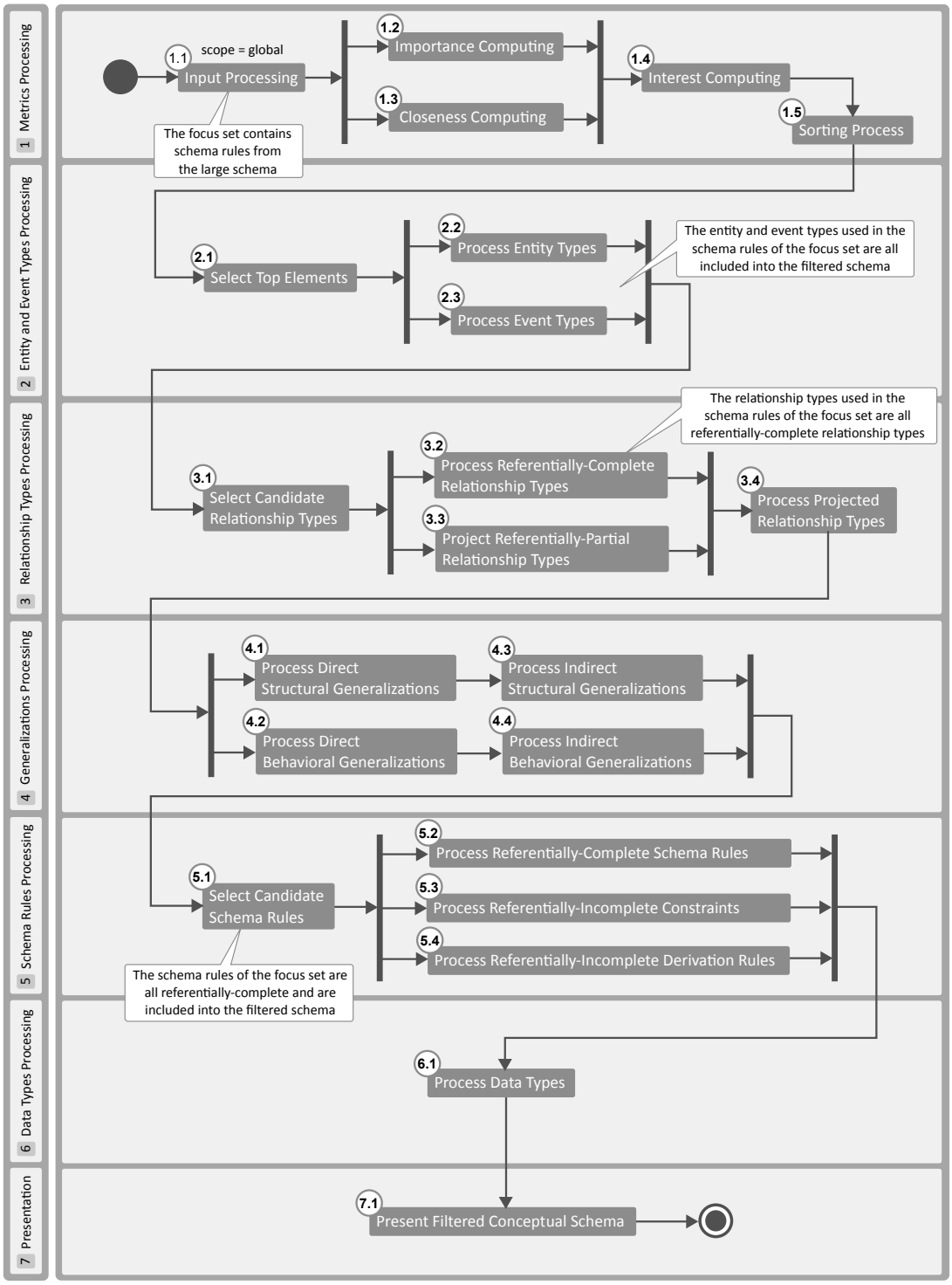


Figure 6.13. Activity diagram for the filtering request $\mathcal{F}_2$ when the scope is *global*.

Example of $\mathcal{F}_2$

The filtering scenario proposed in this example describes a user of the conceptual schema of the Magento e-commerce system that wants to know the schema elements that participate in the specification of a schema rule defined in the schema of Magento. Concretely, the user is interested in the schema rule `isIdentifiedByItsProductItsCartAndItsOptions` defined in the context of the entity type ShoppingCartItem:

```
context ShoppingCartItem inv isIdentifiedByItsProductItsCartAndItsOptions:
  let textOp: Bag(Tuple(o:TextOption,v:String)) =
   self.textOptionRating->collect(t|
      Tuple{o=t.textOption,v=t.value})
  in
   let dateOp: Bag(Tuple(o:DateOption,v:Date)) =
    self.dateOptionRating->collect(t|
     Tuple{o=t.dateOption,v=t.value})
   in
    ShoppingCartItem.allInstances()
    ->isUnique(Tuple{s=shoppingCart,
                     p=product,
                     t=textOp,
                     d=dateOp,
                     v=optionValueInOption})
```

Therefore, the user constructs the input of the method as follows:

```
CS = Magento
FS = {isIdentifiedByItsProductItsCartAndItsOptions}
RS = ∅
scope = local
I  = CEntityRank Extended
```

As a result of the application of the filtering request for schema rules, the user obtains the filtered conceptual schema depicted in Fig. 6.14. It presents only those schema elements that are referenced by the schema rule (the scope is *local*), including entity types and relationship types. Basically, a ShoppingCartItem is connected with a Product, a ShoppingCart, and several options, including text options, date options, and value options. The schema rule of focus indicates that an instance of ShoppingCartItem is identified by its ShoppingCart, its Product, a tuple with the instances of TextOption connected to it and TextOptionRating values, a tuple with the instances of DateOption connected to it and DateOptionRating values, and a set of instance of OptionValueInOption connected to it. Therefore, two different instances of ShoppingCartItem must have different instances of the elements that identify them.

```
context ShoppingCartItem
 inv isIdentifiedByItsProductItsCartAndItsOptions:
  let textOp: Bag(Tuple(o:TextOption,v:String)) =
   self.textOptionRating->collect(t|
      Tuple{o=t.textOption,v=t.value})
  in
   let dateOp: Bag(Tuple(o:DateOption,v:Date)) =
    self.dateOptionRating->collect(t|
     Tuple{o=t.dateOption,v=t.value})
   in
    ShoppingCartItem.allInstances()
    ->isUnique(Tuple{s=shoppingCart,
                     p=product,
                     t=textOp,
                     d=dateOp,
                     v=optionValueInOption})
```

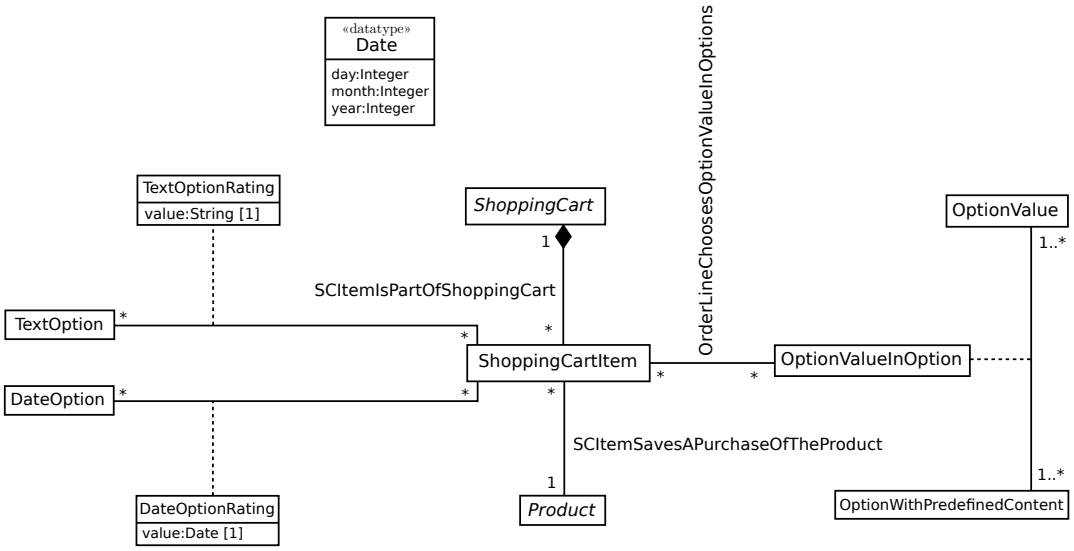


Figure 6.14. Filtered schema for the schema rule of ShoppingCartItem obtained by applying $\mathcal{F}_2$.

On the other hand, the user can construct the same input of the method but setting the scope to *global*:

$\mathcal{CS}$ = Magento
$\mathcal{FS}$ = {isIdentifiedByItsProductItsCartAndItsOptions}
$\mathcal{RS}$ = $\emptyset$
scope = *global*
$\mathcal{K}$ = 10
$\mathcal{I}$ = CEntityRank Extended

Figure 6.15 and Fig. 6.16 present the results for the new request. It is important to note that we also show additional integrity constraints that are referentially complete taking into account the schema elements that appear in the filtered schema our method provides for this filtering request.

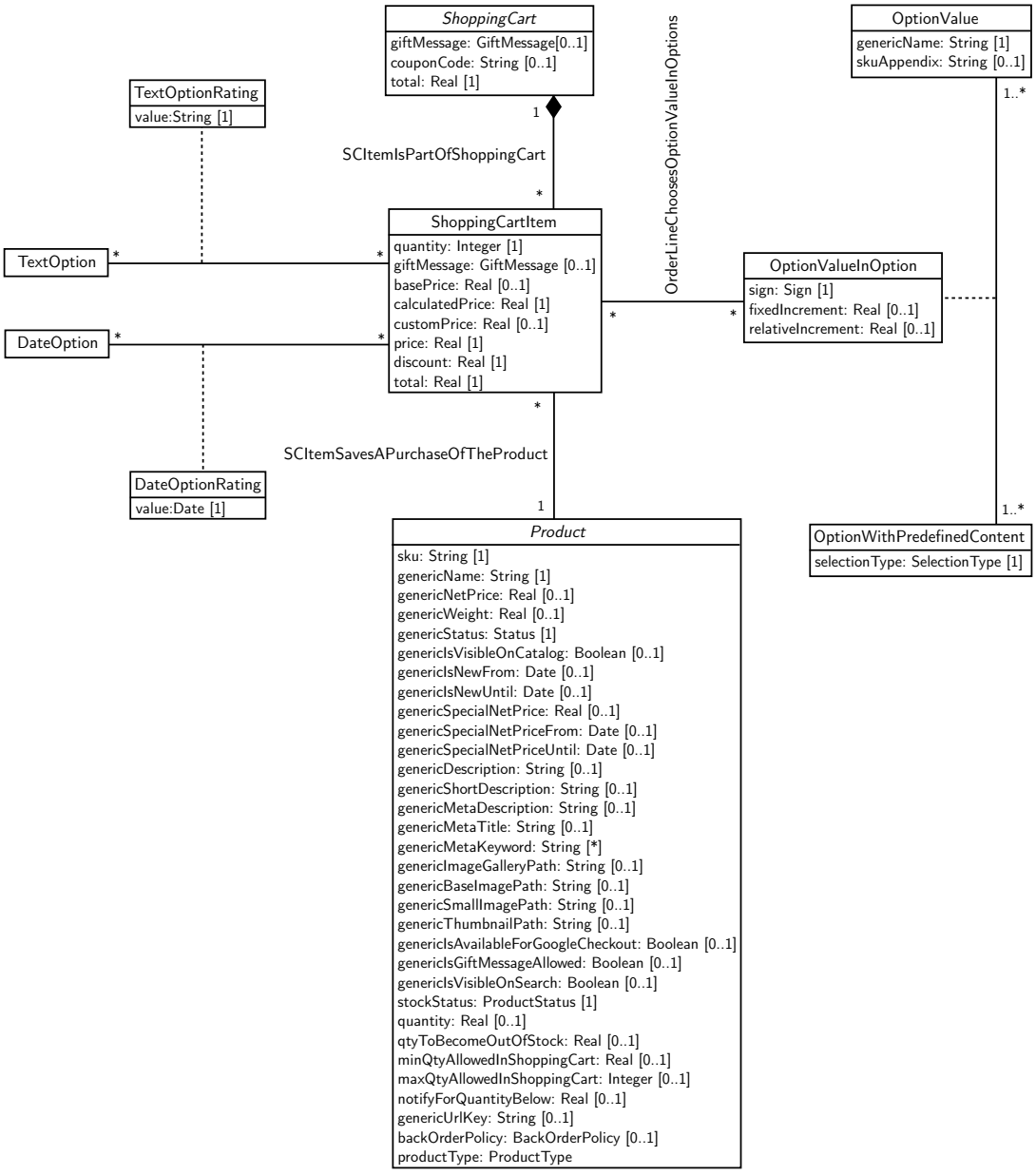
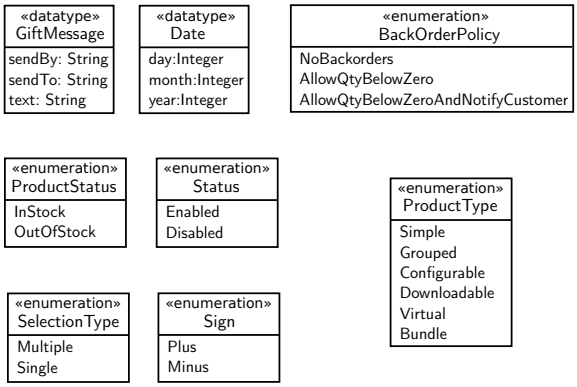


Figure 6.15. Filtered schema for the schema rule of ShoppingCartItem when the scope is *global* (I).




```
context ShoppingCartItem
 inv isIdentifiedByItsProductItsCartAndItsOptions:
  let textOp: Bag(Tuple(o:TextOption,v:String)) =
   self.textOptionRating->collect(t|
      Tuple{o=t.textOption,v=t.value})
  in
   let dateOp: Bag(Tuple(o:DateOption,v:Date)) =
    self.dateOptionRating->collect(t|
     Tuple{o=t.dateOption,v=t.value})
   in
    ShoppingCartItem.allInstances()
    ->isUnique(Tuple{s=shoppingCart,
                     p=product,
                     t=textOp,
                     d=dateOp,
                     v=optionValueInOption})
```

```
context ShoppingCartItem
 inv ratesOnlyOptionsOfItsProduct:
  self.textOption->forAll(to|to.product=self.product)
  and
  self.dateOption->forAll(dop|dop.product=self.product)
  and
  self.optionValueInOption->forAll(ovio|
   ovio.optionWithPredefinedContent.product
   =
   self.product)

context OptionValue
  inv isIdentifiedByItsGenericName:
   OptionValue.allInstances()->isUnique(genericName)

context OptionValueInOption
  inv definesRelativeOrFixedIncrementButNotAll:
   self.fixedIncrement.isUndefined()
   <>
   self.relativeIncrement.isUndefined()

context OptionWithPredefinedContent
  inv isRatedWithOnlyOneValueWhenItIsNeeded:
   self.selectionType = SelectionType::Single
   implies
   self.optionValueInOption->forAll(ov1,ov2 |
    ov1 <> ov2
    implies
    ov1.shoppingCartItem <> ov2.shoppingCartItem
   )

context Product
 inv isIdentifiedBySku:
  Product.allInstances()->isUnique(sku)
```

Figure 6.16. Filtered schema for the schema rule of ShoppingCartItem when the scope is *global* (II).

6.3.3 $\mathcal{F}_3$: Filtering Request for Event Types

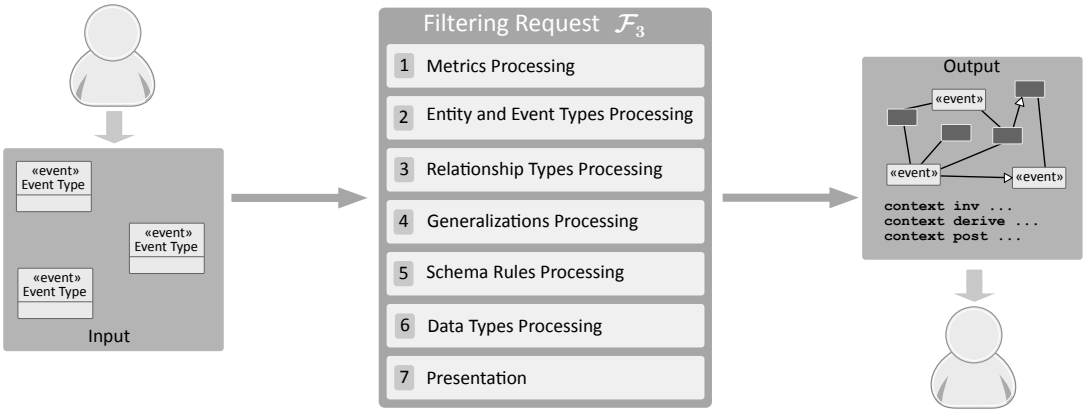


Figure 6.17. $\mathcal{F}_3$: Filtering request for event types.

Application Scenario

$$\mathcal{F}_3 : \mathcal{P}_{\geq 1}(\mathcal{E}_b) \rightarrow \mathcal{CS}_{\mathcal{F}}$$

The user focuses on a set of event types from a large conceptual schema. The user is aware of those event types or she has accessed them via previous filtering requests. The information need consists in obtaining more knowledge from the schema with relation to the event types in the user focus. The method obtains the elements of interest to the user according to the initial selection and the characteristics represented in the large schema. As output, the user obtains a small-sized filtered conceptual schema that includes the combination of the initial event types of focus with the elements of interest gathered by our methodology.

Specific Input

The input for the filtering request $\mathcal{F}_3$ contains the same elements as indicated in Sect. 6.2. In addition, we present here a detailed description of the particularities of such elements for the specific input of $\mathcal{F}_3$.

- **Large conceptual schema:** the source schema $\mathcal{CS} = \langle \mathcal{SS}, \mathcal{BS} \rangle$ where $\mathcal{SS} = \langle \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ is the structural subschema, and $\mathcal{BS} = \langle \mathcal{E}_b, \mathcal{R}_b, \mathcal{G}_b, \mathcal{C}_b \rangle$ is the behavioral subschema. The amount of knowledge represented in $\mathcal{CS}$ is large and makes it very difficult to manually extract fragments of interest to a user.
- **Focus Set:** it works as the conceptual schema viewpoint of the user. Therefore, a focus set is an initial point that should be extended with more knowledge. Formally, the focus set $\mathcal{FS}$ contains a small subset of the event types of $\mathcal{E}_b$ from which the user wants to know

more about. Note that the size of the focus set is reduced with respect to the amount of event types from the large schema ($|\mathcal{FS}| \ll |\mathcal{E}_b|$). Also, it is mandatory for the user to select a non-empty focus set ($\mathcal{FS} \neq \emptyset$).

- **Size threshold:** it denotes the maximum expected number $\mathcal{K}$ of entity and event types in the output. Note that $|\mathcal{E}_{b\mathcal{FS}}| \leq \mathcal{K} \leq |\mathcal{E} \cup \mathcal{E}_b|$, where $\mathcal{E}_{b\mathcal{FS}}$ is the set that includes the event types in the focus set $\mathcal{FS}$. Thus, $\mathcal{E}_{b\mathcal{FS}}$ equals to the focus set, since $\mathcal{FS}$ only contains event types.
- **Rejection Set:** the set $\mathcal{RS}$ with entity types of $\mathcal{E}$ and event types of $\mathcal{E}_b$ that the user does not want to obtain in the output. Note that it is disjoint with the focus set ($\mathcal{RS} \cap \mathcal{FS} = \emptyset$). By default, the rejection set is empty ($|\mathcal{RS}| \geq 0$).
- **Importance method:** the algorithm $\mathcal{I}$ to compute the importance of entity types from $\mathcal{E}$ and event types from $\mathcal{E}_b$. By default $\mathcal{I} = I_{SM}$, the Simple Method described in Chap. 4.

Specific Output

The output of this filtering request is a filtered conceptual schema $\mathcal{CS}_\mathcal{F} = \langle \mathcal{SS}_\mathcal{F}, \mathcal{BS}_\mathcal{F} \rangle$, where $\mathcal{SS}_\mathcal{F} = \langle \mathcal{E}_\mathcal{F}, \mathcal{R}_\mathcal{F}, \mathcal{T}_\mathcal{F}, \mathcal{G}_\mathcal{F}, \mathcal{C}_\mathcal{F}, \mathcal{D}_\mathcal{F} \rangle$ is the structural subschema, and $\mathcal{BS}_\mathcal{F} = \langle \mathcal{E}_{b\mathcal{F}}, \mathcal{R}_{b\mathcal{F}}, \mathcal{G}_{b\mathcal{F}}, \mathcal{C}_{b\mathcal{F}} \rangle$ is the behavioral subschema. The specific constraints that $\mathcal{CS}_\mathcal{F}$ must satisfy are described as follows:

[C1] $\mathcal{E}_\mathcal{F}$ does not contain the entity types from the rejection set $\mathcal{RS}$.

[C2] $\mathcal{E}_{b\mathcal{F}}$ contains the event types $\mathcal{E}_{b\mathcal{FS}}$ from the focus set $\mathcal{FS}$.

[C3] $\mathcal{E}_{b\mathcal{F}}$ does not contain the event types from the rejection set $\mathcal{RS}$.

[C4] $\mathcal{G}_{b\mathcal{F}}$ contains direct generalization relationships between event types of $\mathcal{E}_{b\mathcal{F}}$.

[C5] $\mathcal{G}_\mathcal{F}$ contains direct generalization relationships between entity types of $\mathcal{E}_\mathcal{F}$.

[C6] If c is an integrity constraint or derivation rule of $\mathcal{C}$, $\mathcal{D}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_\mathcal{F}$ and any of the schema elements referenced by c belong to $\mathcal{CS}_\mathcal{F}$, then c is included in $\mathcal{C}_\mathcal{F}$, $\mathcal{D}_\mathcal{F}$ or $\mathcal{C}_{b\mathcal{F}}$.

[C7] If r is a relationship type of $\mathcal{R}$ and its participant entity types belong to $\mathcal{E}_\mathcal{F}$, or are ascendants of entity types of $\mathcal{E}_\mathcal{F}$ (in which case a projection is needed), then r is included in $\mathcal{R}_\mathcal{F}$. The same behavior applies to relationship types of $\mathcal{R}_b$ to be included in $\mathcal{R}_{b\mathcal{F}}$.

[C8] If d is a data type of $\mathcal{T}$ and it is used by attributes of entity types of $\mathcal{E}_\mathcal{F}$, event types of $\mathcal{E}_{b\mathcal{F}}$, or schema rules of $\mathcal{C}_\mathcal{F}$, $\mathcal{C}_{b\mathcal{F}}$ or $\mathcal{D}_\mathcal{F}$, then d is included in $\mathcal{T}_\mathcal{F}$ of $\mathcal{CS}_\mathcal{F}$.

[C9] If e_1 and e_2 are entity types of $\mathcal{E}_\mathcal{F}$ and does not exist a direct generalization between them in $\mathcal{G}_\mathcal{F}$ nor a path of direct generalizations of $\mathcal{G}_\mathcal{F}$ traversing entity types e_i of $\mathcal{E}_\mathcal{F}$, but both e_1 and e_2 belong to different levels of the same hierarchy in $\mathcal{G}$ of $\mathcal{CS}$, a direct generalization g' is included between e_1 and e_2 in $\mathcal{G}_\mathcal{F}$ but is marked as derived. The same behavior applies to pairs of event types of $\mathcal{E}_{b\mathcal{F}}$.

[C10] If c is a constraint of $\mathcal{C}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_\mathcal{F}$ and references schema elements out of $\mathcal{CS}_\mathcal{F}$ only the header of such constraint is included in $\mathcal{C}_\mathcal{F}$ or $\mathcal{C}_{b\mathcal{F}}$. If d is a derivation rule of $\mathcal{D}$ whose context belongs to $\mathcal{CS}_\mathcal{F}$ and references schema elements out of $\mathcal{CS}_\mathcal{F}$, the context element of the rule is marked as materialized in $\mathcal{CS}_\mathcal{F}$ and d is not included in $\mathcal{D}_\mathcal{F}$.

Filtering Stages

The filtering request for event types follows the specific methodology presented in Sect. 5.4 of Ch. 5. In the following, we present a brief summary of the different stages and steps (see Fig. 6.18) that belong to this filtering request.

- **Stage 1: Metrics.** The first step in this stage (see Activity 1.1 in Fig. 6.18) processes the input of the filtering request. It creates some auxiliary sets to gather the event types from the focus set. In this particular case, the auxiliary set $\mathcal{E}_{b\mathcal{FS}}$ contains the event types from the focus set. Also, the set $\mathcal{E}_{\mathcal{FS}}$ that contains entity types from the focus set and the set $\mathcal{R}_{\mathcal{FS}}$ that contains relationship types from the focus set are both empty since this filtering request focus on the event types to construct the input. The rest of the steps follow the same indications presented in Sect. 5.4.1 of Ch. 5. The method computes the importance, closeness, and interest metrics, and sorts the entity and event types in a ranking.
- **Stage 2: Entity and Event Types.** This stage follows the same steps as presented in Sect. 5.4.2 of Ch. 5. The method selects the top entity and event types from the interest ranking taking into account the size threshold of the input and includes them in the resulting filtered conceptual schema. The event types of the focus set are all included in the filtered schema.
- **Stage 3: Relationship Types.** This stage follows the same steps as presented in Sect. 5.4.3 of Ch. 5. The method classifies the relationship types according to their participants as referentially-complete or referentially-partial relationship types. Then, the method projects and selects the final relationship types that are part of the resulting filtered conceptual schema.
- **Stage 4: Generalizations.** This stage follows the same steps as presented in Sect. 5.4.4 of Ch. 5. The method processes the direct and creates indirect generalizations to construct the hierarchies of the resulting filtered conceptual schema.
- **Stage 5: Schema Rules.** This stage follows the same steps as presented in Sect. 5.4.5 of Ch. 5. The method selects the schema rules defined in the context of elements from the filtered schema and processes the referentially-incomplete ones in order to construct the rules of the resulting filtered conceptual schema.
- **Stage 6: Data Types.** This stage follows the same steps as presented in Sect. 5.4.6 of Ch. 5. The method includes in the filtered schema those data types referenced or used by other elements within such schema.

- **Stage 7: Presentation.** This stage follows the same steps as presented in Sect. 5.4.7 of Ch. 5. The method presents the filtered schema to the user. The elements to highlight are the event types from the focus set $\mathcal{FS}$.

Method Correctness

The proposed activities in the stages of the filtering request transform the input into a valid output in the form of a filtered conceptual schema that satisfies a set constraints over such schema. In the following we verify the correctness of the method according to those constraints and the activities that satisfy each of them:

- Constraint **[C1]** is satisfied by the activity 2.2 of the filtering request.
- Constraint **[C2]** is satisfied by the activity 2.3 of the filtering request.
- Constraint **[C3]** is satisfied by the activity 2.3 of the filtering request.
- Constraint **[C4]** is satisfied by the activity 4.2 of the filtering request.
- Constraint **[C5]** is satisfied by the activity 4.1 of the filtering request.
- Constraint **[C6]** is satisfied by the activities 5.1 and 5.2 of the filtering request.
- Constraint **[C7]** is satisfied by the activities 3.1, 3.2, 3.3, and 3.4 of the filtering request.
- Constraint **[C8]** is satisfied by the activity 6.1 of the filtering request.
- Constraint **[C9]** is satisfied by the activities 4.3 and 4.4 of filtering request method.
- Constraint **[C10]** is satisfied by the activities 5.1, 5.3, and 5.4 of the filtering request.

The activities 1.1, 1.2, 1.3, 1.5, and 1.5 of the filtering request process the input and deal with the computation of relevance metrics to filter the large schema. The activity 2.1 selects the additional entity and event types to complete the knowledge from the focus set. The activity 7.1 presents the resulting filtered conceptual schema to the user.

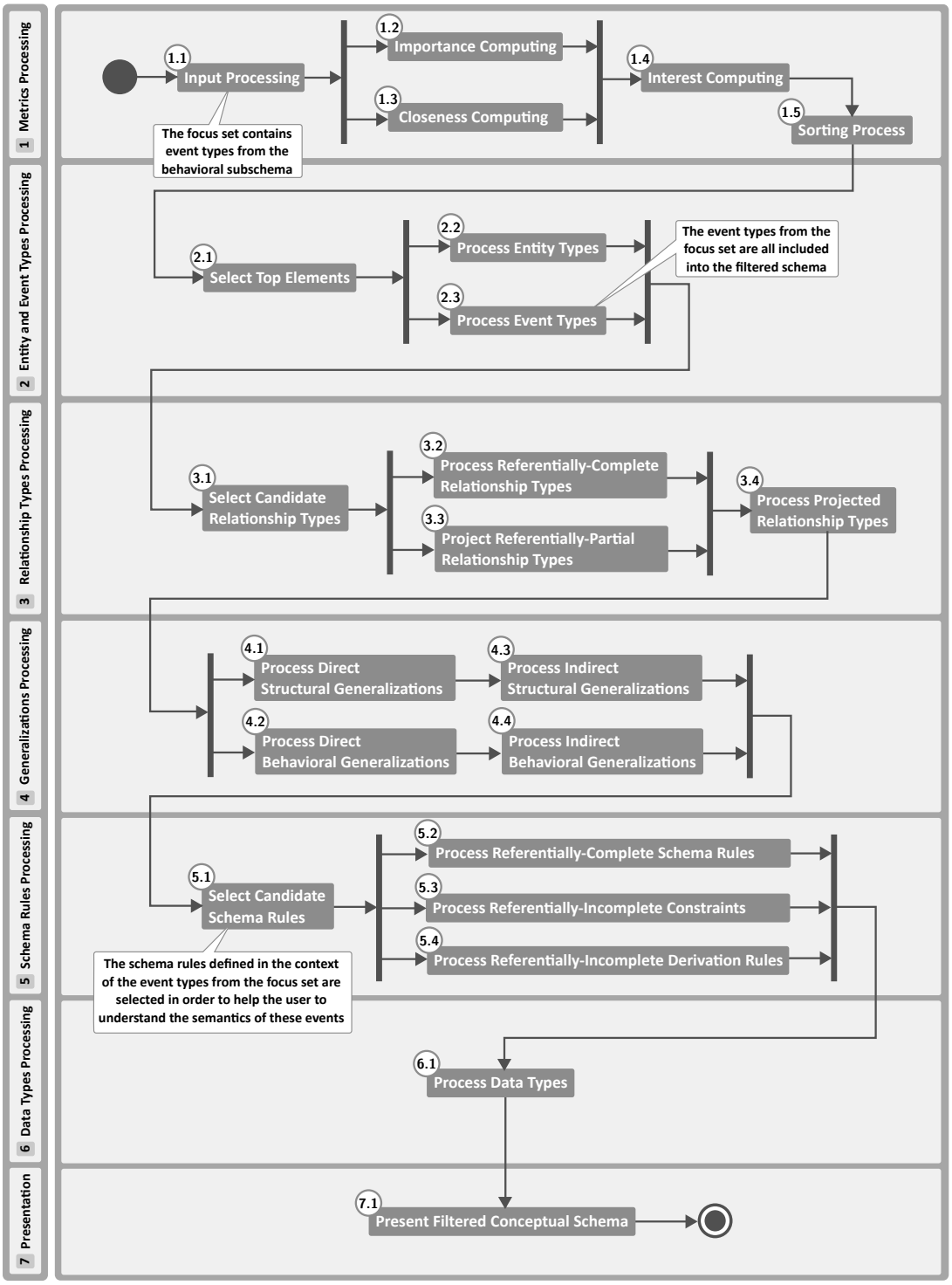


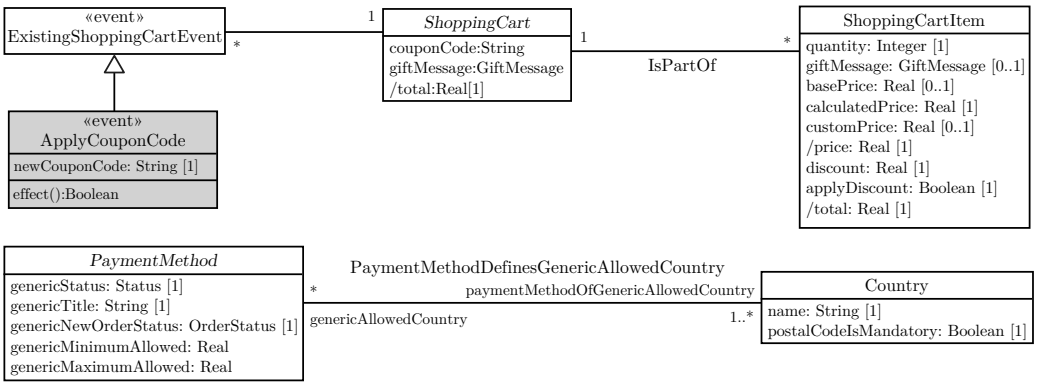
Figure 6.18. Activity diagram for the filtering request $\mathcal{F}_3$.

Example of $\mathcal{F}_3$

The filtering scenario proposed in this example describes a user of the conceptual schema of the Magento e-commerce system that wants to know the schema elements of interest with respect to a set of event types. Concretely, the user focuses in the event type ApplyGrouponCode. The user constructs an input to the filtering request as follows:

$\mathcal{CS}$ = Magento
$\mathcal{FS}$ = {ApplyGrouponCode}
$\mathcal{RS}$ = {Product, StoreView, Website}
$\mathcal{K}$ = 6
$\mathcal{I}$ = CEntityRank Extended

Figure 6.19 shows the graphical representation of the filtered schema our method obtains taking into account the previous input. Note that the user specifies a nonempty rejection set including the entity types Product, StoreView, and Website. Since these entities are of high importance and are close to most of the other concepts in the schema, a good way to hide them is to use the rejection set of the input.




```
context Country
  inv isIdentifiedByItsName:
   Country.allInstances() -> isUnique(name)

context ApplyCouponCode::effect()
  post :
   self.shoppingCart.couponCode = self.newCouponCode

context ShoppingCart::total:Real
  derive : self.shoppingCartItem.total->sum()

context ShoppingCartItem::price:Real
  derive : if not self.customPrice.oclIsUndefined()
           then self.customPrice
           else self.calculatedPrice
           endif

context ShoppingCartItem::total:Real
  derive : if self.applyDiscount
           then self.price * self.quantity - self.discount
           else self.price * self.quantity
           endif
```

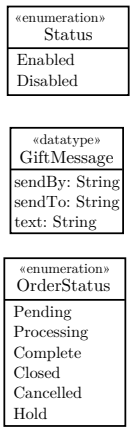


Figure 6.19. Filtered schema for the event type ApplyCouponCode.

The resulting schema indicates that the event type of focus ApplyCouponCode is a descendant of the event type ExistingShoppingCartEvent, which is related to a ShoppingCart. The effect of this event type is specified in OCL and describes that the value of the couponCode attribute of the ShoppingCart that is connected to the event takes the new vale described in the newCouponCode attribute of ApplyCouponCode.

Note that we include derivation rules for the attributes price and total of ShoppingCartItem and total of ShoppingCart, because the OCL expressions within these rules are referentially-complete according to the filtered schema.

The filtered schema also shows that an instance of ShoppingCart is related to several instances of ShoppingCartItem through the relationship type IsPartOf. In addition to it, we observe that payment methods in Magento can define a set of countries where the payment is allowed. We obtain information about payment methods since the entity type PaymentMethod is close to the event type of focus that deals with shopping carts. Although in this case the filtered conceptual schema is unconnected –it contains two connected components– the knowledge we retrieve is meaningful and we may start a new interaction in order to know more about the existing elements that are between ShoppingCart and PaymentMethod. Additionally, we can increase the value of $\mathcal{K}$ to achieve a connected schema.

6.3.4 $\mathcal{F}_4$: Filtering Request for a Conceptual Schema

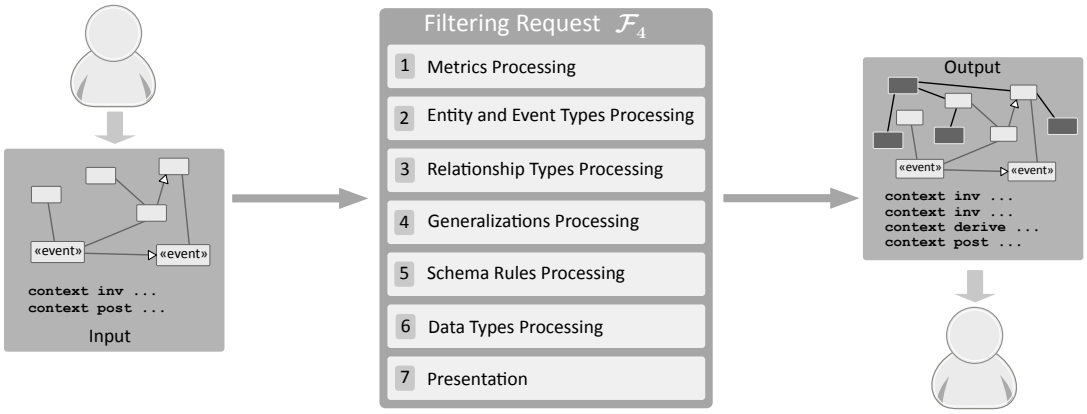


Figure 6.20. $\mathcal{F}_4$: Filtering request for a conceptual schema.

Application Scenario

$$\mathcal{F}_4 : \mathcal{CS} \rightarrow \mathcal{CS}_{\mathcal{F}}$$

The user focuses on a small conceptual schema from the large one. The user is aware of the elements that conform the schema or she has accessed them via previous filtering requests. The information need consists in obtaining more knowledge from the large schema with relation to the elements in the user focus. The method obtains the elements of interest to the user according to the initial selection and the characteristics represented in the large schema. As output, the user obtains a conceptual schema that includes the combination of the elements of the selected schema with the elements of interest gathered by our methodology.

Specific Input

The input for the filtering request $\mathcal{F}_4$ contains the same elements as indicated in Sect. 6.2. In addition, we present here a detailed description of the particularities of such elements for the specific input of $\mathcal{F}_4$.

- **Large conceptual schema:** the source schema $\mathcal{CS} = \langle \mathcal{SS}, \mathcal{BS} \rangle$ where $\mathcal{SS} = \langle \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ is the structural subschema, and $\mathcal{BS} = \langle \mathcal{E}_b, \mathcal{R}_b, \mathcal{G}_b, \mathcal{C}_b \rangle$ is the behavioral subschema. The amount of knowledge represented in $\mathcal{CS}$ is large and makes it very difficult to manually extract fragments of interest to a user.
- **Focus Set:** it works as the conceptual schema viewpoint of the user. Therefore, a focus set is an initial point that should be extended with more knowledge. Formally, the focus set $\mathcal{FS}$ contains a small conceptual schema $\mathcal{CS}' = \langle \mathcal{SS}', \mathcal{BS}' \rangle$ of interest to the user where

$\mathcal{SS}' = \langle \mathcal{E}', \mathcal{R}', \mathcal{T}', \mathcal{G}', \mathcal{C}', \mathcal{D}' \rangle$ is the structural subschema and $\mathcal{BS}' = \langle \mathcal{E}'_b, \mathcal{R}'_b, \mathcal{G}'_b, \mathcal{C}'_b \rangle$ is the behavioral subschema, from which the user wants to know more about. Note that the size of the focus set is smaller than the size of the large schema. It is mandatory for the user to select a non-empty focus set ($\mathcal{FS} \neq \emptyset$).

- **Size threshold:** it denotes the maximum expected number $\mathcal{K}$ of entity and event types in the output. Note that $|\mathcal{E}_{\mathcal{FS}} \cup \mathcal{E}_{b\mathcal{FS}}| \leq \mathcal{K} \leq |\mathcal{E} \cup \mathcal{E}_b|$, where $\mathcal{E}_{\mathcal{FS}}$ is the set that includes the entity types in the focus set $\mathcal{FS}$ and $\mathcal{E}_{b\mathcal{FS}}$ the set that includes the event types in the focus set. Note that $\mathcal{E}_{\mathcal{FS}}$ and $\mathcal{E}_{b\mathcal{FS}}$ also contains those entity and event types that participate in relationship types or are referenced by schema rules from the focus set.

- **Rejection Set:** the set $\mathcal{RS}$ with entity types of $\mathcal{E}$ and event types of $\mathcal{E}_b$ that the user does not want to obtain in the output. Note that it is disjoint with the focus set ($\mathcal{RS} \cap \mathcal{FS} = \emptyset$). By default, the rejection set is empty ($|\mathcal{RS}| \geq 0$).

- **Importance method:** the algorithm $\mathcal{I}$ to compute the importance of entity types from $\mathcal{E}$ and event types from $\mathcal{E}_b$. By default $\mathcal{I} = I_{SM}$, the Simple Method described in Chap. 4.

Specific Output

The output of this filtering request is a filtered conceptual schema $\mathcal{CS}_{\mathcal{F}} = \langle \mathcal{SS}_{\mathcal{F}}, \mathcal{BS}_{\mathcal{F}} \rangle$, where $\mathcal{SS}_{\mathcal{F}} = \langle \mathcal{E}_{\mathcal{F}}, \mathcal{R}_{\mathcal{F}}, \mathcal{T}_{\mathcal{F}}, \mathcal{G}_{\mathcal{F}}, \mathcal{C}_{\mathcal{F}}, \mathcal{D}_{\mathcal{F}} \rangle$ is the structural subschema, and $\mathcal{BS}_{\mathcal{F}} = \langle \mathcal{E}_{b\mathcal{F}}, \mathcal{R}_{b\mathcal{F}}, \mathcal{G}_{b\mathcal{F}}, \mathcal{C}_{b\mathcal{F}} \rangle$ is the behavioral subschema. The specific constraints that $\mathcal{CS}_{\mathcal{F}}$ must satisfy are described as follows:

[C1] $\mathcal{E}_{\mathcal{F}}$ contains the entity types $\mathcal{E}'$ from the schema of focus.

[C2] $\mathcal{E}_{\mathcal{F}}$ does not contain the entity types from the rejection set $\mathcal{RS}$.

[C3] $\mathcal{R}_{\mathcal{F}}$ contains the relationship types $\mathcal{R}'$ from the schema of focus.

[C4] $\mathcal{T}_{\mathcal{F}}$ contains the data types $\mathcal{T}'$ from the schema of focus.

[C5] $\mathcal{G}_{\mathcal{F}}$ contains the generalization relationships $\mathcal{G}'$ from the schema of focus.

[C6] $\mathcal{C}_{\mathcal{F}}$ contains the integrity constraints $\mathcal{C}'$ from the schema of focus.

[C7] $\mathcal{D}_{\mathcal{F}}$ contains the derivation rules $\mathcal{D}'$ from the schema of focus.

[C8] $\mathcal{E}_{b\mathcal{F}}$ contains the event types $\mathcal{E}'_b$ from the schema of focus.

[C9] $\mathcal{E}_{b\mathcal{F}}$ does not contain the event types from the rejection set $\mathcal{RS}$.

[C10] $\mathcal{R}_{b\mathcal{F}}$ contains the relationship types $\mathcal{R}'_b$ from the schema of focus.

[C11] $\mathcal{G}_{b\mathcal{F}}$ contains the generalization relationships $\mathcal{G}'_b$ from the schema of focus.

[C12] $\mathcal{C}_{b\mathcal{F}}$ contains the integrity constraints $\mathcal{C}'_b$ from the schema of focus.

[C13] If r is a relationship type of $\mathcal{R}$ and its participant entity types belong to $\mathcal{E}_{\mathcal{F}}$, or are ascendants of entity types of $\mathcal{E}_{\mathcal{F}}$ (in which case a projection is needed), then r is included in $\mathcal{R}_{\mathcal{F}}$. The same behavior applies to relationships of $\mathcal{R}_b$ to be included in $\mathcal{R}_{b\mathcal{F}}$.

[C14] $\mathcal{G}_{\mathcal{F}}$ contains direct generalization relationships between entity types of $\mathcal{E}_{\mathcal{F}}$.

[C15] $\mathcal{G}_{b\mathcal{F}}$ contains direct generalization relationships between event types of $\mathcal{E}_{b\mathcal{F}}$.

[C16] If c is an integrity constraint or derivation rule of $\mathcal{C}$, $\mathcal{D}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_{\mathcal{F}}$ and any of the schema elements referenced by c belong to $\mathcal{CS}_{\mathcal{F}}$, then c is included in $\mathcal{C}_{\mathcal{F}}$, $\mathcal{D}_{\mathcal{F}}$ or $\mathcal{C}_{b\mathcal{F}}$.

[C17] If d is a data type of $\mathcal{T}$ and it is used by attributes of entity types of $\mathcal{E}_{\mathcal{F}}$, event types of $\mathcal{E}_{b\mathcal{F}}$, or schema rules of $\mathcal{C}_{\mathcal{F}}$, $\mathcal{C}_{b\mathcal{F}}$ or $\mathcal{D}_{\mathcal{F}}$, then d is included in $\mathcal{T}_{\mathcal{F}}$ of $\mathcal{CS}_{\mathcal{F}}$.

[C18] If e_1 and e_2 are entity types of $\mathcal{E}_{\mathcal{F}}$ and does not exist a direct generalization between them in $\mathcal{G}_{\mathcal{F}}$ nor a path of direct generalizations of $\mathcal{G}_{\mathcal{F}}$ traversing entity types e_i of $\mathcal{E}_{\mathcal{F}}$, but both e_1 and e_2 belong to different levels of the same hierarchy in $\mathcal{G}$ of $\mathcal{CS}$, a direct generalization g' is included between e_1 and e_2 in $\mathcal{G}_{\mathcal{F}}$ but is marked as derived. The same behavior applies to pairs of event types of $\mathcal{E}_{b\mathcal{F}}$.

[C19] If c is a constraint of $\mathcal{C}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_{\mathcal{F}}$ and references schema elements out of $\mathcal{CS}_{\mathcal{F}}$ only the header of such constraint is included in $\mathcal{C}_{\mathcal{F}}$ or $\mathcal{C}_{b\mathcal{F}}$. If d is a derivation rule of $\mathcal{D}$ whose context belongs to $\mathcal{CS}_{\mathcal{F}}$ and references schema elements out of $\mathcal{CS}_{\mathcal{F}}$, the context element of the rule is marked as materialized in $\mathcal{CS}_{\mathcal{F}}$ and d is not included in $\mathcal{D}_{\mathcal{F}}$.

Filtering Stages

The filtering request for a conceptual schema follows the specific methodology presented in Sect. 5.4 of Ch. 5. In the following, we present a brief summary of the different stages and steps (see Fig. 6.21) that belong to this filtering request.

- **Stage 1: Metrics.** The first step in this stage (see Activity 1.1 in Fig. 6.21) processes the input of the filtering request. It creates some auxiliary sets to gather the entity, relationship, and event types from the small schema that conforms the focus set. In this particular case, the auxiliary set $\mathcal{E}_{\mathcal{FS}}$ contains the union between the entity types from the focus set, and the entity types that are participants of the relationship types from the focus set. Also, the auxiliary set $\mathcal{E}_{b\mathcal{FS}}$ contains the union between the event types from the focus set, and the event types that are participants of the relationship types from the focus set. Finally, the auxiliary set $\mathcal{R}_{\mathcal{FS}}$ contains the relationship types from the focus set. These three auxiliary sets are the basis of the filtering request in order to construct a new filtered schema with the elements of the small schema of input and additional knowledge of interest. The rest of the steps follow the same indications presented in Sect. 5.4.1 of Ch. 5. The method computes the importance, closeness, and interest metrics, and sorts the entity and event types in a ranking.

- **Stage 2: Entity and Event Types.** This stage follows the same steps as presented in Sect. 5.4.2 of Ch. 5. The method selects the top entity and event types from the interest ranking taking into account the size threshold of the input and includes them in the resulting filtered conceptual schema. The entity and event types from the small schema of focus are also included in such schema.
- **Stage 3: Relationship Types.** This stage follows the same steps as presented in Sect. 5.4.3 of Ch. 5. The method classifies the relationship types according to their participants as referentially-complete or referentially-partial relationship types. Note that those relationship types in the small schema of focus are always referentially-complete since their participants are members on the set $\mathcal{E}_{\mathcal{FS}}$ of entity types or on the set $\mathcal{E}_{b\mathcal{FS}}$ of event types from the focus set. Then, the method projects and selects the final relationship types that are part of the resulting filtered conceptual schema.
- **Stage 4: Generalizations.** This stage follows the same steps as presented in Sect. 5.4.4 of Ch. 5. The method processes the direct and creates indirect generalizations to construct the hierarchies of the resulting filtered conceptual schema. The generalizations in the small schema of focus are also included in the filtered schema because their general and specific participants are all entity and event types already included in that schema.
- **Stage 5: Schema Rules.** This stage follows the same steps as presented in Sect. 5.4.5 of Ch. 5. The method selects the schema rules defined in the context of elements from the filtered schema and processes the referentially-incomplete ones in order to construct the rules of the resulting filtered conceptual schema. The schema rules from the small schema of focus are also included in the filtered schema.
- **Stage 6: Data Types.** This stage follows the same steps as presented in Sect. 5.4.6 of Ch. 5. The method includes in the filtered schema those data types referenced or used by other elements within such schema.
- **Stage 7: Presentation.** This stage follows the same steps as presented in Sect. 5.4.7 of Ch. 5. The method presents the filtered schema to the user. The elements to highlight are the elements of the small schema from the focus set $\mathcal{FS}$.

Method Correctness

The proposed activities in the stages of the filtering request transform the input into a valid output in the form of a filtered conceptual schema that satisfies a set constraints over such schema. In the following we verify the correctness of the method according to those constraints and the activities that satisfy each of them:

- Constraint [**C1**] is satisfied by the activity 2.2 of the filtering request.
- Constraint [**C2**] is satisfied by the activity 2.2 of the filtering request.
- Constraint [**C3**] is satisfied by the activities 3.1, 3.2, and 3.4 of the filtering request.

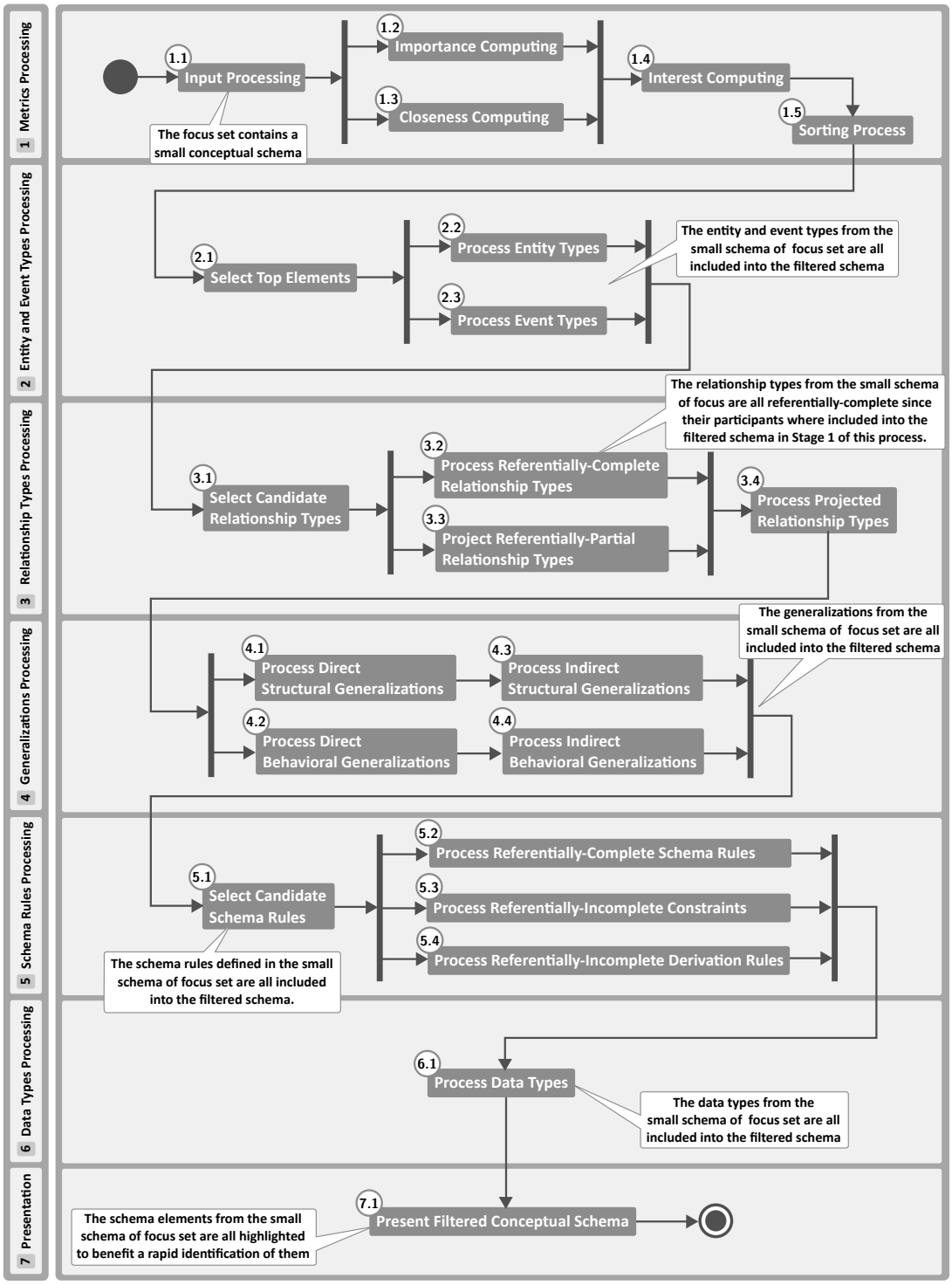


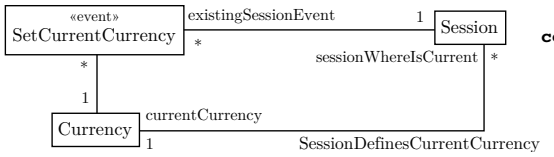
Figure 6.21. Activity diagram for the filtering request $\mathcal{F}_4$.

- Constraint [**C4**] is satisfied by the activity 6.1 of the filtering request.
- Constraint [**C5**] is satisfied by the activity 4.1 of the filtering request.
- Constraint [**C6**] is satisfied by the activities 5.1, 5.2, and 5.3 of the filtering request.
- Constraint [**C7**] is satisfied by the activities 5.1, 5.2, and 5.4 of the filtering request.
- Constraint [**C8**] is satisfied by the activity 2.3 of the filtering request.
- Constraint [**C9**] is satisfied by the activity 2.3 of the filtering request.
- Constraint [**C10**] is satisfied by the activities 3.1, 3.2, and 3.4 of the filtering request.
- Constraint [**C11**] is satisfied by the activity 4.2 of the filtering request.
- Constraint [**C12**] is satisfied by the activities 5.1, 5.2, 5.3, and 5.4 of the filtering request.
- Constraint [**C13**] is satisfied by the activities 3.1, 3.2, 3.3, and 3.4 of the filtering request.
- Constraint [**C14**] is satisfied by the activity 4.1 of the filtering request.
- Constraint [**C15**] is satisfied by the activity 4.2 of the filtering request.
- Constraint [**C16**] is satisfied by the activities 5.1 and 5.2 of the filtering request.
- Constraint [**C17**] is satisfied by the activity 6.1 of the filtering request.
- Constraint [**C18**] is satisfied by the activities 4.3 and 4.4 of the filtering request.
- Constraint [**C19**] is satisfied by the activities 5.1, 5.3 and 5.4 of the filtering request.

The activities 1.1, 1.2, 1.3, 1.5, and 1.5 of the filtering request process the input and deal with the computation of relevance metrics to filter the large schema. The activity 2.1 selects the additional entity and event types to complete the knowledge from the focus set. The activity 7.1 presents the resulting filtered conceptual schema to the user.

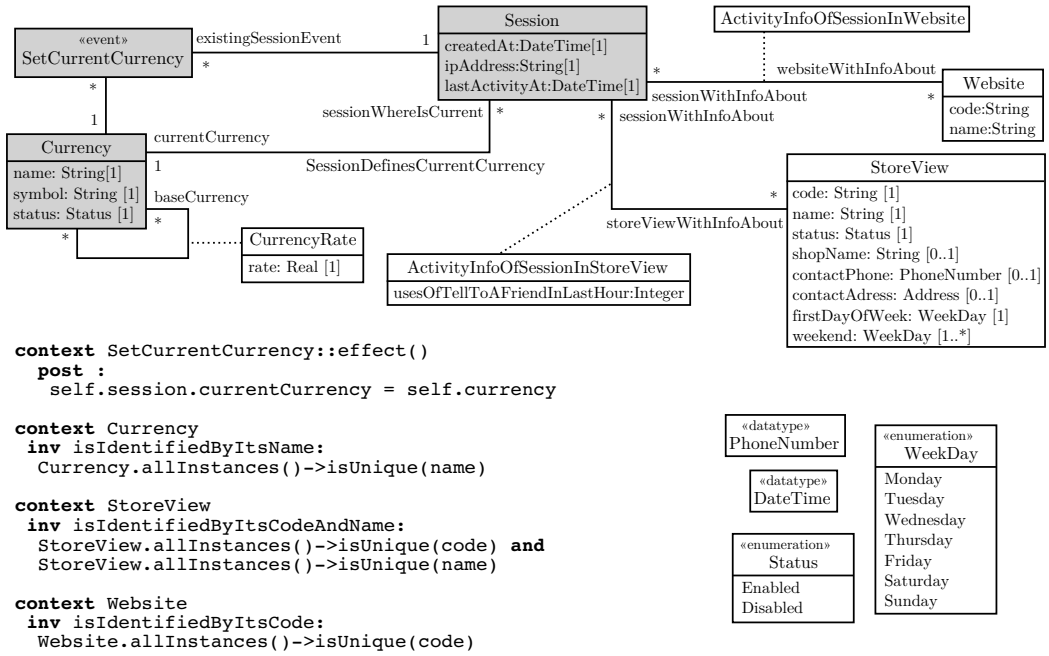
Example of $\mathcal{F}_4$

The filtering scenario proposed in this example describes a user of the conceptual schema of the Magento e-commerce system that wants to know the schema elements of interest with respect to a small fragment of the large schema. Concretely, the user focuses in the conceptual schema shown in Fig. 6.22. Our method takes this schema as the input of the filtering request.




```
context SetCurrentCurrency::effect()
  post :
   self.session.currentCurrency = self.currency
```

Figure 6.22. Example of input for the filtering request $\mathcal{F}_4$.




```
context SetCurrentCurrency::effect()
  post :
   self.session.currentCurrency = self.currency

context Currency
 inv isIdentifiedByItsName:
  Currency.allInstances()->isUnique(name)

context StoreView
 inv isIdentifiedByItsCodeAndName:
  StoreView.allInstances()->isUnique(code) and
  StoreView.allInstances()->isUnique(name)

context Website
 inv isIdentifiedByItsCode:
  Website.allInstances()->isUnique(code)
```


Figure 6.23. Filtered schema for the schema fragment of Fig 6.22.

As a result, the user obtains the filtered schema depicted in Fig. 6.23. The filtering request completes the information from the input schema with additional elements of interest from the large schema. Concretely, the schema of input contains the minimum set of schema elements that are referenced by the effect postcondition of the event type SetCurrentCurrency. This event sets the currency that must be used for the current session in Magento.

The resulting schema our method obtains includes the knowledge about websites and store views, and their relation with the Session entity type. Furthermore, it also shows a reflexive binary relationship type between two instances of Currency, with an association class that indicates the CurrencyRate. This filtered schema provides useful feedback for a modeler that has to modify the definition of the postcondition of SetCurrentCurrency.

6.3.5 $\mathcal{F}_5$: Filtering Request for Context Behavior of Entity Types

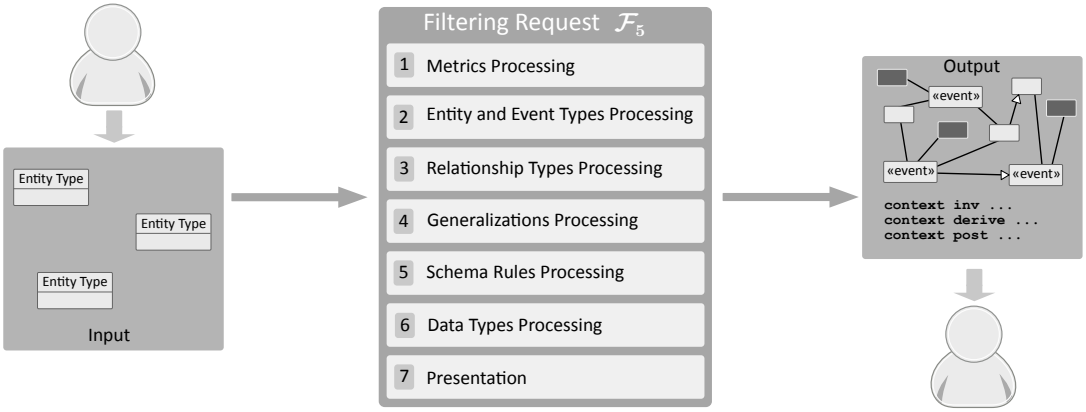


Figure 6.24. $\mathcal{F}_5$: Filtering request for context behavior of entity types.

Application Scenario

The user focuses on a set of entity types from a large conceptual schema. The user is aware of those entity types or she has accessed them via previous filtering requests. The information need consists in obtaining the event types from the schema with relation to the entity types in the user focus. The method obtains those event types of interest to the user according to the initial selection and the characteristics represented in the large schema. As output, the user obtains a small-sized filtered conceptual schema that includes the combination of the selected entity types with the event types of interest gathered by our methodology.

Specific Input

The input for the filtering request $\mathcal{F}_5$ contains the same elements as indicated in Sect. 6.2. In addition, we present here a detailed description of the particularities of such elements for the specific input of $\mathcal{F}_5$.

- **Large conceptual schema:** the source schema $\mathcal{CS} = \langle \mathcal{SS}, \mathcal{BS} \rangle$ where $\mathcal{SS} = \langle \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ is the structural subschema, and $\mathcal{BS} = \langle \mathcal{E}_b, \mathcal{R}_b, \mathcal{G}_b, \mathcal{C}_b \rangle$ is the behavioral subschema. The amount of knowledge represented in $\mathcal{CS}$ is large and makes it very difficult to manually extract fragments of interest to a user.
- **Focus Set:** it works as the conceptual schema viewpoint of the user. Therefore, a focus set is an initial point that should be extended with more knowledge. Formally, the focus set $\mathcal{FS}$ contains a small subset of the entity types of $\mathcal{E}$ from which the user wants to know more about with respect to their relation with the event types from the behavioral subschema $\mathcal{BS}$. Note that the size of the focus set is reduced with respect to the amount

of entity and relationship types from the large schema ($|\mathcal{FS}| \ll |\mathcal{E}|$). Also, it is mandatory for the user to select a non-empty focus set ($\mathcal{FS} \neq \emptyset$).

- **Size threshold:** it denotes the maximum expected number $\mathcal{K}$ of entity and event types in the output. Note that $|\mathcal{E}_{\mathcal{FS}}| \leq \mathcal{K} \leq |\mathcal{E} \cup \mathcal{E}_b|$, where $\mathcal{E}_{\mathcal{FS}}$ is the set that includes the entity types in the focus set $\mathcal{FS}$.
- **Rejection Set:** the set $\mathcal{RS}$ with event types of $\mathcal{E}_b$ that the user does not want to obtain in the output. By default, the rejection set is empty ($|\mathcal{RS}| \geq 0$).
- **Importance method:** the algorithm $\mathcal{I}$ to compute the importance of entity types from $\mathcal{E}$ and event types from $\mathcal{E}_b$. By default $\mathcal{I} = I_{SM}$, the Simple Method described in Chap. 4.

Specific Output

The output of this filtering request is a filtered conceptual schema $\mathcal{CS}_{\mathcal{F}} = \langle \mathcal{SS}_{\mathcal{F}}, \mathcal{BS}_{\mathcal{F}} \rangle$, where $\mathcal{SS}_{\mathcal{F}} = \langle \mathcal{E}_{\mathcal{F}}, \mathcal{R}_{\mathcal{F}}, \mathcal{T}_{\mathcal{F}}, \mathcal{G}_{\mathcal{F}}, \mathcal{C}_{\mathcal{F}}, \mathcal{D}_{\mathcal{F}} \rangle$ is the structural subschema, and $\mathcal{BS}_{\mathcal{F}} = \langle \mathcal{E}_{b\mathcal{F}}, \mathcal{R}_{b\mathcal{F}}, \mathcal{G}_{b\mathcal{F}}, \mathcal{C}_{b\mathcal{F}} \rangle$ is the behavioral subschema. The specific constraints that $\mathcal{CS}_{\mathcal{F}}$ must satisfy are described as follows:

[C1] $\mathcal{E}_{\mathcal{F}}$ contains the entity types $\mathcal{E}_{\mathcal{FS}}$ from the focus set $\mathcal{FS}$.

[C2] $\mathcal{E}_{b\mathcal{F}}$ does not contain the event types from the rejection set $\mathcal{RS}$.

[C3] $\mathcal{G}_{\mathcal{F}}$ contains direct generalization relationships between entity types of $\mathcal{E}_{\mathcal{F}}$.

[C4] $\mathcal{G}_{b\mathcal{F}}$ contains direct generalization relationships between event types of $\mathcal{E}_{b\mathcal{F}}$.

[C5] If c is an integrity constraint or derivation rule of $\mathcal{C}$, $\mathcal{D}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_{\mathcal{F}}$ and any of the schema elements referenced by c belong to $\mathcal{CS}_{\mathcal{F}}$, then c is included in $\mathcal{C}_{\mathcal{F}}$, $\mathcal{D}_{\mathcal{F}}$ or $\mathcal{C}_{b\mathcal{F}}$.

[C6] If r is a relationship type of $\mathcal{R}$ and its participant entity types belong to $\mathcal{E}_{\mathcal{F}}$, or are ascendants of entity types of $\mathcal{E}_{\mathcal{F}}$ (in which case a projection is needed), then r is included in $\mathcal{R}_{\mathcal{F}}$. The same behavior applies to relationship types of $\mathcal{R}_b$ to be included in $\mathcal{R}_{b\mathcal{F}}$.

[C7] If d is a data type of $\mathcal{T}$ and it is used by attributes of entity types of $\mathcal{E}_{\mathcal{F}}$, event types of $\mathcal{E}_{b\mathcal{F}}$, or schema rules of $\mathcal{C}_{\mathcal{F}}$, $\mathcal{C}_{b\mathcal{F}}$ or $\mathcal{D}_{\mathcal{F}}$, then d is included in $\mathcal{T}_{\mathcal{F}}$ of $\mathcal{CS}_{\mathcal{F}}$.

[C8] If e_1 and e_2 are entity types of $\mathcal{E}_{\mathcal{F}}$ and does not exist a direct generalization between them in $\mathcal{G}_{\mathcal{F}}$ nor a path of direct generalizations of $\mathcal{G}_{\mathcal{F}}$ traversing entity types e_i of $\mathcal{E}_{\mathcal{F}}$, but both e_1 and e_2 belong to different levels of the same hierarchy in $\mathcal{G}$ of $\mathcal{CS}$, a direct generalization g' is included between e_1 and e_2 in $\mathcal{G}_{\mathcal{F}}$ but is marked as derived. The same behavior applies to pairs of event types of $\mathcal{E}_{b\mathcal{F}}$.

[C9] If c is a constraint of $\mathcal{C}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_{\mathcal{F}}$ and references schema elements out of $\mathcal{CS}_{\mathcal{F}}$ only the header of such constraint is included in $\mathcal{C}_{\mathcal{F}}$ or $\mathcal{C}_{b\mathcal{F}}$. If d is a derivation rule of $\mathcal{D}$ whose context belongs to $\mathcal{CS}_{\mathcal{F}}$ and references schema elements out of $\mathcal{CS}_{\mathcal{F}}$, the context element of the rule is marked as materialized in $\mathcal{CS}_{\mathcal{F}}$ and d is not included in $\mathcal{D}_{\mathcal{F}}$.

Filtering Stages

The filtering request for context behavior of entity types follows the specific methodology presented in Sect. 5.4 of Ch. 5. In the following, we present a brief summary of the different stages and steps (see Fig. 6.25) that belong to this filtering request.

- **Stage 1: Metrics.** The first step in this stage (see Activity 1.1 in Fig. 6.25) processes the input of the filtering request. It creates an auxiliary set to gather the entity types from the focus set. The rest of the steps follow the same indications presented in Sect. 5.4.1 of Ch. 5. The method computes the importance, closeness, and interest metrics, and sorts the entity and event types in a ranking.

- **Stage 2: Entity and Event Types.** This stage follows the same steps as presented in Sect. 5.4.2 of Ch. 5. However, for this particular filtering request the method selects only event types from the interest ranking of top entity and event types taking into account the size threshold of the input, and includes them in the resulting filtered conceptual schema. Concretely, we select only those event types that are connected with at least one of the entity types from the focus set. Therefore, the filtered conceptual schema will contain the entity types from the focus set and the top event types of interest with connection to the focus set.

- **Stage 3: Relationship Types.** This stage follows the same steps as presented in Sect. 5.4.3 of Ch. 5. The method classifies the relationship types according to their participants as referentially-complete or referentially-partial relationship types. Then, the method projects and selects the final relationship types that are part of the resulting filtered conceptual schema.

- **Stage 4: Generalizations.** This stage follows the same steps as presented in Sect. 5.4.4 of Ch. 5. The method processes the direct and creates indirect generalizations to construct the hierarchies of the resulting filtered conceptual schema.

- **Stage 5: Schema Rules.** This stage follows the same steps as presented in Sect. 5.4.5 of Ch. 5. The method selects the schema rules defined in the context of elements from the filtered schema and processes the referentially-incomplete ones in order to construct the rules of the resulting filtered conceptual schema.

- **Stage 6: Data Types.** This stage follows the same steps as presented in Sect. 5.4.6 of Ch. 5. The method includes in the filtered schema those data types referenced or used by other elements within such schema.

- **Stage 7: Presentation.** This stage follows the same steps as presented in Sect. 5.4.7 of Ch. 5. The method presents the filtered schema to the user. The elements to highlight are the entity types from the focus set $\mathcal{FS}$.

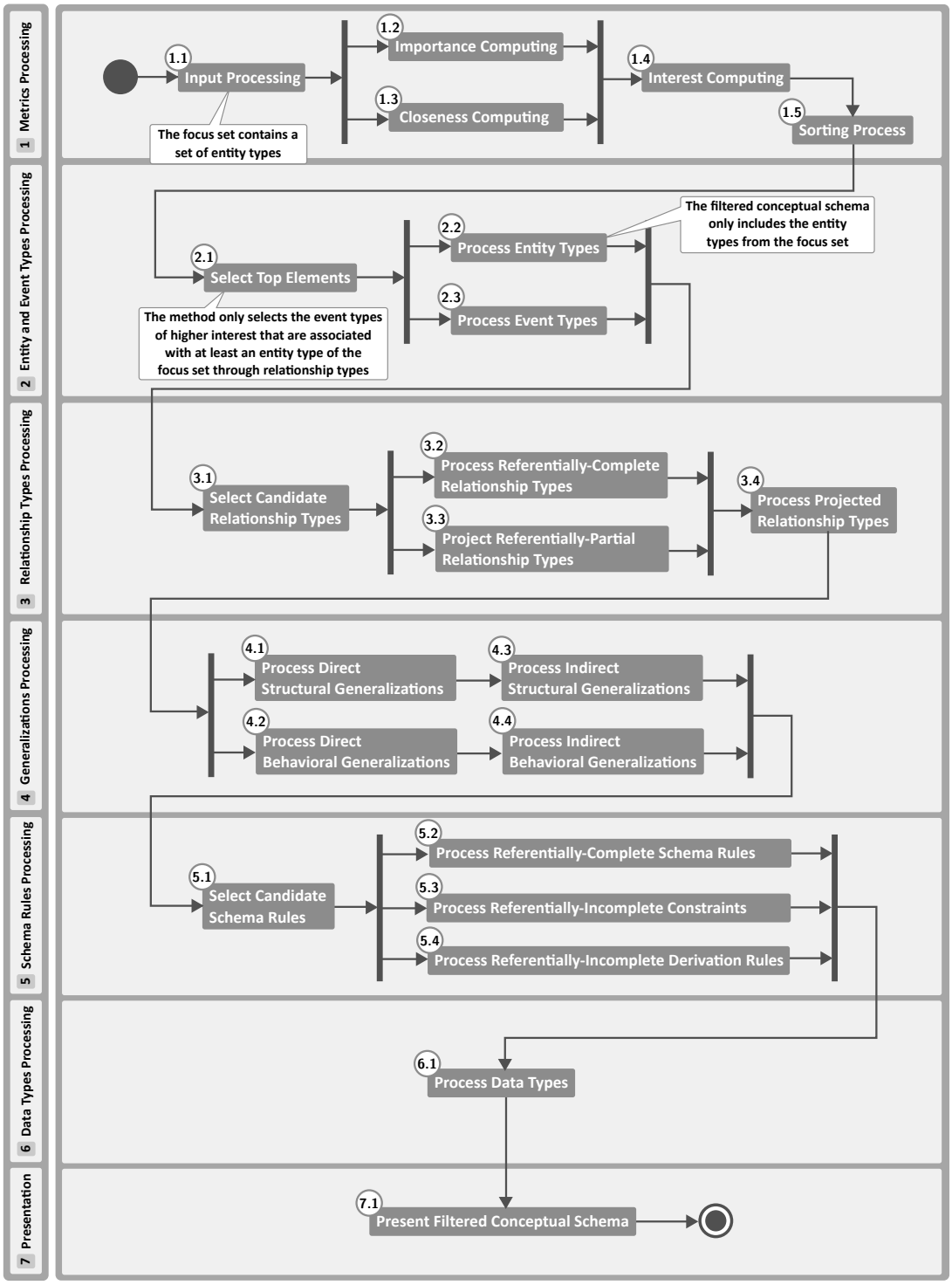


Figure 6.25. Activity diagram for the filtering request $\mathcal{F}_5$.

Method Correctness

The proposed activities in the stages of the filtering request transform the input into a valid output in the form of a filtered conceptual schema that satisfies a set constraints over such schema. In the following we verify the correctness of the method according to those constraints and the activities that satisfy each of them:

- Constraint [**C1**] is satisfied by the activity 2.2 of the filtering request.
- Constraint [**C2**] is satisfied by the activity 2.3 of the filtering request.
- Constraint [**C3**] is satisfied by the activity 4.1 of the filtering request.
- Constraint [**C4**] is satisfied by the activity 4.2 of the filtering request.
- Constraint [**C5**] is satisfied by the activities 5.1 and 5.2 of the filtering request.
- Constraint [**C6**] is satisfied by the activities 3.1, 3.2, 3.3, and 3.4 of the filtering request.
- Constraint [**C7**] is satisfied by the activity 6.1 of the filtering request.
- Constraint [**C8**] is satisfied by the activities 4.3 and 4.4 of the filtering request.
- Constraint [**C9**] is satisfied by the activities 5.1, 5.3, and 5.4 of the filtering request.

The activities 1.1, 1.2, 1.3, 1.5, and 1.5 of the filtering request process the input and deal with the computation of relevance metrics to filter the large schema. The activity 2.1 selects the additional event types to complete the knowledge from the focus set. The activity 7.1 presents the resulting filtered conceptual schema to the user.

Example of $\mathcal{F}_5$

The filtering scenario proposed in this example describes a user of the conceptual schema of the Magento e-commerce system that wants to obtain the related event types of interest with respect to a set of entity types. Concretely, the user focuses in the entity type Product, which is one of the more relevant entity types in the Magento system. The user constructs an input to the filtering request as follows:

```
CS = Magento
FS = {Product}
RS = ∅
K = 4
I = Simple Method Extended
```

Figure 6.26 and Fig. 6.27 depict the resulting filtered schema our method obtains. The entity type Product is related to the event types EditProduct and NewProduct. These are the two event types of greater interest with respect to products in the Magento e-commerce system.

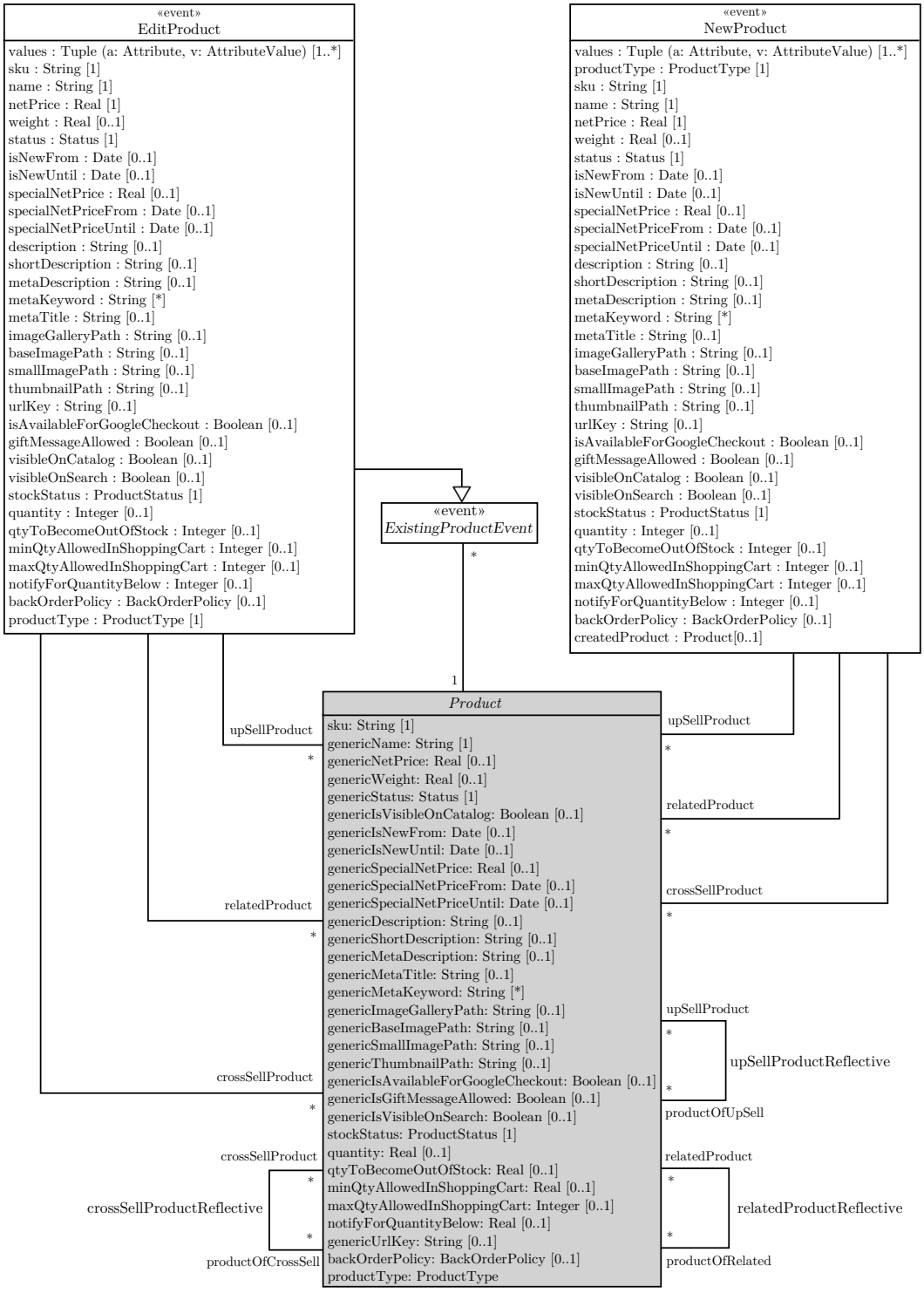
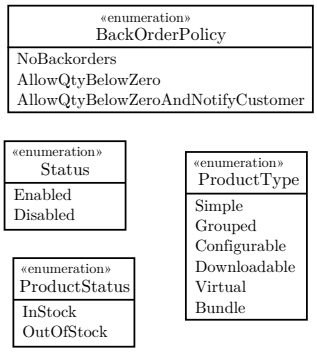


Figure 6.26. Filtered schema for the entity type Product (I).




```
context EditProduct
 inv stockOptionsAreRatedOnlyForTheCorrectProductTypes:
  ( self.productType = ProductType::Grouped or
    self.productType = ProductType::Configurable or
    self.productType = ProductType::Bundle )
  implies (
    self.quantity.oclIsUndefined() and
    self.qtyToBecomeOutOfStock.oclIsUndefined() and
    self.minQtyAllowedInShoppingCart.oclIsUndefined() and
    self.maxQtyAllowedInShoppingCart.oclIsUndefined() and
    self.notifyForQuantityBelow.oclIsUndefined() and
    self.backOrderPolicy.oclIsUndefined()
  )

context NewProduct
 inv stockOptionsAreRatedOnlyForTheCorrectProductTypes:
  ( self.productType = ProductType::Grouped or
    self.productType = ProductType::Configurable or
    self.productType = ProductType::Bundle )
  implies (
    self.quantity.oclIsUndefined() and
    self.qtyToBecomeOutOfStock.oclIsUndefined() and
    self.minQtyAllowedInShoppingCart.oclIsUndefined() and
    self.maxQtyAllowedInShoppingCart.oclIsUndefined() and
    self.notifyForQuantityBelow.oclIsUndefined() and
    self.backOrderPolicy.oclIsUndefined()
  )

context NewProduct
 inv skuDoesNotExist:
  not Product.allInstances()->exists(p | p.sku = self.sku )

context Product
  inv isIdentifiedBySku:
   Product.allInstances()->isUnique(sku)es())
```

Figure 6.27. Filtered schema for the entity type Product (II).

The effects of these events allow to create and modify the products in the store. The EditProduct event is a descendant of ExistingProductEvent. Note that both EditProduct and NewProduct contain attributes to specify the characteristics that are needed to modify an instance of Product, or the values for creating a new one.

6.3.6 $\mathcal{F}_6$: Filtering Request for Contextualized Types

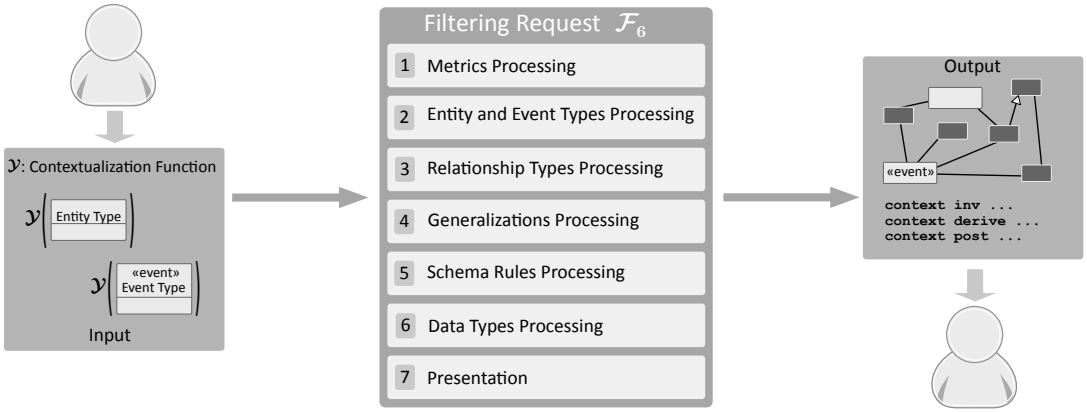


Figure 6.28. $\mathcal{F}_6$: Filtering request for contextualized types.

Application Scenario

The user focuses on a set of entity and event types from a large conceptual schema. The user is aware of those types or she has accessed them via previous filtering requests. The user contextualize the entity and event types by means of a defined function to reduce or limit the characteristics defined over such type taking into account the user interest. The information need consists in obtaining more knowledge from the schema with relation to the entity and event types in the user focus and the applied contextualization function. The method obtains the elements of interest to the user according to the initial selection and the characteristics represented in the large schema. As output, the user obtains a small-sized filtered conceptual schema that includes the combination of the selected entity and event types with the elements of interest gathered by our methodology.

Specific Input

The input for the filtering request $\mathcal{F}_6$ contains the same elements as indicated in Sect. 6.2 plus a contextualization function. In addition, we present here a detailed description of the particularities of such elements for the specific input of $\mathcal{F}_6$.

- **Large conceptual schema:** the source schema $\mathcal{CS} = \langle \mathcal{SS}, \mathcal{BS} \rangle$ where $\mathcal{SS} = \langle \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ is the structural subschema, and $\mathcal{BS} = \langle \mathcal{E}_b, \mathcal{R}_b, \mathcal{G}_b, \mathcal{C}_b \rangle$ is the behavioral subschema. The amount of knowledge represented in $\mathcal{CS}$ is large and makes it very difficult to manually extract fragments of interest to a user.
- **Focus Set:** it works as the conceptual schema viewpoint of the user. Therefore, a focus set is an initial point that should be extended with more knowledge. Formally, the focus set $\mathcal{FS}$ contains a small subset of entity types of $\mathcal{E}$ and event types of $\mathcal{E}_b$ from which the

user wants to know more about. Note that the size of the focus set is reduced with respect to the amount of entity and event types from the large schema ($|\mathcal{FS}| \ll |\mathcal{E}_b \cup \mathcal{E}|$). Also, it is mandatory for the user to select a non-empty focus set ($\mathcal{FS} \neq \emptyset$).

- **Size threshold:** it denotes the maximum expected number $\mathcal{K}$ of entity and event types in the output. Note that $|\mathcal{FS}| \leq \mathcal{K} \leq |\mathcal{E} \cup \mathcal{E}_b|$.
- **Rejection Set:** the set $\mathcal{RS}$ with entity types of $\mathcal{E}$ and event types of $\mathcal{E}_b$ that the user does not want to obtain in the output. Note that it is disjoint with the focus set ($\mathcal{RS} \cap \mathcal{FS} = \emptyset$). By default, the rejection set is empty ($|\mathcal{RS}| \geq 0$).
- **Importance method:** the algorithm $\mathcal{I}$ to compute the importance of entity types from $\mathcal{E}$ and event types from $\mathcal{E}_b$. By default $\mathcal{I} = I_{SM}$, the Simple Method described in Chap. 4.
- **Contextualization function:** the function $\mathcal{Y}$ applied to an entity or event type e, $\mathcal{Y}(e)$, produces a contextualized entity or event type e'. This function allows to:
  - select a default literal value for attributes of e in e' whose type is an enumeration.
  - delete a defined attribute of e in e', whenever the minimum multiplicity of such attribute equals to zero.
  - redefine the multiplicity of an attribute of e in e'.
  - delete a relationship type between e and another schema element in e' whenever the minimum multiplicity in the opposite relationship end of e equals zero.
  - redefine the multiplicity of the opposite relationship end of e in e', in a relationship type between e and another schema element.

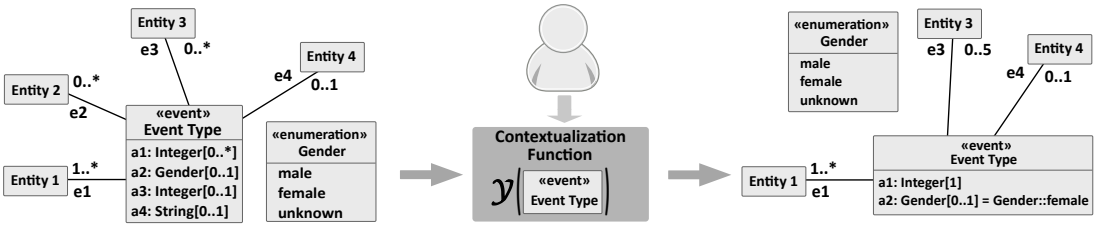


Figure 6.29. Example of application of a contextualization function $\mathcal{Y}$.

Figure 6.29 presents an example of application of the contextualization function to an event type. On the left there is an event type with four attributes and the same number of relationship types where it participates. A user may define its own contextualization function to obtain the contextualized event type depicted on the right side of Fig. 6.29. The multiplicity of the attribute `a1` is redefined to `1..1` in the contextualized event type. The attribute `a2`, whose type is the `Gender` enumeration has the default value `female`. Also, the attributes `a3` and `a4`, whose minimum multiplicity is zero, do not appear on the contextualized event type on the right. The relationship type between the event type and the entity type `Entity 2`, whose minimum multiplicity on the side of `Entity 2` is zero, does not appear on the contextualized event type on the right. Finally, the multiplicity of the relationship end of `Entity 3` is redefined from `0..*` to `0..5` in the contextualized event type. Note that to redefine a multiplicity `n..m` to `x..y`, then $\texttt{n} \leq \texttt{x} \leq \texttt{y} \leq \texttt{m}$.

Specific Output

The output of this filtering request is a filtered conceptual schema $\mathcal{CS}_{\mathcal{F}} = \langle \mathcal{SS}_{\mathcal{F}}, \mathcal{BS}_{\mathcal{F}} \rangle$, where $\mathcal{SS}_{\mathcal{F}} = \langle \mathcal{E}_{\mathcal{F}}, \mathcal{R}_{\mathcal{F}}, \mathcal{T}_{\mathcal{F}}, \mathcal{G}_{\mathcal{F}}, \mathcal{C}_{\mathcal{F}}, \mathcal{D}_{\mathcal{F}} \rangle$ is the structural subschema, and $\mathcal{BS}_{\mathcal{F}} = \langle \mathcal{E}_{b\mathcal{F}}, \mathcal{R}_{b\mathcal{F}}, \mathcal{G}_{b\mathcal{F}}, \mathcal{C}_{b\mathcal{F}} \rangle$ is the behavioral subschema. The specific constraints that $\mathcal{CS}_{\mathcal{F}}$ must satisfy are described as follows:

[C1] $\mathcal{E}_{\mathcal{F}}$ contains the entity types $\mathcal{E}_{\mathcal{FS}}$ from the focus set $\mathcal{FS}$.

[C2] $\mathcal{E}_{\mathcal{F}}$ does not contain the entity types from the rejection set $\mathcal{RS}$.

[C3] $\mathcal{E}_{b\mathcal{F}}$ contains the event types $\mathcal{E}_{b\mathcal{FS}}$ from the focus set $\mathcal{FS}$.

[C4] $\mathcal{E}_{b\mathcal{F}}$ does not contain the event types from the rejection set $\mathcal{RS}$.

[C5] $\mathcal{G}_{b\mathcal{F}}$ contains direct generalization relationships between event types of $\mathcal{E}_{b\mathcal{F}}$.

[C6] $\mathcal{G}_{\mathcal{F}}$ contains direct generalization relationships between entity types of $\mathcal{E}_{\mathcal{F}}$.

[C7] If c is an integrity constraint or derivation rule of $\mathcal{C}$, $\mathcal{D}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_{\mathcal{F}}$ and any of the schema elements referenced by c belong to $\mathcal{CS}_{\mathcal{F}}$, then c is included in $\mathcal{C}_{\mathcal{F}}$, $\mathcal{D}_{\mathcal{F}}$ or $\mathcal{C}_{b\mathcal{F}}$.

[C8] If an attribute a of an entity or event type e whose type is an enumeration is contextualized by applying the function $\mathcal{Y}$, which selects a default literal value for a, then the attribute appears with its default value in $\mathcal{CS}_{\mathcal{F}}$. If the multiplicity of the attribute is redefined by applying the function $\mathcal{Y}$, then the redefined multiplicity appears as the multiplicity of the attribute in $\mathcal{CS}_{\mathcal{F}}$. If the attribute is deleted in the contextualization, it does not appear in $\mathcal{CS}_{\mathcal{F}}$ nor counts in the importance method $\mathcal{I}$.

[C9] If a relationship type r in which an entity or event type e participates is deleted by applying the contextualization function $\mathcal{Y}$, then r does not appear in $\mathcal{R}_{b\mathcal{F}}$. Also, r does not count in the importance method $\mathcal{I}$ nor in the closeness computation.

[C10] If r is a relationship type of $\mathcal{R}$ and its participant entity types belong to $\mathcal{E}_{\mathcal{F}}$, or are ascendants of entity types of $\mathcal{E}_{\mathcal{F}}$ (in which case a projection is needed), then r is included in $\mathcal{R}_{\mathcal{F}}$. The same behavior applies to relationship types of $\mathcal{R}_b$ to be included in $\mathcal{R}_{b\mathcal{F}}$. If the multiplicity of a relationship end is redefined by applying the function $\mathcal{Y}$, then the redefined multiplicity appears as the multiplicity of the relationship end in $\mathcal{CS}_{\mathcal{F}}$.

[C11] If d is a data type of $\mathcal{T}$ and it is used by attributes of entity types of $\mathcal{E}_{\mathcal{F}}$, event types of $\mathcal{E}_{b\mathcal{F}}$, or schema rules of $\mathcal{C}_{\mathcal{F}}$, $\mathcal{C}_{b\mathcal{F}}$ or $\mathcal{D}_{\mathcal{F}}$, then d is included in $\mathcal{T}_{\mathcal{F}}$ of $\mathcal{CS}_{\mathcal{F}}$.

[C12] If e_1 and e_2 are entity types of $\mathcal{E}_{\mathcal{F}}$ and does not exist a direct generalization between them in $\mathcal{G}_{\mathcal{F}}$ nor a path of direct generalizations of $\mathcal{G}_{\mathcal{F}}$ traversing entity types e_i of $\mathcal{E}_{\mathcal{F}}$, but both e_1 and e_2 belong to different levels of the same hierarchy in $\mathcal{G}$ of $\mathcal{CS}$, a direct generalization g' is included between e_1 and e_2 in $\mathcal{G}_{\mathcal{F}}$ but is marked as derived. The same behavior applies to pairs of event types of $\mathcal{E}_{b\mathcal{F}}$.

[C13] If c is a constraint of $\mathcal{C}$ or $\mathcal{C}_b$ defined in the context of an schema element of $\mathcal{CS}_\mathcal{F}$ and references schema elements out of $\mathcal{CS}_\mathcal{F}$ only the header of such constraint is included in $\mathcal{C}_\mathcal{F}$ or $\mathcal{C}_{b\mathcal{F}}$. If d is a derivation rule of $\mathcal{D}$ whose context belongs to $\mathcal{CS}_\mathcal{F}$ and references schema elements out of $\mathcal{CS}_\mathcal{F}$, the context element of the rule is marked as materialized in $\mathcal{CS}_\mathcal{F}$ and d is not included in $\mathcal{D}_\mathcal{F}$.

Filtering Stages

The filtering request for contextualized types follows the specific methodology presented in Sect. 5.4 of Ch. 5. In the following, we present a brief summary of the different stages and steps (see Fig. 6.30) that belong to this filtering request.

- **Stage 1: Metrics.** The first step in this stage (see Activity 1.1 in Fig. 6.30) processes the input of the filtering request. It creates some auxiliary sets to gather the entity and event types from the focus set. In this particular case, the auxiliary sets $\mathcal{E}_{\mathcal{FS}}$ and $\mathcal{E}_{b\mathcal{FS}}$ contain the entity and event types from the focus set, respectively. Also, the set $\mathcal{R}_{\mathcal{FS}}$ that contains relationship types from the focus set is empty since this filtering request focus on the entity and event types to construct the input. The second step (see Activity 1.1.1 in Fig. 6.30) applies the contextualization function to the entity and event types of the focus set. The deleted attributes and relationship types from the contextualization do not participate in the importance and closeness computing. The rest of the steps follow the same indications presented in Sect. 5.4.1 of Ch. 5. The method computes the importance, closeness, and interest metrics, and sorts the entity and event types in a ranking.
- **Stage 2: Entity and Event Types.** This stage follows the same steps as presented in Sect. 5.4.2 of Ch. 5. The method selects the top entity and event types from the interest ranking taking into account the size threshold of the input and includes them in the resulting filtered conceptual schema. The entity and event types of the focus set are all included in the filtered schema. The contextualized attributes with an enumeration type are included in the filtered schema with the default value selected by the contextualization function.
- **Stage 3: Relationship Types.** This stage follows the same steps as presented in Sect. 5.4.3 of Ch. 5. The method classifies the relationship types according to their participants as referentially-complete or referentially-partial relationship types. Then, the method projects and selects the final relationship types that are part of the resulting filtered conceptual schema. The relationship types deleted in the contextualization are not selected as candidate relationships and, therefore, they will not appear in the filtered schema.
- **Stage 4: Generalizations.** This stage follows the same steps as presented in Sect. 5.4.4 of Ch. 5. The method processes the direct and creates indirect generalizations to construct the hierarchies of the resulting filtered conceptual schema.
- **Stage 5: Schema Rules.** This stage follows the same steps as presented in Sect. 5.4.5 of Ch. 5. The method selects the schema rules defined in the context of elements from

the filtered schema and processes the referentially-incomplete ones in order to construct the rules of the resulting filtered conceptual schema.

- **Stage 6: Data Types.** This stage follows the same steps as presented in Sect. 5.4.6 of Ch. 5. The method includes in the filtered schema those data types referenced or used by other elements within such schema.
- **Stage 7: Presentation.** This stage follows the same steps as presented in Sect. 5.4.7 of Ch. 5. The method presents the filtered schema to the user. The elements to highlight are the entity and event types from the focus set $\mathcal{FS}$.

Method Correctness

The proposed activities in the stages of the filtering request transform the input into a valid output in the form of a filtered conceptual schema that satisfies a set constraints over such schema. In the following we verify the correctness of the method according to those constraints and the activities that satisfy each of them:

- Constraint **[C1]** is satisfied by the activity 2.2 of the filtering request.
- Constraint **[C2]** is satisfied by the activity 2.2 of the filtering request.
- Constraint **[C3]** is satisfied by the activity 2.3 of the filtering request.
- Constraint **[C4]** is satisfied by the activity 2.3 of the filtering request.
- Constraint **[C5]** is satisfied by the activity 4.2 of the filtering request.
- Constraint **[C6]** is satisfied by the activity 4.1 of the filtering request.
- Constraint **[C7]** is satisfied by the activities 5.1 and 5.2 of the filtering request.
- Constraint **[C8]** is satisfied by the activities 1.1.1 and 1.2 of the filtering request.
- Constraint **[C9]** is satisfied by the activities 1.1.1, 1.2, 1.3, and 3.1 of the filtering request.
- Constraint **[C10]** is satisfied by the activities 3.1, 3.2, 3.3, and 3.4 of the filtering request.
- Constraint **[C11]** is satisfied by the activity 6.1 of the filtering request.
- Constraint **[C12]** is satisfied by the activities 4.3 and 4.4 of filtering request method.
- Constraint **[C13]** is satisfied by the activities 5.1, 5.3, and 5.4 of the filtering request.

The activities 1.1, 1.2, 1.3, 1.5, and 1.5 of the filtering request process the input and deal with the computation of relevance metrics to filter the large schema. The activity 1.1.1 applies the contextualization function $\mathcal{Y}$ to the entity and event types of the focus set $\mathcal{FS}$. The activity 2.1 selects the additional entity and event types to complete the knowledge from the focus set. The activity 7.1 presents the resulting filtered conceptual schema to the user.

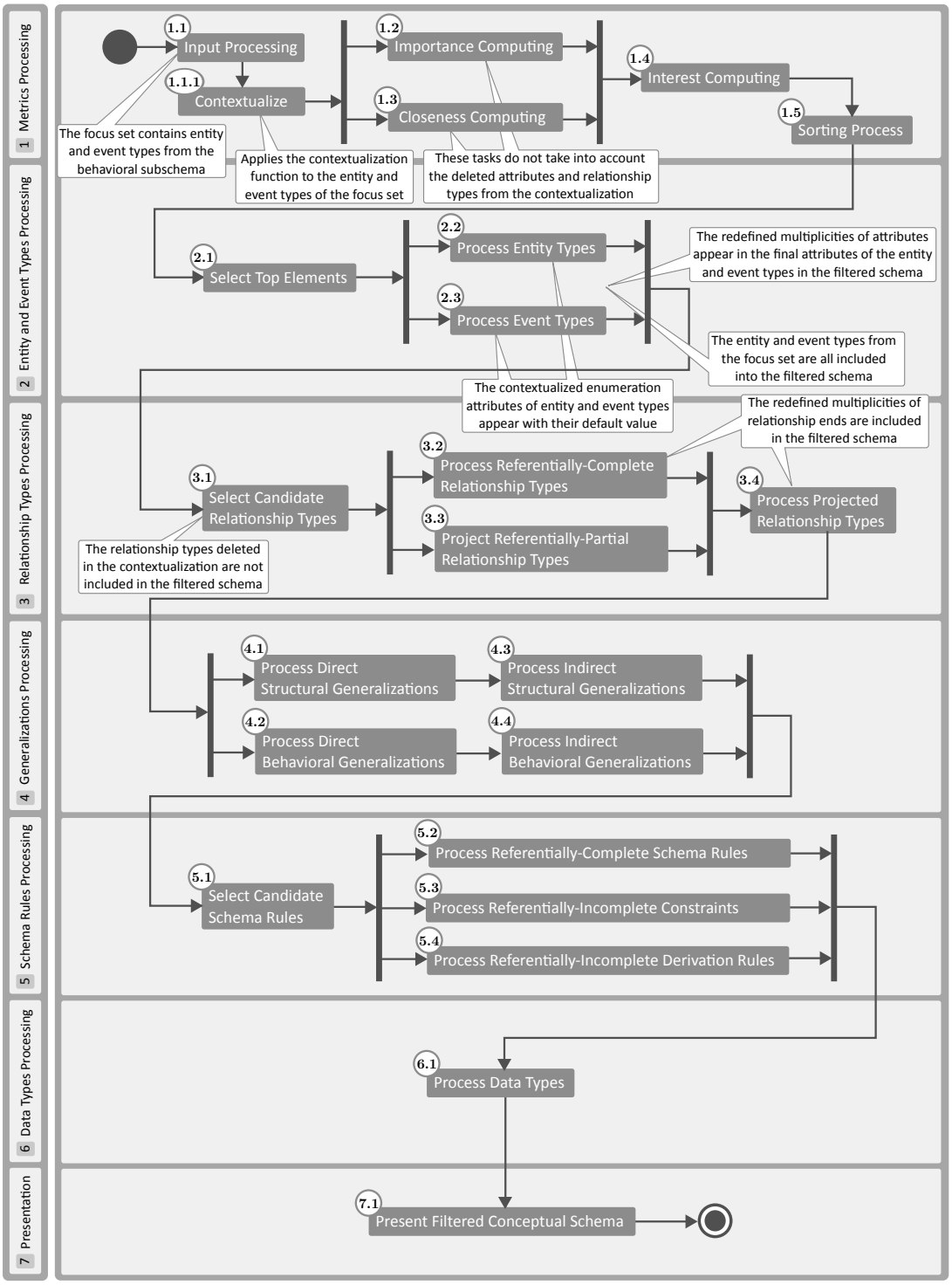


Figure 6.30. Activity diagram for the filtering request $\mathcal{F}_6$.

Example of $\mathcal{F}_6$

The filtering scenario proposed in this example describes a user of the conceptual schema of the Magento e-commerce system that wants to contextualize the characteristics of interest with respect to a set of entity and event types. Concretely, the user focuses in the event type NewProduct, which is one of the more relevant event types in the Magento system. The user constructs an input to the filtering request as follows:

```
CS = Magento
FS = {NewProduct}
RS = ∅
K = 3
Y = { NewProduct::status = Status::Disabled,
      NewProduct::stockStatus = ProductStatus::InStock,
      NewProduct::minQtyAllowedInShoppingCart -> 0..0,
      NewProduct::maxQtyAllowedInShoppingCart -> 0..0,
      NewProduct::notifyForQuantityBelow -> 0..0,
      NewProduct::backOrderPolicy -> 0..0,
      NewProduct::qtyToBecomeOutOfStock -> 0..0,
      NewProduct::createdProduct -> 0..0,
      NewProduct::metaKeyword -> 1..10,
      NewProduct::description -> 1..1,
      NewProduct::upsellProduct -> 0..0,
      NewProduct::crossSellProduct -> 0..0,
      NewProduct::relatedProduct -> 1..* }
I = Simple Method Extended
```

Figure 6.31 and Fig. 6.32 present the filtered schema our method obtains with respect to the previous input. Note that the contextualization function $\mathcal{Y}$ contains several statements to change the original characteristics of the event type NewProduct. The first two statements set a default value for the enumeration attributes status and stockStatus. Therefore, the new instance of Product that the event NewProduct creates will have a disabled status and its default stock status will indicate that the product is in stock.

In addition to it, the last three statements of the contextualization function redefine the multiplicity of the relationship types between NewProduct and Product. In the example, the multiplicities of the association ends with rolenames upsellProduct and crossSellProduct are set to 0..0, which indicates that the relationship type will not be part of the resulting schema. Note that Fig. 6.31 does not include the aforementioned relationships. On the contrary, the multiplicity of the association end relatedProduct is redefined to 1..* in order to indicate that at least a product must be related to the new instance of product that NewProduct constructs. The other statements redefine the multiplicity of some attributes of the NewProduct event. Concretely, the attributes that are set to a 0..0 multiplicity do not appear in the filtered schema. Furthermore, the multiplicity of the attribute metaKeyword is set to 1..10, to indicate that the new product must contain at least one value, and at most 10 –originally, it was set to 0..*. And the optional attribute description (0..1) is mandatory (1..1) after the contextualization.

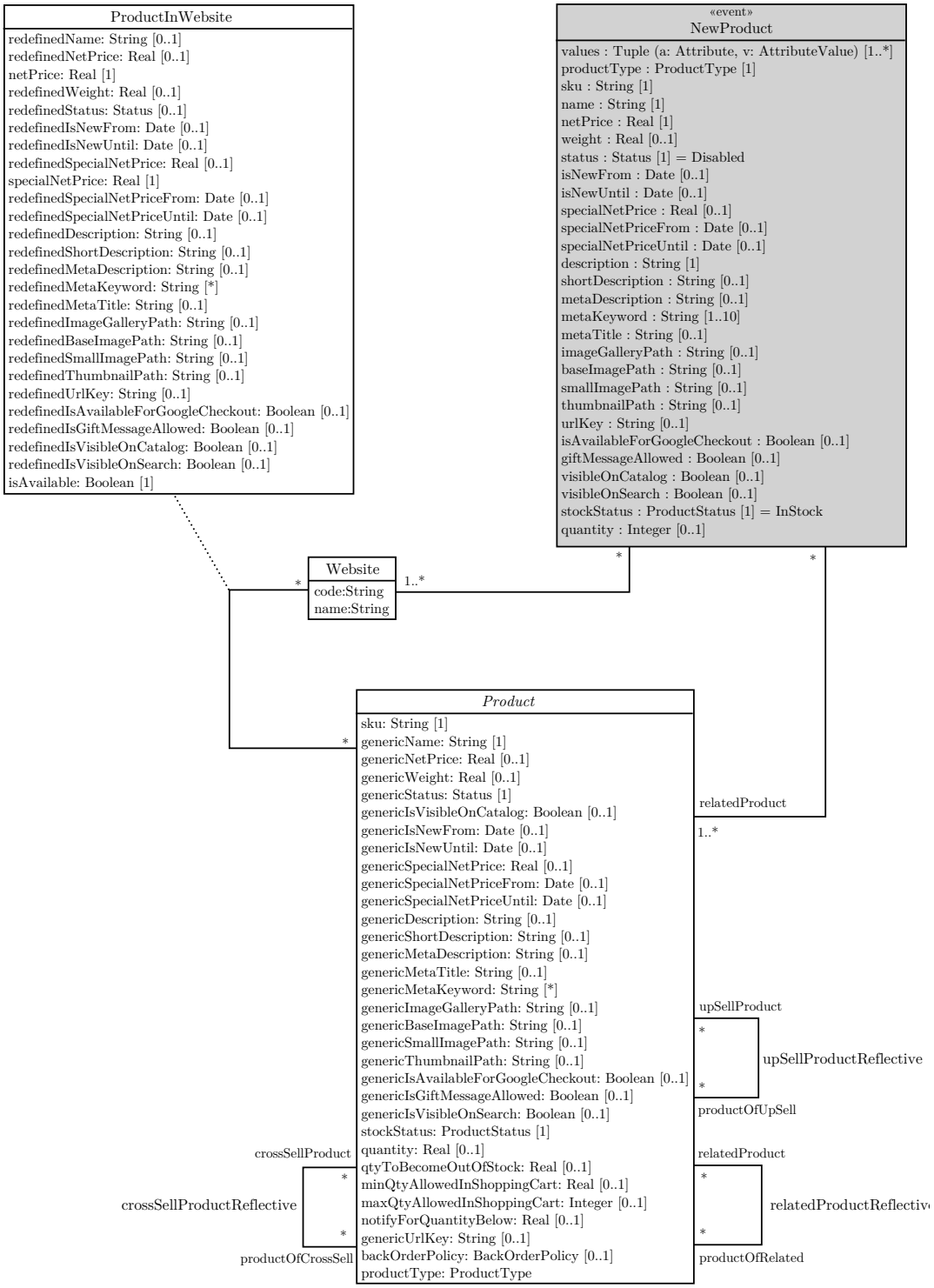
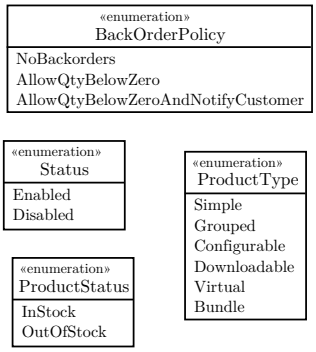


Figure 6.31. Filtered schema for the filtering request $\mathcal{F}_6$ (I).




```
context NewProduct
 inv skuDoesNotExist:
  not Product.allInstances()->exists(p | p.sku = self.sku )

context Product
  inv isIdentifiedBySku:
   Product.allInstances()->isUnique(sku)es())

context Website
  inv isIdentifiedByItsCode:
   Website.allInstances()->isUnique(code)

context Product
 inv isSpecifiedInAllWebsites:
  self.website-> ncludesAll( Website.allInstances() )
```

Figure 6.32. Filtered schema for the filtering request $\mathcal{F}_6$ (II).

6.4 Combination of Filtering Requests

As aforementioned, our filtering methodology satisfies a user's information need over a large conceptual schema providing a catalog of filtering requests. The interaction between the user and our proposed mechanism to gather the knowledge of interest is an iterative process that must be applied as many times as required.

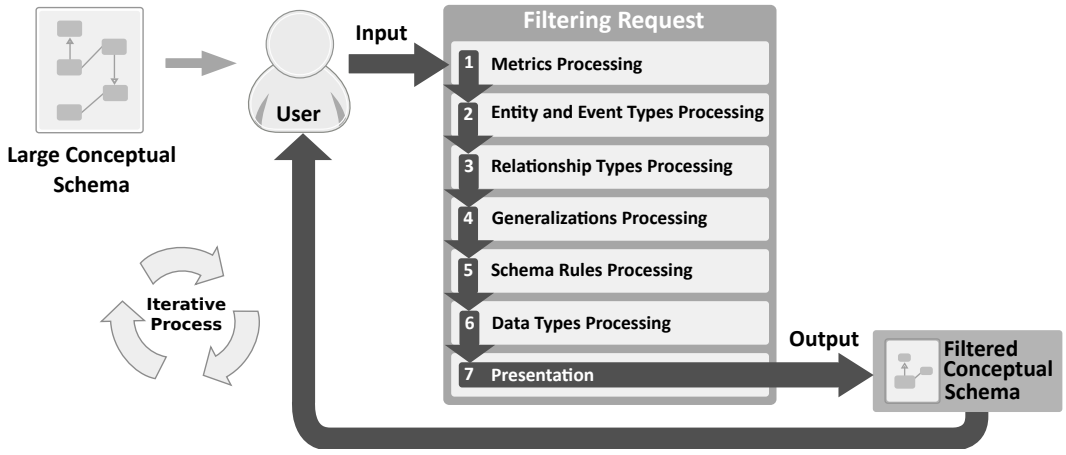


Figure 6.33. Representation of the iterative process to extract knowledge from a large conceptual schema.

Figure 6.33 presents this iterative process. The user wants to know more about a fragment or subset of the large conceptual schema, and therefore uses the filtering request from the catalog that covers a specific need of information. The filtering request follows the seven stages as presented in previous sections in order to construct a filtered conceptual schema centered on the knowledge in the user focus. The user obtains the filtered schema and may continue with a new iteration of the process using a different filtering request if necessary. Finally, the process ends when the user believes that she has obtained enough knowledge from the large schema to cover the specific information need. By applying the filtering requests from our proposed catalog, the user reduces the effort of traversing the whole large schema, making a more-efficient exploration based on the relevance of the knowledge within it and saving time.

A key point about the catalog of filtering requests presented in Sect. 6.3 relates to the sequence of application to combine filtering requests in the scope of the iterative process of Fig. 6.33. The usage of each of the six filtering requests requires a specific input in order to be able to apply the stages that conform the request. To this end, we say that two filtering requests can be directly combined whenever the output that results from the application of the first filtering request always contains a valid input for the second filtering request. On the other hand, two filtering requests can be conditionally combined whenever the output that results from the application of the first filtering request may in some cases contain a valid input for the second filtering request.

Figure 6.34 describes the existing relations between the filtering requests of the catalog and their intended interaction to satisfy a user's information need over a large conceptual schema

by combining them. As an example, we can start by selecting a set of entity and relationship types from the large schema to conform the focus set for the filtering request $\mathcal{F}_1$. The filtered conceptual schema that we obtain from the application of $\mathcal{F}_1$ contains the selected entity and relationship types of focus plus a set of additional schema elements of high interest. Then, we can start a new iteration with any of the six filtering requests as indicated in the graph of Fig. 6.34. However, we can only combine $\mathcal{F}_1$ with the filtering requests $\mathcal{F}_2$ or $\mathcal{F}_6$. In the same way, we can combine $\mathcal{F}_1$ with $\mathcal{F}_3$ only if the filtered conceptual schema that we obtained from $\mathcal{F}_1$ contains schema rules. On the other hand, we can always combine $\mathcal{F}_1$ with $\mathcal{F}_4$ or $\mathcal{F}_5$ since the filtered conceptual schema that we obtained from $\mathcal{F}_1$ is a small conceptual schema that can be the input for $\mathcal{F}_4$, and contains the entity types of focus that can be the required input for $\mathcal{F}_5$. This way, we can combine sequences of filtering requests and, therefore, navigate through filtered conceptual schemas of small size instead of traversing the whole large conceptual schema.

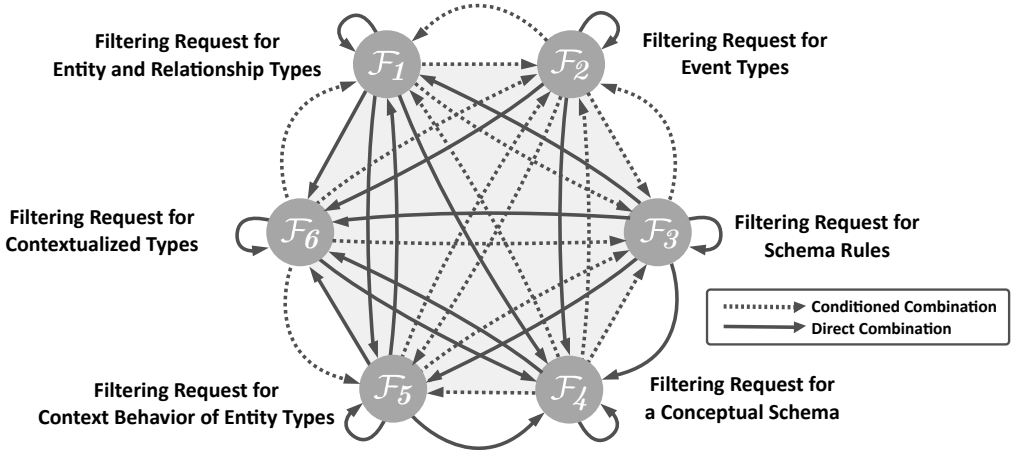


Figure 6.34. Combination of filtered requests.

6.5 Summary

In Ch. 5, a general methodology to extract knowledge of interest from a large conceptual schema has been presented. Such methodology provides users with an automatic process that requires of an specific filtering input in order to obtain a reduced filtered conceptual schema with the knowledge of high relevance according to the user preferences. Chapter 4, introduces several techniques to compute the relevance of the elements of a large conceptual schema, which are used in Ch. 5 to support the filtering methodology.

Chapter 6 deals with the particular requirements of users to apply the proposed filtering method under several filtering circumstances such as the need to specify different kinds of inputs that will produce many differences in the resulting filtered conceptual schema, according to the initial input specification and the user expertise when working with large conceptual schemas. By means of these requirements, we have undertaken the description and formalization of a catalog of specific filtering requests that instantiate the previous filtering methodology and make use of the relevance metrics to filter a large schema in different situations according to

their particular input. Our main contribution here is the definition of six filtering requests in Sect. 6.3, their specific input and output, and the different activities that process the input and a large schema in order to obtain a small-sized filtered conceptual schema as output. Section 6.3.1 introduces the first filtering request of the catalog, which deals with a filtering interaction centered on the entity and relationship types from a large schema. Section 6.3.2 presents the second request where the user focuses on a set of integrity constraints and derivation rules from the large schema and wants to obtain more knowledge from the schema with relation to the schema rules in the user focus. Section 6.3.3 describes the third specific request that focus on a set of event types from the large conceptual schema to construct the core of the resulting filtered schema. The fourth filtering request takes as main input a small conceptual schema, which may be a filtered schema of a previous request, and produces a new filtered schema from it as detailed in Sect. 6.3.4. Section 6.3.5 deals with the fifth request that focuses on a set of entity types from a large conceptual schema in order to obtain the event types from the schema with relation to the entity types in the user focus. The last specific filtering request is analyzed in Sect. 6.3.6 and focuses on a set of entity and event types that the user contextualize by means of a defined contextualization function to reduce or limit the characteristics defined over such types taking into account the user interest. As output, the user obtains a small-sized filtered conceptual schema that includes the combination of the selected entity and event types with the elements of interest gathered by our methodology taking into account the user contextualization. Finally, Sect. 6.4 presents the sequence of application to combine filtering requests in the scope of the iterative process to filter a large schema.

In Ch. 7 we describe the application of our filtering methodology to a set of four real-case large conceptual schemas from several domains. In Ch. 8, we present a web-based filtering engine that implements the filtering requests.

The difference between theory and practice is that in theory, there is no difference between theory and practice."

Richard Moore

7

Application of the Filtering Methodology

The relevance-computing methods provided us a way to discover the core concepts described in a large conceptual schema. With that in mind, we extended these methods and created a more accurate one in order to obtain the most interesting concepts of the schema with respect to an specific user request of knowledge. Then, we adapted the general filtering method to several filtering scenarios and constructed a catalog of filtering requests that provides different ways to explore a large conceptual schema. In this chapter, we apply these filtering requests to a set of real-world large conceptual schemas in different scenarios, and analyze the results from this experimentation.

The chapter starts with an overview of three different case studies in Sect. 7.1. Section 7.2 introduces the first case study that deals with two large conceptual schemas from the e-commerce domain. Section 7.3 describes the second case study, which explores the conceptual schema of a car rental system. The last case study processes the conceptual schema of the formal specification of the UML metaschema in Sect. 7.4. For each case study, we detail the characteristics of the large conceptual schemas where we apply the filtering methodology explained in Ch. 5. Also, we show a common filtering application scenario in order to describe the benefits our methodology provides by using the filtering requests from the catalog presented in Ch. 6. Consequently, Sect. 7.5 experimentally evaluates the effectiveness and efficiency of our filtering approach with respect to the conceptual schemas from the previous case studies and reports the main results from the experimentation. Finally, Sect. 7.6 summarizes the chapter.

7.1 Case Studies Overview

Case study research consists of a detailed investigation of phenomena, within their context. The aim is to provide an analysis of the context and processes which illuminate the theoretical issues being studied. The phenomenon is not isolated from its context (as in, say, laboratory research) but is of interest precisely because the aim is to understand how behavior and/or processes are influenced by an influence context. The case study research design is also useful for testing whether scientific theories and models actually work in the real world [56].

In the design-science paradigm, knowledge and understanding of a problem domain and its solution are achieved in the building and application of the designed artifact [59]. As aforementioned in Ch. 1, design science addresses research through the building and evaluation of artifacts designed to meet the identified business need. Research assessment via the justify/evaluate activities can result in the identification of weaknesses in the theory or artifact and the need to refine and reassess.

Evaluation is a crucial component of the research process. The evaluation phase provides essential feedback to the construction phase as to the quality of the design process and the design product under development. A design artifact is complete and effective when it satisfies the requirements and constraints of the problem it was meant to solve. In this chapter, we report the application of the filtering methodology in the following experimental case studies:

- A comparison between two large conceptual schemas representing the knowledge of two frameworks for e-commerce applications, the osCommerce and the Magento.
- An exploration of the behavioral components of a large conceptual schema for a car rental system, the EU-Rent, in order to understand the specific functionality of the system.
- An exploration of the metaschema of the Unified Modeling Language, which is guided by the contents of the UML Superstructure specification document.

The overall goals of these case studies are: (1) analyzing the viability of using our filtering methodology to compare the specification of several concepts from the e-commerce domain in two different conceptual schemas, (2) characterizing the functionalities of a real conceptual schema through the application of filtering request to its behavioral subschema, (3) identifying filtering patterns in a formal exploration following the contents of a normative specification of a model-based standard, and (4) using the lessons learned to improve and refine the filtering method and filtering requests.

These case studies are representative of different kinds of user interaction with large conceptual schemas and they correspond to different filtering scenarios. We covered the whole set of filtering requests from the catalog of Ch. 6 and we obtained information about the utility of the filtering method when applied to specific situations.

In the following, we briefly present the characteristics of the large conceptual schemas that participate on the case studies.

7.1.1 The osCommerce e-Commerce System

Figure 7.1. Screenshot of the osCommerce system.

osCommerce is an online shop e-commerce solution that offers a wide range of out-of-the-box features that allows online stores to be setup fairly quickly with ease, and is available for free as an open source based solution released under the GNU General Public License. osCommerce was started in March 2000 and has since matured to a solution that is currently powering 12,666 registered live shops around the world[1].

Today, osCommerce has been taken to the next level, moving towards an e-commerce framework solution that not only remains easy to setup and maintain, but also making it easier for store administrators to present their stores to their customers with their own unique requirements. The success of osCommerce is secured by a great and active community where members help one another out and participate in development issues reflecting upon the current state of the project. Figure 7.1 presents a screenshot of the osCommerce system.

The conceptual schema of the osCommerce is specified using standard UML and OCL, and was obtained as a result of a reverse engineering process from the source code of the osCommerce system [118]. The structural subschema of the osCommerce contains 84 entity types and 209 attributes. On the other hand, the behavioral subschema contains 262 event types and 220 pre- and postconditions specifying the effect of the events. Furthermore, the conceptual schema connects its elements through a set of 183 relationship types and 393 generalization relationships, and includes 204 integrity constraints and 17 enumeration types. Figure 3.1 graphically shows this conceptual schema.

[1] `www.oscommerce.com`

7.1.2 The Magento e-Commerce System

Figure 7.2. Screenshot of the Magento system.

Magento is a full-fledged, open source e-commerce platform aimed at web site designers, developers, and business owners who are looking for a complete e-commerce web site solution. The Magento system provides the scalability, flexibility and features for business growth. Magento enables feature-rich e-commerce platforms that offer merchants complete flexibility and control over the presentation, content, and functionality of their online channel. It was launched on March 2008, and with more than 750,000 downloads, Magento is the fastest-growing open source e-commerce solution[2].

The conceptual schema of Magento was already used in this thesis to illustrate the seven stages of the filtering method introduced in Ch. 5, and the six filtering requests from the catalog of Ch. 6. It is specified using standard UML and OCL, and was obtained as a result of a reverse engineering process from the source code of the Magento system [94]. The structural subschema of the Magento contains 218 entity types and 983 attributes. On the other hand, the behavioral subschema contains 187 event types and 69 pre- and postconditions specifying the effect of the events. Furthermore, the conceptual schema connects its elements through a set of 319 relationship types and 165 generalization relationships, and includes 386 integrity constraints, 185 derivation rules, 15 data types, and 46 enumeration types. Figure 5.7 graphically shows this conceptual schema.

[2] `www.magentocommerce.com`

7.1.3 The EU-Rent Car Rental System

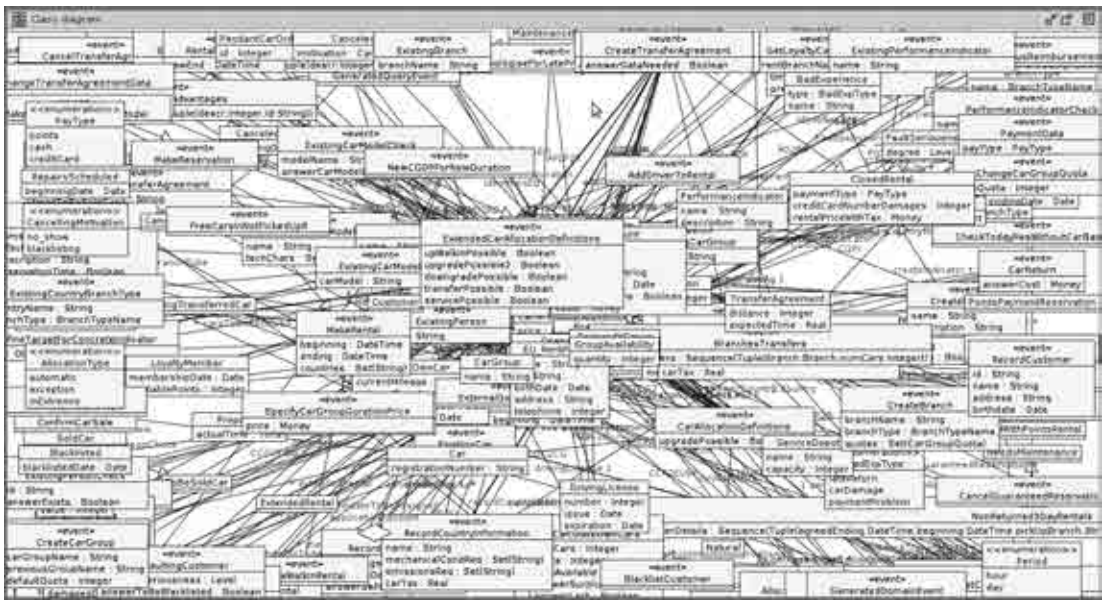


Figure 7.3. Conceptual schema of the EU-Rent car rental system.

EU-Rent is a case study being promoted as a basis for demonstration of product capabilities which originally was developed by Model Systems, Ltd [57]. It presents EU-Rent, a car rental company owned by EU-Corporation. It is one of three businesses –the other two being hotels and an airline– that each has its own business and IT systems, but with a shared customer base. Many of the car rental customers also fly with EU-Fly and stay at EU-Stay hotels.

EU-Rent has 1000 branches in towns in several countries. At each branch, cars –classified by car group– are available for rental. Each branch has a manager and booking clerks who handle rentals. Most rentals are by advance reservation; the rental period and the car group are specified at the time of reservation. EU-Rent will also accept immediate rentals, if cars are available. At the end of each day cars are assigned to reservations for the following day. If more cars have been requested than are available in a group at a branch, the branch manager may ask other branches if they have cars they can transfer to him/her. Apart from collecting information about cars, branches, etc., effort is done to capture information about customers (if they are good clients or had had bad experiences otherwise).

The conceptual schema of the EU-Rent car rental system (see Fig. 7.3) described in [47] was already used in this thesis for the experimental evaluation of the importance-computing methods introduced in Ch. 4. It is specified using standard UML and OCL. The structural subschema of the EU-Rent contains 65 entity types and 85 attributes. On the other hand, the behavioral subschema contains 120 event types and 166 pre- and postconditions specifying the effect of the events. Furthermore, the conceptual schema connects its elements through a set of 152 relationship types and 207 generalization relationships, and includes 107 integrity constraints and 7 enumeration types.

7.1.4 The UML Metaschema Formal Specification

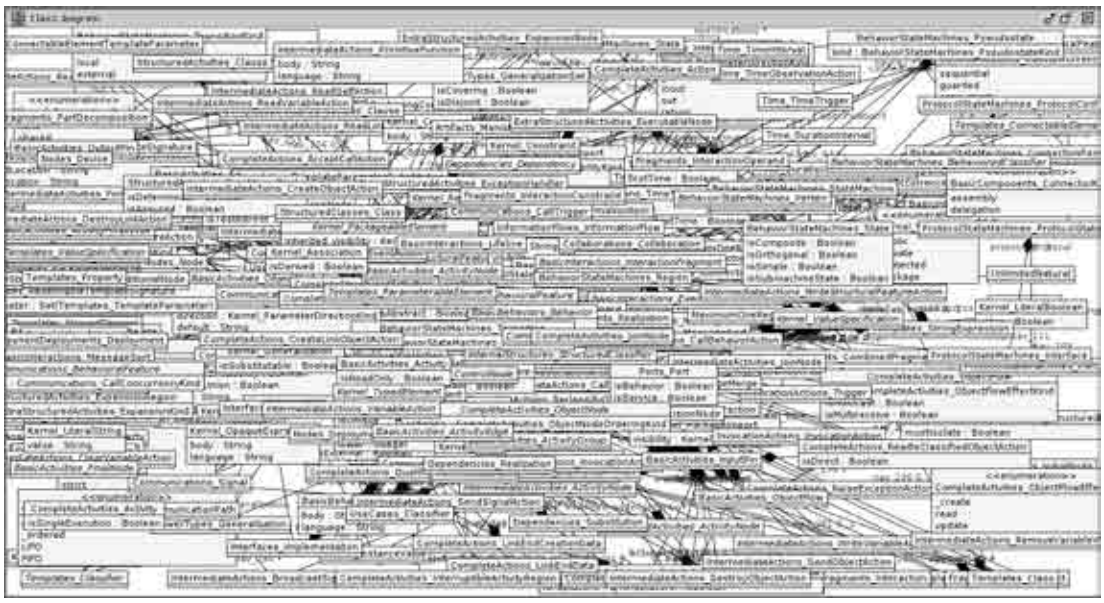


Figure 7.4. Metaschema of the UML Superstructure specification.

The Unified Modeling Language (UML) is managed, and was created, by the Object Management Group. It was first added to the list of OMG adopted technologies in 1997, and has since become the industry standard for modeling software-intensive systems[3]. The objective of UML is to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes.

The UML metaschema is like the grammar for UML models. In the same way that the Java grammar defines the structure of Java programs, the UML metaschema defines the concepts (i.e. modeling primitives) that you can use when defining UML models and the possible ways to relate them. Formally, the UML metaschema indicates a specification of the human-readable notation elements for representing the individual UML modeling concepts as well as rules for combining them into a variety of different diagram types.

The conceptual schema of the UML metaschema (see Fig. 7.4) was already used in this thesis for the experimental evaluation of the importance-computing methods introduced in Ch. 4. It is described using standard UML and OCL in a normative specification [84]. The UML metaschema contains 293 entity types and 93 attributes. It also specifies 9 pre- and postconditions, and 107 integrity constraints. Furthermore, the conceptual schema connects its elements through a set of 377 relationship types and 355 generalization relationships, and 13 enumeration types.

[3] `www.uml.org`

7.2 The e-Commerce Case Study

In this section, we describe the application of our filtering methodology in the comparison of the conceptual schemas of the osCommerce and Magento e-commerce systems introduced in Sect. 7.1.1 and Sect. 7.1.2. We describe the filtering scenario and show examples of usage of several filtering requests that help users to explore large conceptual schemas with the purpose of comparing the knowledge they contain.

7.2.1 Filtering Scenario

Nowadays, there is a wide range of tools and systems that provide similar functionalities for the particular domains for with they are developed. The selection of the most appropriate solution for a concrete problem is a central task in every organization.

As aforementioned, the osCommerce and the Magento are both valid frameworks that allow commercial organizations to enhance their business and start their activities through the e-commerce domain. Since the conceptual schema of an information system describes the knowledge an organization needs to know to perform its functions, the aim of this case study is to perform a conceptual schema comparison between osCommerce and Magento in order to support the process of selecting one of them.

Previous research papers have proposed many techniques to achieve a partial automation of schema matching operations [93]. Furthermore, there are generic comparison approaches with the ability to see the differences in a model when compared to another one [22, 107]. Our purpose here is to show that our filtering methodology can assist users through the comparison between two large conceptual schemas. To this end, we propose the use of the following filtering requests in this filtering scenario:

- $\mathcal{F}_1$ – **Filtering request for entity and relationship types**: the execution of this request for the same concept specified in osCommerce and Magento allows to explore the particularities and the level of detail specified within each conceptual schema.
- $\mathcal{F}_2$ – **Filtering request for schema rules**: the application of this request to the pre- and postconditions of the same event type specified in osCommerce and Magento allows to understand the differences in event effects.
- $\mathcal{F}_3$ – **Filtering request for event types**: the application of this request to event types specified in osCommerce and Magento allows to understand the differences in the structural definition of event types.
- $\mathcal{F}_5$ – **Filtering request for context behavior of entity types**: the usage of this request allows to obtain the different sets of event types that affect the same entity type in osCommerce and Magento.

The osCommerce conceptual schema contains 262 event types and 84 entity types. On the other hand, the conceptual schema of Magento defines 187 event types and 218 entity

types. There are 40 event types (≈15% of osCommerce's total and ≈21% of Magento's) and 36 entity types (≈43% of osCommerce's total and ≈16% of Magento's) that are specified in both conceptual schemas with different characteristics but sharing the same name.

A user that wants to compare both schemas can make use of filtering requests $\mathcal{F}_1$ and $\mathcal{F}_3$ to directly obtain the filtered schema with respect to a shared entity or event type. For instance, Fig. 7.5 presents the filtered schemas a user may obtain focusing on the entity type CreditCard. The knowledge we obtain from Magento is quite different from the knowledge we obtain from the osCommerce. It is possible to observe that the CreditCard concept in both schemas descends from the more general concept PaymentMethod. However, the filtered conceptual schema we obtain from Magento (see Fig. 7.5 (left)) contains a more precise level of detail about credit cards, including general attributes inherited from PaymentMethod and specific characteristics when related to a website (see the CreditCardInWebsite association class). On the other side, the filtered schema from osCommerce (see Fig. 7.5 (right) is simpler, specifying less information than Magento's.

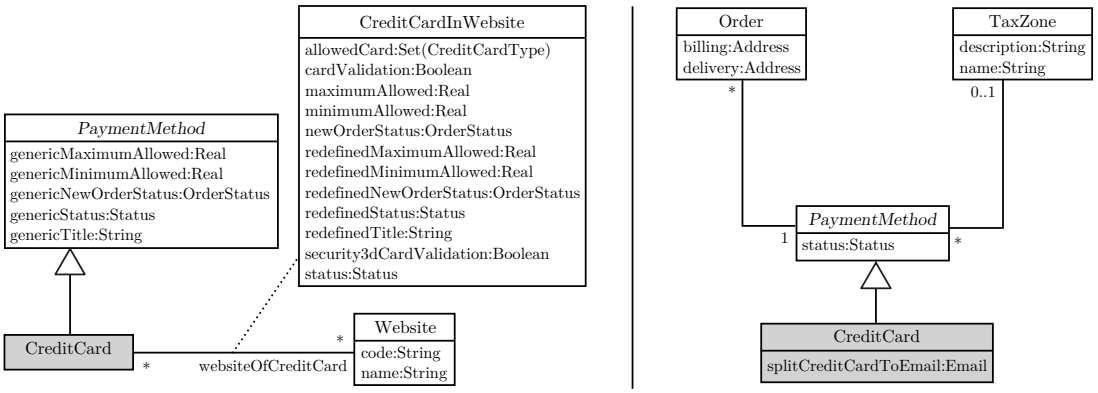


Figure 7.5. Entity type CreditCard in Magento (left) and osCommerce (right) through $\mathcal{F}_1$ ($\mathcal{K} = 4$).

Similarly, Fig. 7.6 presents the filtered schemas a user may obtain focusing on the event type CancelOrder. We observe that CancelOrder descends from ExistingOrderEvent in both schemas, which is directly related to the entity type Order. From the filtered schema of Magento (see Fig. 7.6 (left)), the user obtains additional information about the event type of focus. Concretely, it contains the specification of the event types AddInvoice and AddRefund, which both are semantically related to orders and relevant siblings of CancelOrder. On the other side, the filtered schema from osCommerce (see Fig. 7.6 (right)) only includes the event type of focus CancelOrder, its generalization relationship with ExistingOrderEvent, and the relationship type that connects it with Order. Also, the entity type Order only defines the billing and delivery address of the order as attributes of the entity. By contrast, in Magento the same entity type Order contains a set of 16 attributes, including information about shipping costs, coupon code (for discounts), e-mail address where to send information about the order, total amount of the order (indicating how much has been paid, invoiced, or refund), or purchase date. Furthermore, Magento allows versioning of orders through the relationship type OrdersVersion, which provides information about previous and new versions of a particular instance of Order. We can observe that this functionality is not supported in the osCommerce system.

Apart from directly comparing the filtered conceptual schemas of the shared entity and

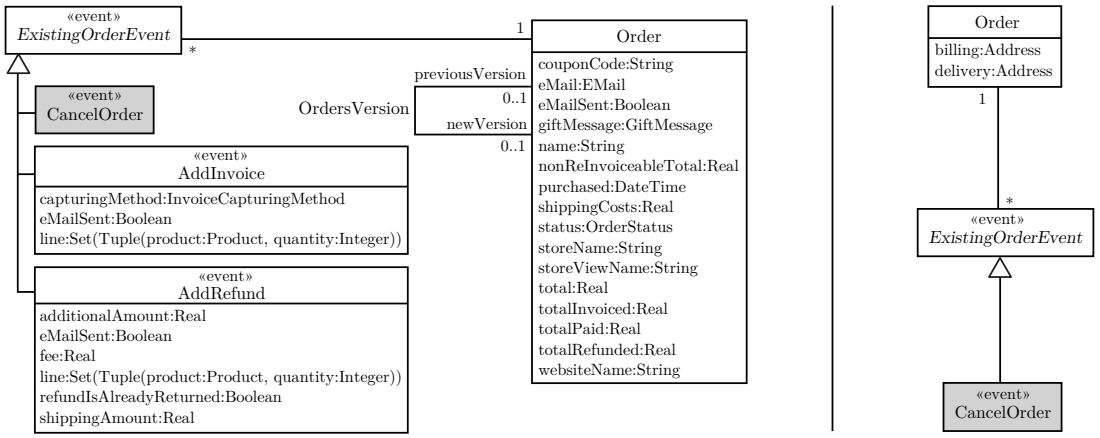


Figure 7.6. Event type CancelOrder in Magento (left) and osCommerce (right) through $\mathcal{F}_3$ ($\mathcal{K} = 5$).

event types, a user with interest in exploring the behavioral aspects of both schemas can use the filtering request $\mathcal{F}_5$ to obtain the specific event types that are related to a concrete entity type. As an example, Fig. 7.7 presents the filtered schemas a user may obtain through $\mathcal{F}_5$ focusing on the entity type ShoppingCart, which is one of the most relevant entity types in both schemas and represents a common metaphor (from the original grocery store shopping cart) for the catalog or other pages of the e-store where a user makes selections. Typically, the user checks off any products or services that are being ordered and then, when finished ordering, indicates that and proceeds to a page where the total order is placed and confirmed.

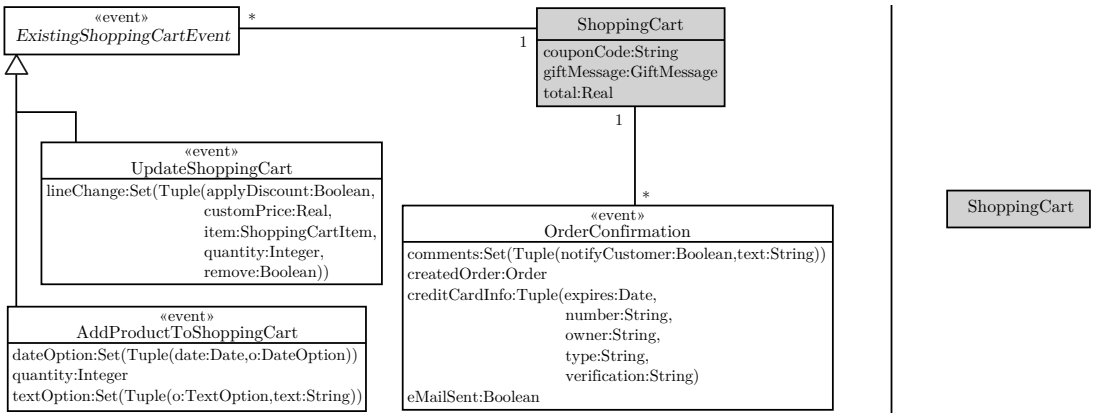


Figure 7.7. Related events to ShoppingCart in Magento (left) and osCommerce (right) through $\mathcal{F}_5$ ($\mathcal{K} = 5$).

Figure 7.7 (left) shows that in Magento, the entity type ShoppingCart is related to the event types OrderConfirmation, which requires the shopping cart of the user in order to confirm the products within it in a new instance of Order, and ExistingShoppingCartEvent, which is a general abstract event that generalizes the event types UpdateShoppingCart and AddProductToShoppingCart. Surprisingly, the osCommerce schema does not define any event type directly connected to the entity type ShoppingCart (see Fig. 7.7 (right)).

In order to deeply explore the event types that affect the state of the instances of the entity type ShoppingCart, the user can use the filtering request $\mathcal{F}_2$ to explore the pre- and postconditions that define the effect of one of the event types in Fig. 7.7. For instance, Fig. 7.8 presents the filtered schema affected by the effect postcondition of the event type AddProductToShoppingCart. We observe that in Magento, the effect of AddProductToShoppingCart (see Fig. 7.8 (top)) creates a new instance of ShoppingCartItem and connects it with the specific ShoppingCart and Product that is connected to the event itself. Also, the postcondition completes the information of the new instance about options of the product in the shopping cart (date and text options, with their ratings), and the quantity.

On the other side, we observe that the same event type in the osCommerce also creates a new instance of ShoppingCartItem with the instance of Product connected to AddProductToShoppingCart. Furthermore, the quantity of the new item of the shopping cart is specified in the event type (in the same manner as in Magento), and the Attributes of the selected product (which were the options in Magento) are connected to the new ShoppingCartItem. It is important to note that the ShoppingCart where the new ShoppingCartItem has to be connected is associated with the instance of Session in osCommerce. That is the reason why the event type AddProductToShoppingCart did not appear in the filtered schema of Fig. 7.7 (right). In addition to it, if the Session does not contain a ShoppingCart, the event creates a new instance to place the new item, which could be an AnonymousShoppingCart or a CustomerShoppingCart depending on whether the current session contains a logged customer or not.

7.2.2 Lessons Learned

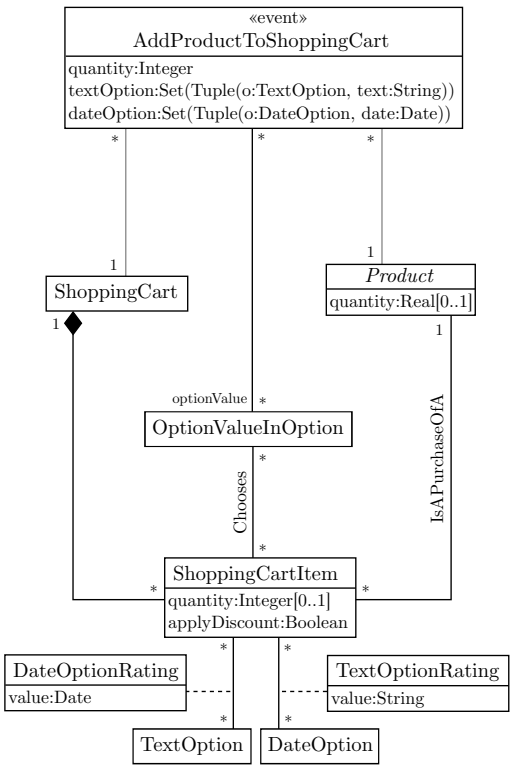
The analysis of the previous study indicates that our filtering method and, specially the catalog of filtering requests, are useful and provide adequate feedback to the user in the task of comparing the knowledge contained within two large conceptual schemas from the same domain of interest. Among the different filtering requests in our catalog, we believe that the filtering request for entity and relationship types ($\mathcal{F}_1$), and the filtering request for event types ($\mathcal{F}_3$) are the requests of choice for those users that start exploring and comparing large schemas.

A more experienced user shall use the filtering request for context behavior of entity types ($\mathcal{F}_5$) in order to know which are the event types that play a key role creating, updating and deleting instances of a particular entity type of interest. Furthermore, that user shall obtain the definition of the event pre- and postconditions and the fragment of the schema affected by the event effect by using the filtering request for schema rules ($\mathcal{F}_2$). We recommend the usage of this filtering request to users with experience with the syntax and semantics of OCL.

The results of the e-commerce comparison from this case study indicate that the Magento framework provides a more detailed approach to the e-commerce business than the one proposed by osCommerce. The entity and event types specified in the Magento contain a higher number of attributes and are associated through more relationship types that in osCommerce. However, the configuration of the osCommerce environment is easier than Magento's. Therefore, we believe that osCommerce is the option of choice for small-to-medium organizations that want to start sharing their business through a basic e-commerce system. On the contrary, those medium-

to-large organizations with experience in the e-commerce domain should select Magento as the best e-commerce solution.

```
context AddProductToShoppingCart::effect()
post addProduct:
  let existingItem:ShoppingCartItem
    = self.shoppingCart@pre.shoppingCartItem@pre
      ->any(i|i.product=self.product)
  in let productExists:Boolean=not existingItem.oclIsUndefined() and
       existingItem.optionValueInOption=self.optionValue and
       existingItem.textOptionRating
        ->collect(r| Tuple{o=r.textOption,text=r.value})->asSet()
        = self.textOption and existingItem.dateOptionRating
        ->collect(r| Tuple{o=r.dateOption,date=r.value})->asSet()
        = self.dateOption
  in if productExists then
    let i=self.shoppingCart.shoppingCartItem->any(i|i.product=self.product)
    in i.quantity = i@pre.quantity + self.quantity
  else
   (ShoppingCartItem.allInstances()-ShoppingCartItem.allInstances@pre())
    ->one(i:ShoppingCartItem |
      i.oclIsNew() and i.oclIsTypeOf(ShoppingCartItem) and
      i.shoppingCart = self.shoppingCart and i.product = self.product and
      i.quantity = self.quantity and i.applyDiscount = true and
      (self.textOption->notEmpty() implies
     self.textOption ->forAll(tupleOpt |
    (TextOptionRating.allInstances()-TextOptionRating.allInstances@pre())
        ->exists(r:TextOptionRating |
          r.oclIsNew() and r.oclIsTypeOf(TextOptionRating) and
          r.textOption = tupleOpt.o and r.value = tupleOpt.text and
          r.shoppingCartItem = i
    ) ) ) and (self.dateOption->notEmpty() implies
     self.dateOption->forAll(tupleOpt|
     (DateOptionRating.allInstances()-DateOptionRating.allInstances@pre())
      ->exists(r:DateOptionRating |
      r.oclIsNew() and r.oclIsTypeOf(DateOptionRating) and
      r.dateOption = tupleOpt.o and r.value = tupleOpt.date and
      r.shoppingCartItem = i
  ) ) ) and i.optionValueInOption = self.optionValue )
  endif
```




```
context AddProductToShoppingCart::effect()
post ShoppingCartItemIsCreated:
  (ShoppingCartItem.allInstances()-ShoppingCartItem.allInstances@pre())
   ->one(sci:ShoppingCartItem |
     sci.oclIsNew and  sci.oclIsTypeOf(ShoppingCartItem) and
     sci.quantity = self.quantity and sci.product = self.product and
     sci.attribute = self.attribute and
     if self.session.shoppingCart->notEmpty() then
        self.session.shoppingCart.shoppingCartItem->includes(sci)
     else if self.session.customer->isEmpty() then
        (AnonymousShoppingCart.allInstances() -
         AnonymousShoppingCart.allInstances@pre())
          ->one(sc:AnonymousShoppingCart |
          sc.oclIsNew() and oclIsTypeOf(AnonymousShoppingCart) and
          self.session.shoppingCart = sc and
          sc.shoppingCartItem->includes(sci))
     else if self.session.customer.customerShoppingCart->notEmpty() then
        self.session.shoppingCart = self.session.customer
                                          .customerShoppingCart and
        self.session.shoppingCart.shoppingCartItem->includes(sci)
     else (CustomerShoppingCart.allInstances() -
           CustomerShoppingCart.allInstances@pre())
            ->one(csc:CustomerShoppingCart |
              csc.oclIsNew() and
              csc.oclIsTypeOf(CustomerShoppingCart)  and
              self.session.shoppingCart = csc and
              csc.shoppingCartItem->includes(sci))
     endif endif endif )
```

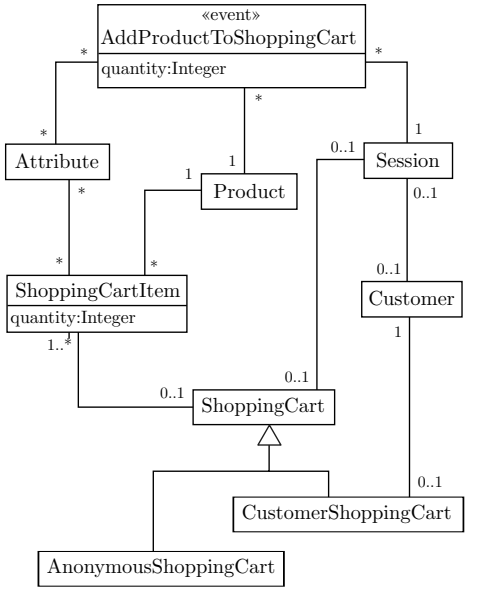


Figure 7.8. Effect of event type AddProductToShoppingCart in Magento (top) and osCommerce (bottom) through $\mathcal{F}_2$ (scope=*local*).

7.3 The EU-Rent Case Study

In this section, we describe the application of our filtering methodology in the exploration of the conceptual schema of the EU-Rent car rental system introduced in Sect. 7.1.3. We describe the filtering scenario and show examples of usage of several filtering requests that help users to explore the behavioral subschema of this large conceptual schema with the purpose of understanding the specific knowledge it contains and the way event types affect it through their pre- and postconditions.

7.3.1 Filtering Scenario

Modelers need to understand the knowledge contained in the behavioral subschema of a large conceptual schema. In the development of information systems, modelers need eventually to check with a domain expert that the behavioral part of the schema is correct, database designers need to implement that behavior into a relational database, software testers need to write tests checking that the behavior has been correctly implemented in the system components, and the members of the maintenance team need to change the behavior of a set of event types.

In large conceptual schemas, human understanding of the semantics of the behavioral subschema is difficult due to the pre- and postconditions the effect of event types contain. The problem is not the formal language in which they are written (the OCL in UML [85]), but the fact that the elements (entity types, attributes, relationship types) involved in an expression are defined in different places in the schema, which may be very distant from each other and embedded in an intricate web of irrelevant elements for the purpose at hand. The problem is insignificant when the schema is small, but very significant when it is large.

Our purpose here is to show that our filtering methodology can assist users through the exploration of the behavioral subschema of a given conceptual schema. To this end, we propose the use of the following filtering requests in this filtering scenario:

- $\mathcal{F}_2$ – **Filtering request for schema rules**: the application of this request to the pre- and postconditions of an event type specified in EU-Rent allows to understand the effect of the event and the structural schema that is affected by such effect.
- $\mathcal{F}_3$ – **Filtering request for event types**: the application of this request to event types specified in EU-Rent allows to understand the structural definition of event types in the schema.
- $\mathcal{F}_5$ – **Filtering request for context behavior of entity types**: the usage of this request allows to obtain the different sets of event types that affect an entity type in EU-Rent.
- $\mathcal{F}_6$ – **Filtering request for contextualized types**: the execution of this request allows to derive a specific view of interest from a set of entity and event types of focus taking into account a contextualization function.

The behavioral subschema of the EU-Rent conceptual schema contains 120 event types and 166 pre- and postconditions. It almost doubles the number of entity types from the structural subschema (65), which denotes the relevance of the behavioral subschema.

A user that wants to explore the behavioral subschema may start using the filtering request for context behavior of entity types ($\mathcal{F}_5$) to directly obtain a filtered schema with the most interesting event types with respect to an entity type of focus. For instance, Fig. 7.9 presents the filtered schema a user may obtain focusing on the entity type Car, which is the most relevant one in EU-Rent according to the importance-computing methods presented in Ch. 4.

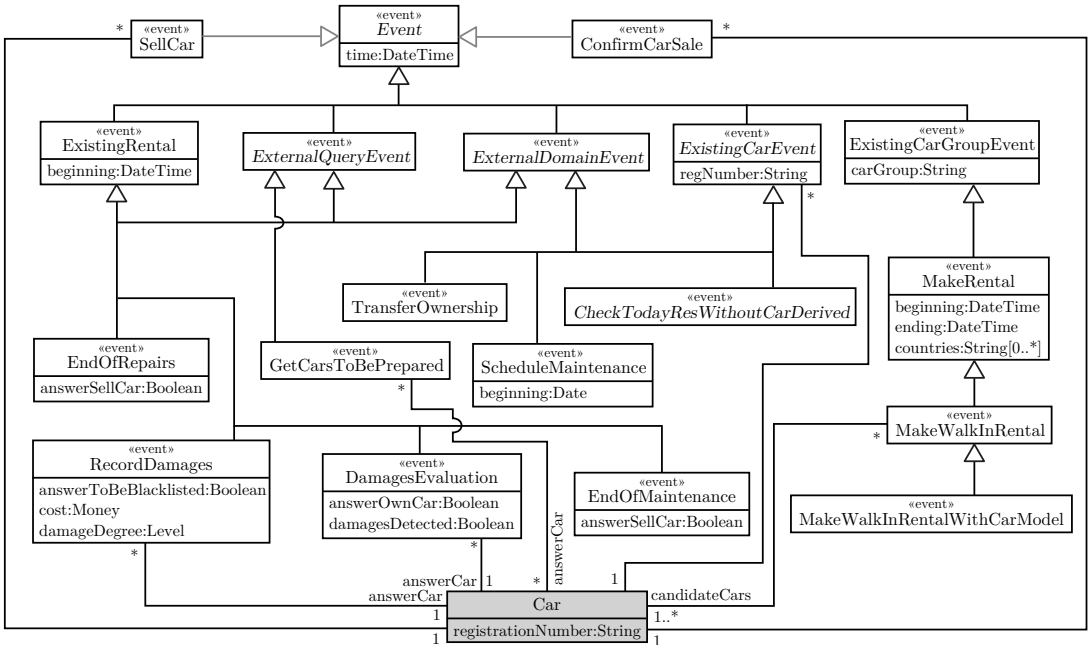


Figure 7.9. Related events to Car in EU-Rent through $\mathcal{F}_5$ ($\mathcal{K} = 20$).

This filtered conceptual schema contains a hierarchy of event types with relation to Car. Analyzing the name of the events is enough to obtain a first view of the functionalities EU-Rent provides. It is possible to observe that there are events to create a car rental (MakeRental), schedule some maintenance of a car, transfer the ownership of a car, or event evaluate its damages, and record them in a rental. This filtering request provides us with a summary of the behavioral schema that serves as starting point and makes possible to continue the exploration with additional filtering requests.

The user may select one the event types in the filtered schema of Fig. 7.9 and explore in detail the fragment of the large schema with higher relation to that event. As an example, Fig. 7.10 shows the filtered conceptual schema our filtering request for event types ($\mathcal{F}_3$) provides when the user focus on the event type MakeRental.

There are two types of event types to create rentals: MakeRental and MakeWalkInRental, which is a descendant of MakeRental. The event type MakeRental contains information about the rental period (that is, pick-up time and day of drop-off), the pick-up and drop-off branches

where to get and return the car of the rental, and the instance of EU-RentPerson that indicates the person who makes the rental. Recording the moment in which the rental is made will be also useful for defining a priority criteria to allocate cars. On the other hand, the MakeWalkInRental event creates walk-in rentals, which are immediate and depend on the availability of cars.

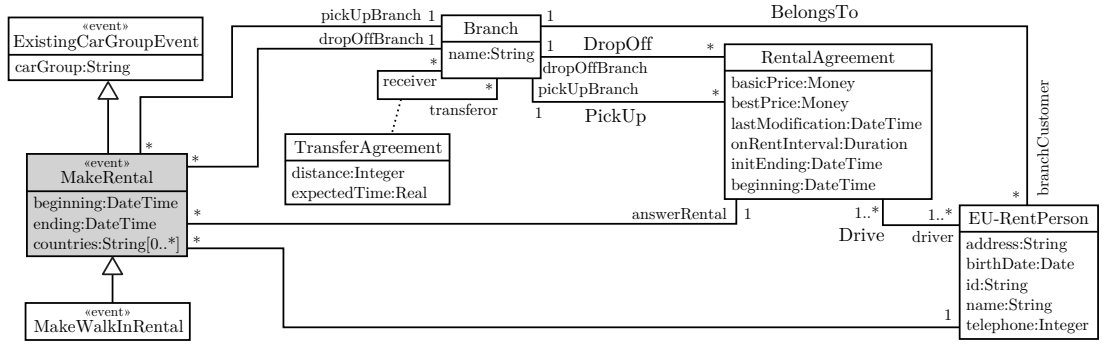
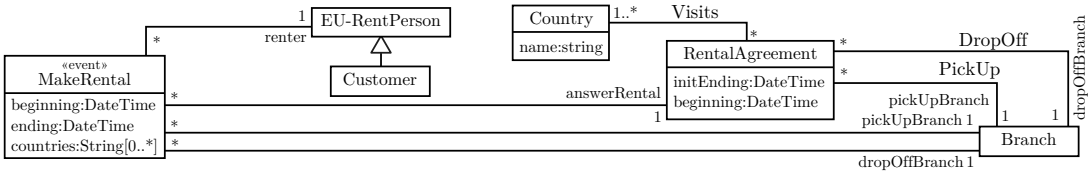


Figure 7.10. Event type MakeRental in EU-Rent through $\mathcal{F}_3$ ($\mathcal{K} = 7$).

In order to analyze the details of the MakeRental effect, the user may use the filtering request for schema rules ($\mathcal{F}_2$) applied to the postcondition of this event type. Figure 7.11 presents the filtered schema that results from the application of $\mathcal{F}_2$ to the postcondition of the effect of MakeRental. Note that it only includes the elements that are referenced by that postcondition, which indicates that the effect of MakeRental consists of the creation of a new instance of RentalAgreement.

Then, the postcondition sets the characteristics of the new instance, including the beginning and ending dates of the rental (which are attributes of MakeRental), the customer who makes the rental (the renter associated with MakeRental), and the pick-up and drop-off branches of the rental (also associated to MakeRental). We assume that countries through which the customer is going to travel are also given in the countries attribute of MakeRental, and transferred to the instance of RentalAgreement that represents the new rental in the EU-Rent system.

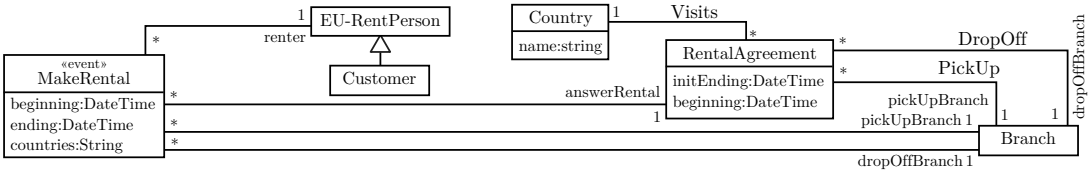



```
context MakeRental::effect():Boolean
post:
 let getCountries:Set(Country)= Country.allInstances()->select(c|
       self.countries->includes(c.name))
 in (RentalAgreement.allInstances() - RentalAgreement.allInstances@pre())
    ->one(rent:RentalAgreement |  renter.oclIsTypeOf(Customer) and
        rent.oclIsNew() and  rent.oclIsTypeOf(RentalAgreement) and
        rent.beginning=self.beginning and rent.renter=self.renter and
        rent.initEnding=self.ending and rent.pickUpBranch=self.pickUpBranch and
        rent.dropOffBranch=self.dropOffBranch and rent.country=getCountries and
        self.answerRental=rent )
```

Figure 7.11. Effect of event type MakeRental in EU-Rent through $\mathcal{F}_2$ (scope=*local*).

Once the user understands the effect of the event type, she may need to modify its specified behavior in order to update the definition of the schema according to new business requirements of the EU-Rent organization. Then, the user should use the filtering request for contextualized types ($\mathcal{F}_6$) and construct a contextualization function that matches the new business context.

As an example, Fig. 7.12 presents the filtered conceptual schema that results from the application of the filtering request $\mathcal{F}_6$ with a focus set containing the entity and event types from the schema of Fig. 7.11 and a contextualization function. Concretely, the contextualization function modifies the multiplicity of the attribute countries of MakeRental from 0..* to 1..1, and the multiplicity of the association Visits in the side of Country from 0..* to 1..1. With these modifications, we can simulate an alternative event type MakeRental in which only one country is allowed in a rental agreement. It means that the customer cannot drive across countries with the car rented through this event type. Note that the postcondition effect in Fig. 7.12 has been manually modified according to the contextualization.




```
context MakeRental::effect():Boolean
post:
 let getCountry:Country=Country.allInstances()->select(c| c.name=self.country)->any(true)
 in (RentalAgreement.allInstances() - RentalAgreement.allInstances@pre())
     ->one(rent:RentalAgreement |  renter.oclIsTypeOf(Customer) and
         rent.oclIsNew() and  rent.oclIsTypeOf(RentalAgreement) and
         rent.beginning=self.beginning and rent.renter=self.renter and
         rent.initEnding=self.ending and rent.pickUpBranch=self.pickUpBranch and
         rent.dropOffBranch=self.dropOffBranch and rent.country=getCountry and
         self.answerRental=rent )
```

Figure 7.12. Contextualization for the effect of the event type MakeRental in EU-Rent through $\mathcal{F}_6$.

7.3.2 Lessons Learned

The analysis of the previous study indicates that our filtering method and, specially the catalog of filtering requests, are useful and provide adequate feedback to the user in the task of exploring the behavioral subschema within a large conceptual schemas. Among the different filtering requests in our catalog, we believe that the filtering request for context behavior of entity types ($\mathcal{F}_5$), and the filtering request for event types ($\mathcal{F}_3$) are the requests of choice for those users that start exploring the event types from behavioral subschemas.

A more experienced user shall use the filtering request for schema rules ($\mathcal{F}_2$) in order to know the details about the pre- and postconditions and the fragment of the schema affected by an event effect. We recommend the usage of this filtering request to users with experience with the syntax and semantics of OCL. Also, the user interested in obtaining a contextualized filtered schema shall make use of the filtering request for contextualized types ($\mathcal{F}_6$). We have identified a need of future work to automatically refactor schema rules after the application of a contextualization function that changes multiplicities of elements referenced by those rules.

7.4 The UML Metaschema Case Study

In this section, we describe the application of our filtering methodology in the exploration of the UML metaschema introduced in Sect. 7.1.4. We describe the filtering scenario and show examples of usage of several filtering requests that help users to explore this large conceptual schema. Since the UML metaschema is currently described through a formal specification document provided by the Object Management Group (OMG, see [84]), we compare the contents of this document with the results of applying a set of filtering requests.

7.4.1 Filtering Scenario

The Unified Modeling Language (UML) provides system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes. The OMG maintains and supports the specification of the UML metaschema and provides a full document with its formal definition in [84].

The document with the UML metaschema definition includes more than 700 pages with the technical aspects of this formal specification. It defines the static, structural constructs (e.g., classes, components, node artifacts) used in various structural diagrams provided by the UML graphical modeling language, such as class diagrams, component diagrams, and deployment diagrams. Also, the document specifies the dynamic, behavioral constructs (e.g., activities, interactions, state machines) used in various behavioral diagrams, such as activity diagrams, sequence diagrams, and state machine diagrams. Finally, it defines auxiliary constructs (e.g., information flows, models, templates, primitive types) and the profiles used to customize UML for various domains, platforms, and methods.

Although the clauses of the UML metaschema document are organized in a logical manner and can be read sequentially, it is a reference specification and is intended to be read in a non-sequential manner. Consequently, extensive cross-references are provided to facilitate browsing and search.

Our purpose here is to show that our filtering methodology can also be useful even when there is a good-enough documentation with the formal specification of a given large conceptual schema a user wants to explore. To this end, we propose the use of the following filtering requests in this filtering scenario:

- $\mathcal{F}_1$ – **Filtering request for entity and relationship types**: the execution of this request with a set of entity and relationship types of focus allows to explore the particularities and the level of detail specified about these elements within the UML metaschema.
- $\mathcal{F}_2$ – **Filtering request for schema rules**: the application of this request to the integrity constraints, derivation rules, and pre- and postconditions specified in the UML metaschema allows to understand the structural schema that is affected by the definition of each of these schema rules.

A user that wants to explore the UML metaschema may start using the filtering request for entity and relationship types ($\mathcal{F}_1$) to directly obtain a filtered schema with the most interesting elements with respect to a focus set of entity and/or relationship types. For instance, Fig. 7.13 presents the filtered schema a user may obtain focusing on the entity type Property, which is one of the most relevant entity types in the UML metaschema according to the importance-computing methods presented in Ch. 4.

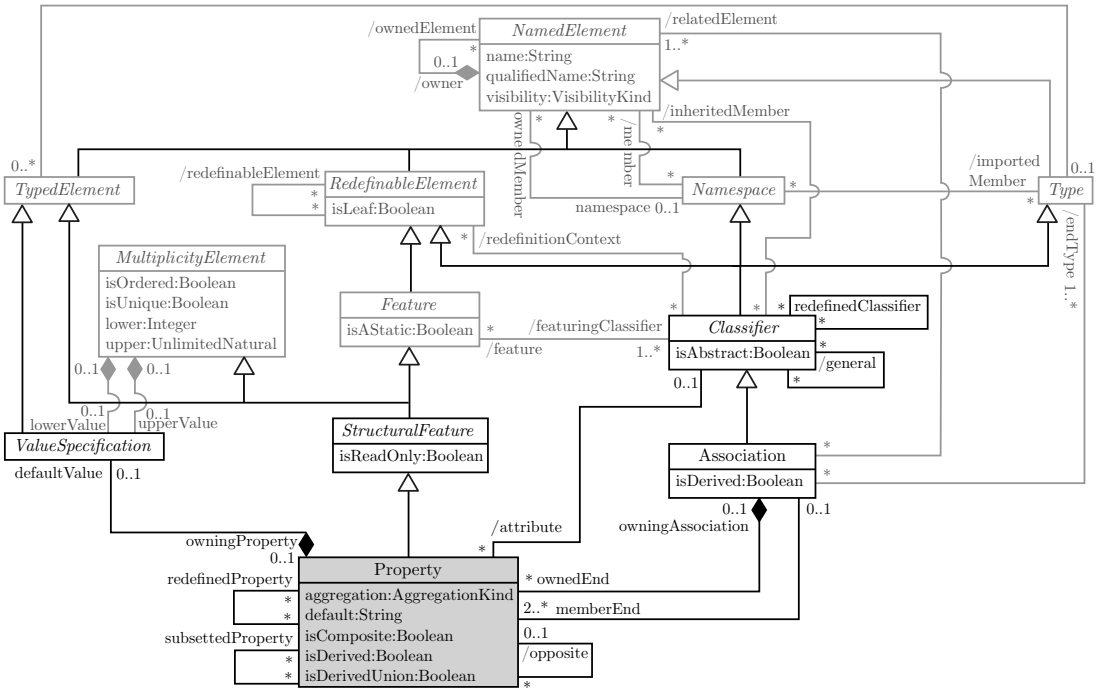


Figure 7.13. Entity type Property in the UML metaschema through $\mathcal{F}_1$ ($\mathcal{K} = 5$).

Alternatively, the user may explore the UML metaschema documentation and find a similar fragment of the whole large schema that focus on the entity types Property, Class and Association, as shown in Fig. 7.14. The UML metaschema document contains several figures that show fragments of the large metaschema. We can see these fragments as static schema summaries or clusters of elements. As stated in the review of techniques and approaches to deal with large conceptual schemas of Ch. 3, clustering and summarization methods provide good feedback to users that want to obtain a general view of a given large conceptual schema. However, these techniques are of little use to those users that want to explore particular aspects of the schema or sets of elements whose information is spread out in several summaries or clusters.

In the previous example of Fig. 7.13, we observe that a Property is a descendant of the entity type StructuralFeature, which also descends from the more general entity types Feature, MultiplicityElement, and TypedElement. The entity types that are marked in gray are auxiliary and the relationships that are also gray are projected relationship types from the original schema, as explained in Ch. 5. It is important to note that the filtered schema obtained through the application of the filtering request for entity and relationship types ($\mathcal{F}_1$) explicitly indicates the inheritance paths of interest for Property, which are specially useful to observe

that a Property is a MultiplicityElement that has a lower and an upper value. This knowledge is of high relevance for the user of UML, but it does not appear in the fragment of the specification shown in Fig. 7.14. A user that wants to explore the same hierarchy of the entity type Property that appears in Fig. 7.13 must traverse a set of 7 fragments of schemas in the UML metaschema specification document.

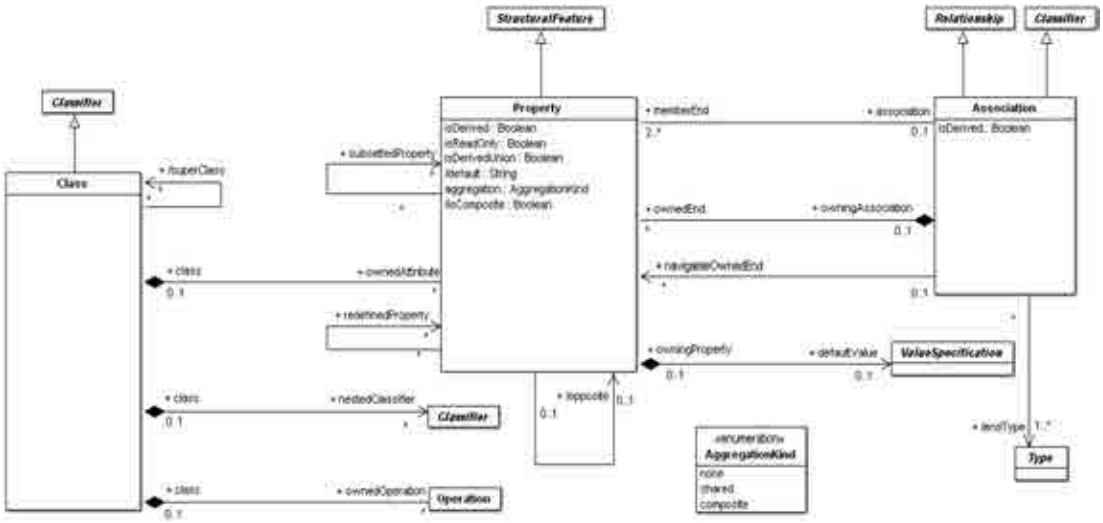


Figure 7.14. Entity type Property in the UML metaschema formal specification (see [84]).

Once the user has obtained the filtered conceptual schema of interest, she can continue exploring the schema rules defined in the context of the elements that appear in that filtered schema. For instance, Fig. 7.13 shows that an association end with rolename *opposite* in the reflexive binary relationship type between two instances of Property is derived. Therefore, the user may be interested in understanding the semantics of the derivation rule of *opposite*. To this purpose, the user can explore the UML metaschema document and then find the definition of the derivation rule of interest, which is shown in Fig. 7.15. Therefore, the user must simultaneously study that OCL expression defining the derivation rule and the schema summary of Fig. 7.14 in order to understand the semantics of the rule and discover the elements that are involved in the derivation.

Constraints

[1] If this property is owned by a class associated with a binary association, and the other end of the association is also owned by a class, then opposite gives the other end.

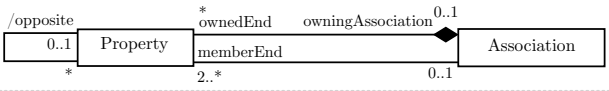
```
opposite =
    if owningAssociation->isEmpty() and association.memberEnd->size() = 2 then
        let otherEnd = (association.memberEnd - self)->any() in

            if otherEnd.owningAssociation->isEmpty() then otherEnd else Set{} endif
    else Set {}
    endif
```

Figure 7.15. Derivation rule of Property::*opposite* in the UML metaschema specification (see [84]).

Alternatively, the user can make use of the filtering request for schema rules ($\mathcal{F}_2$) and locally focus on the derivation rule of the *opposite* association end. As a result, the user automatically obtains the minimum filtered conceptual schema with the elements that are referenced by that derivation rule. Figure 7.16 presents the corresponding filtered schema and the specification of the derivation rule for the association end *opposite*. It is possible to observe that only the entity types Property and Association take part in the derivation rule.

Analyzing the filtered schema it is easy to understand that the opposite Property p' of a given Property p is the member end property that participates in a binary association with p and that it is not owned by an association. Concretely, if we have the binary relationship type $r(p, p')$, then $p = opposite(p')$ and $p' = opposite(p)$.




```
context Property::opposite:Property
derive:
   if owningAssociation->isEmpty() and association.memberEnd->size() = 2
   then
     let otherEnd = (association.memberEnd - self)->any()
     in
       if otherEnd.owningAssociation->isEmpty()
       then otherEnd
       else Set{}
       endif
   else Set{}
   endif
```

Figure 7.16. Derivation rule of Property::*opposite* in the UML metaschema through $\mathcal{F}_2$ (scope=*local*).

7.4.2 Lessons Learned

The analysis of the previous study indicates that our filtering method and, specially the catalog of filtering requests, are useful and provide adequate feedback to the user in the task of exploring a large conceptual schema that also includes a good-enough documentation that formally specifies the contents of the schema and its semantics. Among the different filtering requests in our catalog, we believe that the filtering request for entity and relationship types ($\mathcal{F}_1$), and the filtering request for schema rules ($\mathcal{F}_2$) are the requests of choice for those users that start exploring the characteristics of a documented conceptual schema.

The results of analyzing this case study indicate that having a good documentation for a large conceptual schema is useful to inexperienced users that have interest on exploring the characteristics of the schema. However, there are situations where a user is interested on several aspects from a large schema that are defined in different places of the documentation, which may be very distant from each other and embedded in an intricate web of other irrelevant elements for the purpose at hand. The application of the filtering requests to a large schema like the UML metaschema provides a more dynamic exploration approach that saves time and reduces the searching effort a user must dedicate to find the specific portions of the whole schema of interest to satisfy a particular need of information.

7.5 Experimental Evaluation

Finding a measure that reflects the ability of our filtering methodology to satisfy the user is a complicated task in the field of information retrieval [10]. Usually, the distinction is made between the evaluation of the effectiveness and the efficiency of a retrieval method. While the effectiveness measures the benefits obtained from the application of the filtering method, the efficiency indicates the time interval between the request being made and the answer being given.

We have implemented all the filtering requests of our filtering method (see the details in Ch. 8) and we have then evaluated the efficiency and effectiveness of these filtering requests by using the conceptual schemas of the previous case studies: the schema of Magento [94], the UML metaschema [84, 8], the schema of osCommerce [118], and the EU-Rent car rental schema [47]. In the following, we present the results we have obtained from the analysis of the resulting data.

7.5.1 Effectiveness

The two most frequent and basic measures for information retrieval effectiveness are precision and recall [10]. Precision is the fraction of retrieved elements that are relevant, while recall is the fraction of relevant elements that are retrieved. The measures of precision and recall concentrate the evaluation on the return of true positives, asking what percentage of the relevant elements have been found and how many false positives have also been returned. Unfortunately, there is a wide range of information retrieval situations in which precision and recall are not directly applicable.

The main reason why we cannot apply precision and recall to our filtering methodology evaluation is that we do not have an oracle or expert that could help us to mark the elements of a very large schema as relevant or irrelevant due to the size of the schema itself. Furthermore, we cannot have an expert capable of marking the relevance of elements according to the needs of a particular user. Therefore, our effectiveness evaluation computes the benefits of our method with respect to the user effort when manually exploring a large conceptual schema with the purpose of obtaining a fragment of interest.

The application of the filtering requests of our method produces a filtered conceptual schema of small size that helps understanding the elements defined in a large schema. We compare the final size of a filtered schema with the size of its contextual schema, i.e, the portion of the large schema the user needs to manually explore in order to cover the elements referenced by the filtered schema starting from the elements of focus from the input of a filtered request.

As stated in the first part of this thesis, we formally define a conceptual schema as a tuple $\mathcal{CS} = \langle \mathcal{SS}, \mathcal{BS} \rangle$, where $\mathcal{SS} = \langle \mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{C}, \mathcal{D} \rangle$ is the structural subschema, and $\mathcal{BS} = \langle \mathcal{E}_b, \mathcal{R}_b, \mathcal{G}_b, \mathcal{C}_b \rangle$ is the behavioral subschema. Similarly, we formally define a filtered conceptual schema as a tuple $\mathcal{CS}_\mathcal{F} = \langle \mathcal{SS}_\mathcal{F}, \mathcal{BS}_\mathcal{F} \rangle$, where $\mathcal{SS}_\mathcal{F} = \langle \mathcal{E}_\mathcal{F}, \mathcal{R}_\mathcal{F}, \mathcal{T}_\mathcal{F}, \mathcal{G}_\mathcal{F}, \mathcal{C}_\mathcal{F}, \mathcal{D}_\mathcal{F} \rangle$ is the structural subschema, and $\mathcal{BS}_\mathcal{F} = \langle \mathcal{E}_{b\mathcal{F}}, \mathcal{R}_{b\mathcal{F}}, \mathcal{G}_{b\mathcal{F}}, \mathcal{C}_{b\mathcal{F}} \rangle$ is the behavioral subschema.

Consequently, we formally define the components of the contextual schema $\mathcal{CS}_\mathcal{C} = \langle \mathcal{E}_\mathcal{C}, \mathcal{E}_{b\mathcal{C}}, \mathcal{R}_\mathcal{C}, \mathcal{A}_\mathcal{C}, \mathcal{G}_\mathcal{C} \rangle$ as:

$$\begin{aligned}
\mathcal{E}_\mathcal{C} &= \mathcal{E}_\mathcal{F} \cup \mathcal{E}_\mathcal{G} \cup \mathcal{E}_\mathcal{R}, \\
\mathcal{E}_{b\mathcal{C}} &= \mathcal{E}_{b\mathcal{F}} \cup \mathcal{E}_{b\mathcal{G}} \cup \mathcal{E}_{b\mathcal{R}}, \\
\mathcal{R}_\mathcal{C} &= \{R(p_1{:}C_1, ..., p_n{:}C_n) \in \mathcal{R} \mid \exists C_i, p_i \ (C_i \in \{\mathcal{E}_\mathcal{C} \cup \mathcal{E}_{b\mathcal{C}}\} \wedge p_i{:}C_i \in R)\}, \\
\mathcal{A}_\mathcal{C} &= \{a \in \{\mathcal{A} \cup \mathcal{A}_b\} \mid \exists C, T \ (C \in \{\mathcal{E}_\mathcal{C} \cup \mathcal{E}_{b\mathcal{C}}\} \wedge T \in \mathcal{T} \wedge a(C, T))\}, \\
\mathcal{G}_\mathcal{C} &= \{g \in \{\mathcal{G} \cup \mathcal{G}_b\} \mid \exists C_i, C_j \ (C_i, C_j \in \{\mathcal{E}_\mathcal{C} \cup \mathcal{E}_{b\mathcal{C}}\} \wedge g{:}(C_i \text{ IsA } C_j))\},
\end{aligned}$$

where $\mathcal{E}_\mathcal{F}$ and $\mathcal{E}_{b\mathcal{F}}$ are the entity and event types from the filtered schema. $\mathcal{E}_\mathcal{G}$ and $\mathcal{E}_{b\mathcal{G}}$ contain the entity and event types that are intermediate members of the paths of generalization relationships between members of $\mathcal{E}_\mathcal{F}$ and $\mathcal{E}_{b\mathcal{F}}$, respectively. As a very simple example, consider a schema with the *2*-level specialization hierarchy e'' IsA e' IsA e of entity types. If only $e'', e \in \mathcal{E}_\mathcal{F}$ of the filtered schema, then the filtering method creates an indirect generalization relationship $g{:}(e''$ IsA $e)$ between them s.t. $g \in \mathcal{G}_\mathcal{F}$. In that case, the set $\mathcal{E}_\mathcal{G}$ from the contextual schema contains e' ($\mathcal{E}_\mathcal{G} = \{e'\}$).

In the same way, $\mathcal{E}_\mathcal{R}$ and $\mathcal{E}_{b\mathcal{R}}$ are the participant entity and event types in relationship types and attributes of the filtered schema before applying projection. As a very simple example, consider a schema with the *1*-level specialization hierarchy e' IsA e of entity types, and the relationship type *R(r1:e, r2:e'')*$\in \mathcal{R}$. If only $e'', e' \in \mathcal{E}_\mathcal{F}$ of the filtered schema, then the filtering method projects the relationship type R to these entity types as in *R(r1:e', r2:e'')*$\in \mathcal{R}_\mathcal{F}$. In that case, the set $\mathcal{E}_\mathcal{R}$ from the contextual schema contains e and e'' ($\mathcal{E}_\mathcal{R} = \{e, e''\}$).

Finally, the set $\mathcal{R}_\mathcal{C}$ contains the relationship types whose participants belong to $\mathcal{E}_\mathcal{C}$ or $\mathcal{E}_{b\mathcal{C}}$. The set $\mathcal{A}_\mathcal{C}$ contains the attributes defined in the context of entity types of $\mathcal{E}_\mathcal{C}$ or event types of $\mathcal{E}_{b\mathcal{C}}$. And the set $\mathcal{G}_\mathcal{C}$ contains the generalization relationships types between entity types of $\mathcal{E}_\mathcal{C}$ or event types of $\mathcal{E}_{b\mathcal{C}}$.

Therefore, we define the filtering utility factor Δ between the size of a given filtered schema $\mathcal{CS}_\mathcal{F}$ and the size of its respective contextual schema $\mathcal{CS}_\mathcal{C}$ as follows:

$$\begin{aligned}
\text{Filtering Utility Factor: } \Delta &= 1 - \frac{\Sigma(\mathcal{CS}_\mathcal{F})}{\Sigma(\mathcal{CS}_\mathcal{C})}, \text{ where} \\
\Sigma(\mathcal{CS}_\mathcal{F}) &= |\mathcal{E}_\mathcal{F}| + |\mathcal{E}_{b\mathcal{F}}| + |\mathcal{R}_\mathcal{F}| + |\mathcal{R}_{b\mathcal{F}}| + |\mathcal{A}_\mathcal{F}| + |\mathcal{A}_{b\mathcal{F}}| + |\mathcal{G}_\mathcal{F}| + |\mathcal{G}_{b\mathcal{F}}|, \\
\text{and } \Sigma(\mathcal{CS}_\mathcal{C}) &= |\mathcal{E}_\mathcal{C}| + |\mathcal{E}_{b\mathcal{C}}| + |\mathcal{R}_\mathcal{C}| + |\mathcal{A}_\mathcal{C}| + |\mathcal{G}_\mathcal{C}|.
\end{aligned}$$

As a result, Fig. 7.17 presents a set of box plots with the resulting values for the filtering utility factor applied to each of the previous case studies. For each schema, the plot indicates the smallest observation (sample minimum), lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation (sample maximum). Also, the black diamonds indicate the mean of each sample. The bottom and top of each box (Q1 and Q3) are the 5th and 95th percentiles, which means that the box contains the 90% of the samples.

Figure 7.17(a) indicates the results of the filtering request for entity and relationship types ($\mathcal{F}_1$) when applied to each of the 660 entity types and of the 1,031 relationship types specified in the four schemas from the case studies.

Figure 7.17(b) indicates the results of the filtering request for schema rules ($\mathcal{F}_2$) when applied to each of the 1,453 integrity constraints, derivation rules, pre- and postconditions specified in the four schemas from the case studies setting the scope of the request to local. Figure 7.17(c) presents similar results when the scope is set to global.

Figure 7.17(d) indicates the results of the filtering request for event types ($\mathcal{F}_3$) when applied to each of the 569 event types specified in three of the four schemas from the case studies. Note that the UML metaschema does not contain a behavioral subschema with event types.

Figure 7.17(e) indicates the results of the filtering request for a conceptual schema ($\mathcal{F}_4$) when applied to each of the 569 filtered conceptual schemas obtained from the application of each of the filtering requests used on the other box plots. Note that we executed this request demanding a resulting filtered schema with a 10% more of elements than the one from the input. It was achieved by properly setting the size threshold $\mathcal{K}$.

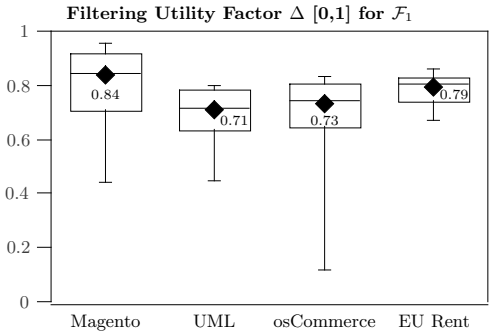
Figure 7.17(f) indicates the results of the filtering request for context behavior of entity types ($\mathcal{F}_5$) when applied to each of the 367 entity types specified in three of the four schemas from the case studies. Note that the UML metaschema does not contain a behavioral subschema with event types. Since none of the entity types are related to event types, Fig. 7.17(g) presents the same results as in Fig. 7.17(f) but without the results where the filtering utility factor was zero, which indicates that situation of entity types that are not related to event types. Concretely, there were 74 entity types without direct relationships with event types.

It is important to note that there is no box plot for the analysis of the effectiveness of the filtering request for contextualized types ($\mathcal{F}_6$) because the definition of a contextualization function requires a specific user intervention. However, we can assume the results from this request to be similar than those of $\mathcal{F}_1$, $\mathcal{F}_3$, and $\mathcal{F}_4$ because the filtering approach is the same.

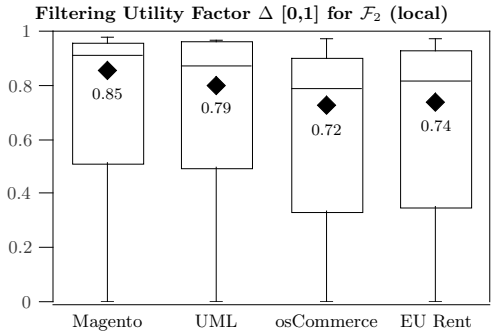
We observe that the mean value of the filtering utility factor exceeds 0.7 in the box plots for $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}_4$, which indicates a size reduction greater than 70% using filtered schemas instead of exploring the whole schema manually. The results of the $\mathcal{F}_3$ are quite similar, showing a reduction in the filtering utility factor of the osCommerce with a mean value of 0.63, which is good enough to continue using that request. The results from the previous box plots indicate a significant reduction of the cognitive effort a user has to face when understanding the characteristics of a large schema.

The smallest observations of the filtering utility factor in the box plot of $\mathcal{F}_2$ ($\Delta = 0$) indicate that the size of the filtered schema equals the size of the contextual schema ($\Sigma(\mathcal{CS}_{\mathcal{F}}) = \Sigma(\mathcal{CS}_{\mathcal{C}})$). This situation occurs whenever a schema rule of focus only references all the attributes of a single entity or event type without relationships to other elements, as in the case of primary key constraints of isolated types. In our experimentation, the schema rules that cause this represent less than a 2% of the total schema rules analyzed by this study.

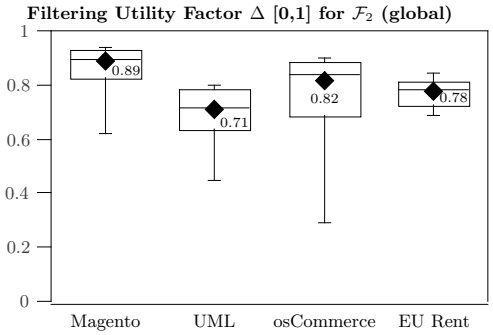
Also, the results from $\mathcal{F}_5$ in Fig. 7.17(f) and Fig. 7.17(g) are slightly smaller than the other results because a substantial portion of the 367 entity types that were analyzed are only related to one event type, which reduces the overall filtering utility factor of the box plots. However, the normalized results from Fig. 7.17(g) indicates a size reduction greater than 50% using our method, which is good enough in comparison to manually exploring the whole schema.



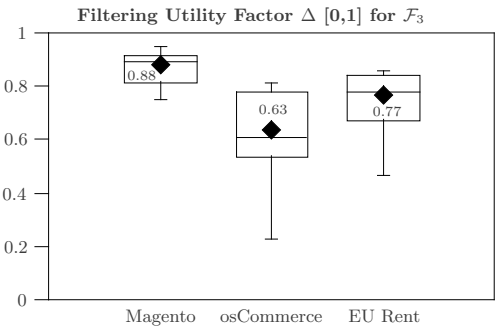
(a) Filtering utility factor for $\mathcal{F}_1$.



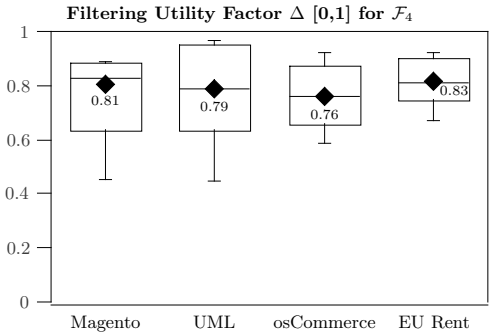
(b) Filtering utility factor for $\mathcal{F}_2$ (local).



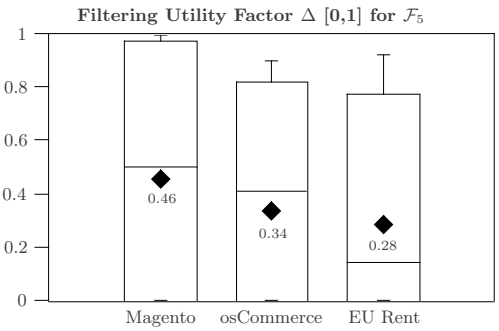
(c) Filtering utility factor for $\mathcal{F}_2$ (global).



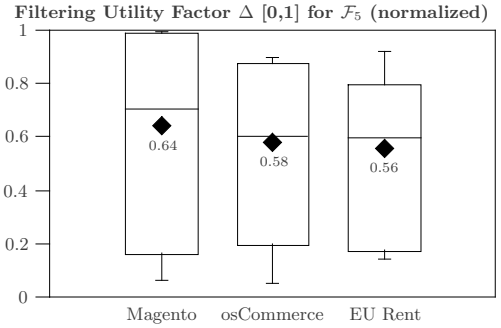
(d) Filtering utility factor for $\mathcal{F}_3$.



(e) Filtering utility factor for $\mathcal{F}_4$.



(f) Filtering utility factor for $\mathcal{F}_5$.



(g) Filtering utility factor for $\mathcal{F}_5$ (normalized).

Figure 7.17. Effectiveness analysis.

7.5.2 Efficiency

It is clear that a good method does not only need to be useful, but it also needs to obtain the results in an acceptable time according to the user's expectations. To find the time spent by our method it is only necessary to record the time lapse between the request of knowledge, i.e. once a filtering request is executed with the user focus and the rest of components that conform the input of or method, and the obtainment of the filtered schema.

Figure 7.18 presents a set of box plots with the resulting values for the response time (in milliseconds) obtained by an Intel Core 2 Duo 3GHz processor with 4GB of DDR2 RAM after applying the aforementioned filtering requests shown in Fig. 7.18 to the conceptual schemas of the case studies.

The mean value of the response time for the case studies is less than 45 milliseconds, which indicates that the time a user expends waiting for the resulting filtered schema is negligible. Furthermore, the largest observations of the response time in any schema are below 150 milliseconds. It is expected that as the number of projections of relationship types and subsumed generalization relationships to process increases, the response time will increase linearly. However, the resulting times for all the filtering requests of the case studies are short enough for our purpose.

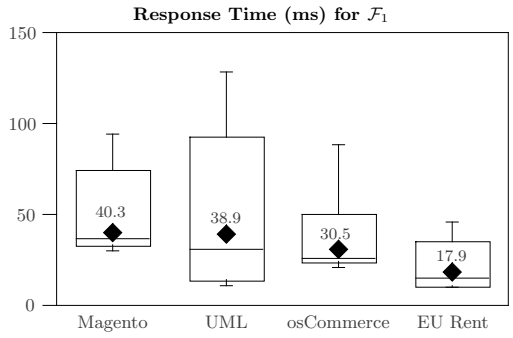
The case of the filtering request for schema rules ($\mathcal{F}_2$) when the scope value is set to local is slightly different, as shown in Fig. 7.18(b). The process behind this filtering request does not need to compute the interest measure for the entity and event types of the large schema. In fact, it only requires to explore the OCL specification of a schema rule in order to extract its referenced elements, and then construct a filtered schema from them. That is the reason why the resulting values for the response time in this filtering request are below 25 milliseconds, with mean values under 5 milliseconds.

7.6 Summary

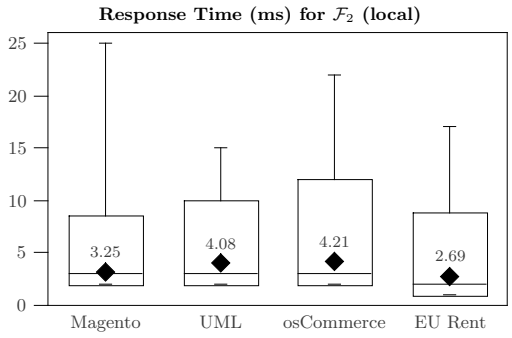
In this thesis, we have focused on the problem of understanding the knowledge defined in large conceptual schemas, in which the elements of interest to a user that are specified within the schema may be very distant from each other and embedded in an intricate web of irrelevant elements for her purpose.

In Ch. 5, we have proposed a filtering method in which a user focuses on a set of elements of focus and the method obtains a filtered schema that includes them and additional elements of high interest to the user. We have implemented our method in a prototype tool and we have evaluated it by means of its application to four large conceptual schemas. The results show that in most cases our method achieves a size reduction greater than 70% in the number of schema elements to explore when using filtered schemas instead of manually exploring the large schema, with an average time per request that is short enough for the purpose at hand.

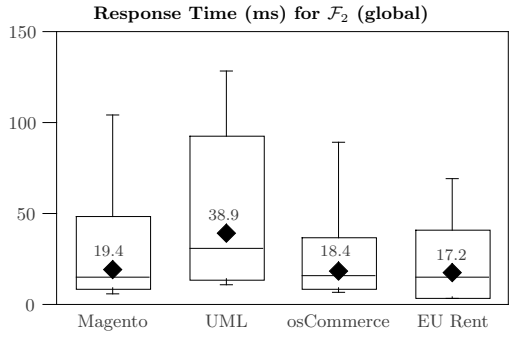
In the following chapter we present the details about the implementation of our filtering method, including the filtering requests and a web-based infrastructure of components that support users that need to explore a large conceptual schema through our filtering approach.



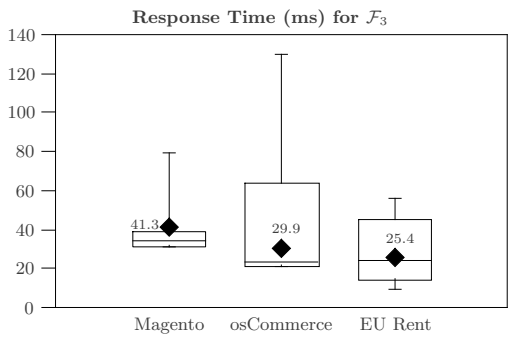
(a) Response time for $\mathcal{F}_1$.



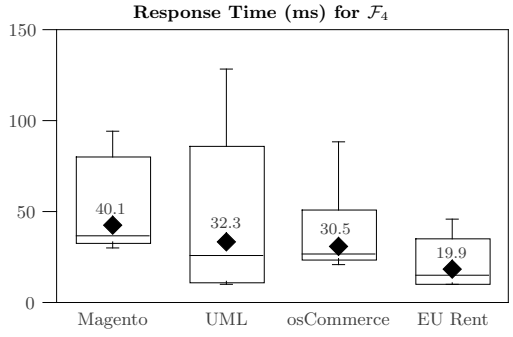
(b) Response time for $\mathcal{F}_2$ (local).



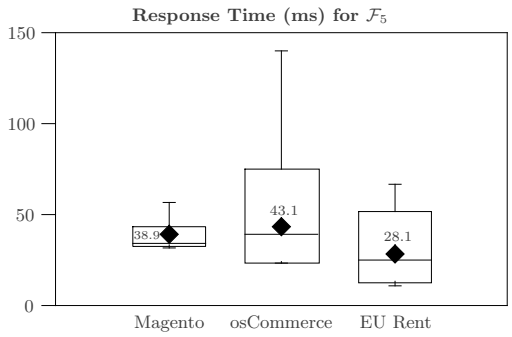
(c) Response time for $\mathcal{F}_2$ (global).



(d) Response time for $\mathcal{F}_3$.



(e) Response time for $\mathcal{F}_4$.



(f) Response time for $\mathcal{F}_5$.

Figure 7.18. Efficiency analysis.

The Web does not just connect machines,
it connects people.

Tim Berners-Lee

8

Web-based Filtering Engine for Large Conceptual Schemas

The chapter focuses on the design and development of a web-based and service-oriented implementation of the filtering methodology introduced in this thesis.

Section 8.1 presents a brief explanation about the motivation of providing a web-based implementation of the filtering engine. Section 8.2 introduces basic concepts about a service-oriented architecture and development. Concretely, we explore the Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL). The combination of these web technologies and the prominent use of existing development frameworks allows us to easily transform a desktop-oriented application into a web service. Section 8.3 describes the details about our web-based filtering engine, including its web architecture and available filtering components. An important component of the filtering engine is the set of web services that deal with the different activities of the filtering process. We provide an explanation of these services, including the schema manager, the service that computes the relevance methods, the service that filters the schema, and the service that provides schema visualization in order to produce the resulting feedback to the user. Section 8.4 presents a set of interaction patterns that a user must follow in order to use our filtering engine. Section 8.5 describes a web-based implementation for our six filtering requests presented in Ch. 6. Finally, Sect. 8.6 summarizes the chapter.

8.1 Motivation

The World Wide Web is growing and its value and utilization as a powerful tool has become evident to all kind of users. In our field, users of large conceptual schemas usually need to extract a portion of interest of the schema and to share it with other stakeholders in order to check and validate that the knowledge it specifies is correct, and make changes whenever necessary. The development of a web-service oriented implementation of our filtering engine provides several benefits to users over traditional desktop-based solutions. Following is a list of these benefits:

- **Interoperability:** This is the most important benefit of a web-service based tool. Typically, web services work outside of private networks, offering developers a non-proprietary route to their solutions. In addition, the use of standard-based communications methods implies that web services are virtually platform-independent. Therefore, developers can use their preferred programming languages to interact with them.

- **Reusability:** Web services allow the business logic of many different systems to be exposed over the web. This gives your applications the freedom to chose the web services that they need to complete their functionality instead of re-inventing the wheel. Web services are self-describing software modules which encapsulates discrete functionality. Therefore, web services are loosely coupled applications and can be used by other applications developed in any technologies.

- **Ubiquity:** Desktop applications are confined to a physical location and hence have usability constraints. On the other hand, web-based development makes it convenient for the users to access the application from any location —computers and mobile devices— using the internet.

- **Maintainability:** Web based applications are directly accessed through internet, whereas desktop applications require to be installed separately on each computer. Also updating the application is cumbersome with desktop applications as it needs to be done on every single computer which is not the case with web applications.

- **Accessibility:** Web services can be set up to be accessible from anywhere in the world which allows subscribers of the service the freedom of choosing how and when they would like to utilize the functionalities provided by the service. Not only does having web-enabled services allow you to reach a broader audience, it also gives you the freedom of allowing your users access to relevant, up-to-date information on demand. Every person with an internet connection and a web-browser is a potential user of your service.

With all these benefits in mind, we believe that the design and development of a web-based and service-oriented implementation of the filtering methodology increases the usability of our filtering engine to the users that want to understand the specific characteristics included in a large conceptual schema. In the following, we describe the details of our development approach to achieve this goal.

8.2 Service-Oriented Architecture

In software engineering, a Service-Oriented Architecture (SOA) is a set of principles and methodologies for designing and developing software in the form of interoperable services. These services are well-defined functionalities that are built as software components (discrete pieces of code and/or data structures) that can be reused for different purposes [41].

The W3C defines a web service as a software system designed to support interoperable machine-to-machine interaction over a network [136]. In essence, a web service is an application functionality packaged as a single unit and exposed to the network [34]. It provides a technology for application integration and interoperability based on open standards [27]. The web services framework is divided into four areas:

- **Service Processes:** This part of the architecture generally involves more than one web service. For example, discovery belongs in this part of the architecture, since it allows us to locate one particular service from among a collection of web services.
- **Service Description:** One of the most interesting features of web services is that they are self-describing. This means that, once you've located a web service, you can ask it to 'describe itself' and tell you what operations it supports and how to invoke it. This is handled by the Web Services Description Language (WSDL).
- **Service Invocation:** Invoking a web service involves passing messages between the client and the server. The Simple Object Access Protocol (SOAP) specifies how we should format requests to the server, and how the server should format its responses. In theory, we could use other service invocation languages (such as XML-RPC). However, SOAP is by far the most popular choice for web services.
- **Transport:** Finally, all these messages must be transmitted somehow between the server and the client. The protocol of choice for this part of the architecture is the HyperText Transfer Protocol (HTTP) [44], the same protocol used to access conventional web pages on the internet. Again, in theory we could be able to use other protocols, but HTTP is currently the most used one.

8.2.1 Web Services Description Language (WSDL)

As communications protocols and message formats are standardized in the web community, it becomes increasingly possible and important to be able to describe the communications in some structured way. The Web Services Description Language (WSDL) addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication [135].

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network

deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Figure 8.1 summarizes the structure of a WSDL specification file.

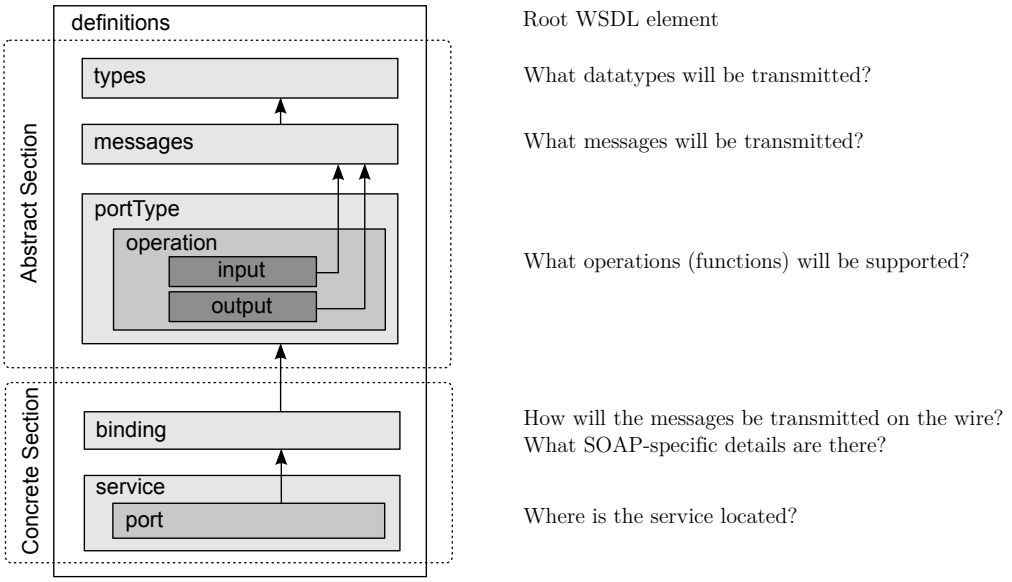


Figure 8.1. Structure of WSDL specification.

8.2.2 Simple Object Access Protocol (SOAP)

The Simple Object Access Protocol (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics. Figure 8.2 shows an example of SOAP message [137].

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header></soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>AAPL</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Figure 8.2. Example of SOAP message.

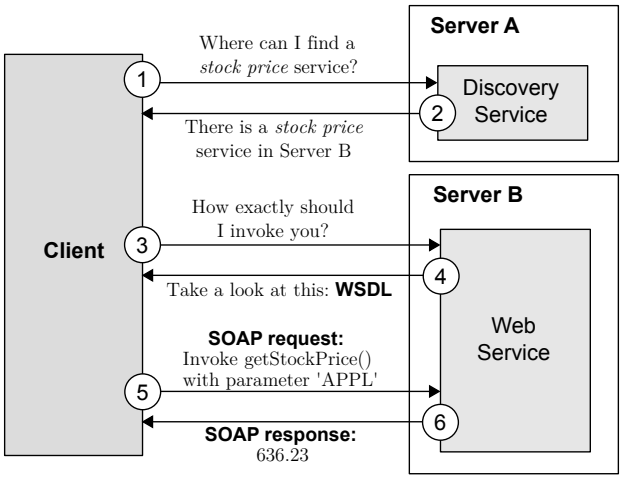


Figure 8.3. Example of typical web service invocation.

8.2.3 Web Service Invocation

Figure 8.3 depicts the steps involved in a complete web Service invocation. The details of these steps are as follows:

1. As aforementioned, a client may have no knowledge of what web service it is going to invoke. The first step is to **discover a web service** that meets the requirements of the client. For instance, a client might be interested in locating a public web service which can give the stock price of an organization in the NASDAQ stock market. It is possible to do this by contacting a discovery service (which is itself a web service).

2. The discovery service will reply, telling the client **what servers** can provide the service it requires.

3. Now, the client knows the location of a web service, but it has no idea of how to actually invoke it. The client knows the service can give it the stock price for an organization according to the NASDAQ market, but ignores how to perform the actual service invocation. The method the client has to invoke might be called `getNasdaqPrice(String name, Date time):String`, but it could also be called `getStockPrice(String stockName):Real`. The client has to ask the web service to **describe** itself (i.e. tell the client how exactly it should invoke the functionalities of the service).

4. The web service replies to the client with the specification in **WSDL** format of the available operations it provides.

5. The client finally knows where the web service is located and how to invoke it. The invocation itself is done in through the SOAP protocol. Therefore, the client will first send a **SOAP request** asking for the stock price of a certain organization.

6. The web service will kindly reply with a **SOAP response** which includes the stock price the client asked for, or maybe an error message if the SOAP request was incorrect.

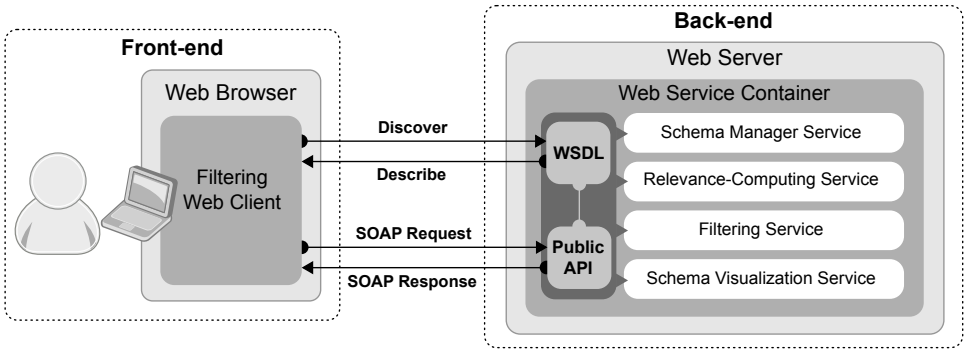


Figure 8.4. Web architecture of the filtering engine.

8.3 Web Architecture of the Filtering Engine

The architecture of our filtering engine follows the principles of a basic service-oriented architecture. Figure 8.4 illustrates the components of the engine. It shows a service consumer at the left sending a service request message to a service provider at the right. The service provider returns a response message to any request of the service consumer. The request and subsequent response connections are defined in SOAP, which is understandable to both the service consumer and service provider.

The service consumer works as the front-end of our filtering engine. It is a filtering web client that runs on a web browser and receives the user interaction. The client accesses the available operations our filtering engine provides in its back-end component. The details about the interaction patterns the web client allows to a user that wants to explore a large conceptual schema by using our filtering methodology are described in Sect. 8.4.

On the other side, the back-end of our filtering engine consists of a web server that contains a web service container, which hosts the core services of the engine. In our prototype, we use Apache Axis 2 [2] as the web service container, which runs on an Apache Tomcat [3] application web server. We provide a WSDL specification of the available operations of the core web services that allow to filter a large conceptual schema. These operations conform the public application programming interface (API) of the engine, which can be used by any service consumer (or web client). In this case, the filtering API is addressed to fulfill the requirements of the filtering web client in the front-end of the engine, although any web client with interest on filtering a large schema could benefit from it.

The backbone of our filtering engine comprises four web services that are responsible for performing the different tasks that are part of the filtering methodology introduced in Ch. 5 and Ch. 6. The implementation of these core web services follows a bottom-up approach, as explained in [99]. By using the web service development assistant provided by the Web Tools Platform project [39] inside the Eclipse development framework [38], we easily obtain a complete web service, including runtime components and WSDL specification, from a Java class that contains the implementation of the public operations of the filtering engine API. The details of these core services are presented in the following sections.

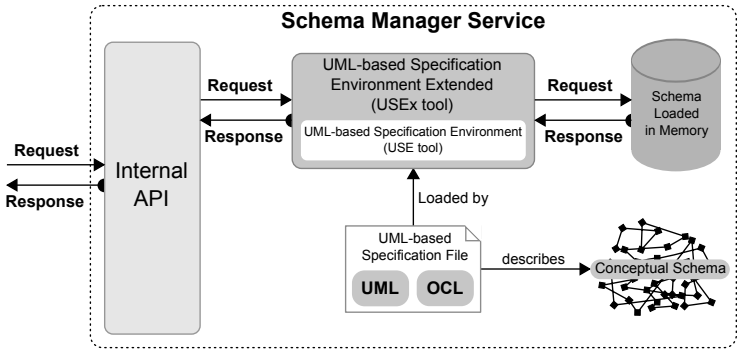


Figure 8.5. Internal structure of the schema manager service.

8.3.1 Schema Manager Service

The schema manager service is one of the most critical web services that provides functionality to our filtering engine. Figure 8.5 presents the internal structure of this service. In order to extract a portion of the knowledge contained in a large conceptual schema of interest to a user, our engine needs an intermediate component to deal with the representation of the schema of context. To achieve this goal, we have extended an existing tool that manages conceptual schemas expressed in UML/OCL and provides access to the elements they contain: the USE (UML-based Specification Environment) tool [49].

The USE system supports developers in analyzing the structure (classes, associations, attributes, and invariants) and the behavior (operations and pre- and postconditions) of a conceptual schema by generating typical snapshots (system states) and by executing typical operation sequences (scenarios). Developers can formally check constraints (invariants and pre- and postconditions) against their expectations and can, to a certain extent, derive formal properties about the schema. A UML/OCL conceptual schema is given to USE in textual form by a USE specification file (see Fig. 8.5). Then, USE compiles the textual specification and creates a tree-hierarchy representation of the corresponding schema elements, which are allocated in memory and can be accessed by using the USE available operations.

Since USE only supports a subset of the UML, we have extended it in order to be able to process additional constructions that are used in this thesis. Concretely, we have developed a project fork named USEx (UML-based Specification Environment Extended) that allows the specification of default expressions in OCL for attributes and binary associations, derivation rules for attributes and binary association ends, the declaration of event types in the same way as entity types, and the inclusion of custom data types. Additionally, USEx provides ways to specify generalization sets (with disjointness and completeness constraints), and to indicate specific multiplicities of attributes.

In order to open the access to the extended UML constructions we have extended the operations of USE and we have implemented new operations in USEx to query the conceptual schema. The schema manager service provides its own internal API that contains these operations, which are required by the other core services of the web-based filtering engine.

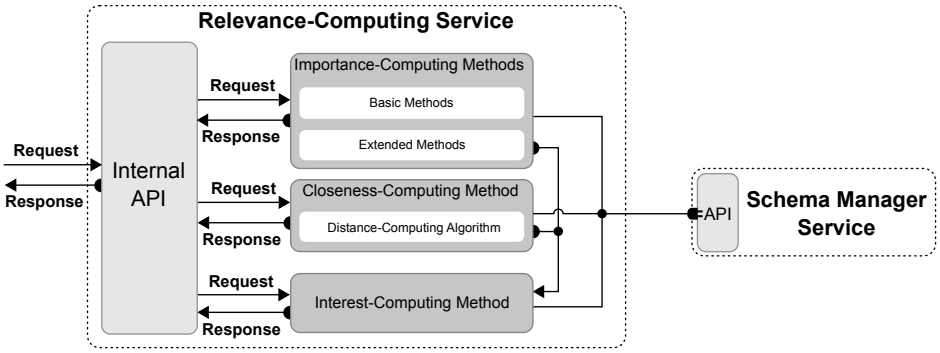


Figure 8.6. Internal structure of the relevance-computing service.

8.3.2 Relevance-Computing Service

Once the engine contains a loaded conceptual schema that can be accessed by other services, it is time to work with it. The relevance-computing service assists the filtering engine on computing the metrics and methods introduced in Ch. 4. Figure 8.6 presents the internal structure of this core web service.

The first component of the service deals with the implementation and analysis of the importance-computing methods presented in Sect. 4.3 of Ch. 4. This component includes the basic methods adapted from the existing literature to be used with UML/OCL schemas, and also the extension of these methods to incorporate the analysis of additional knowledge extracted from the analysis of OCL expressions, reification of association classes, and multiplicities of schema elements. To support the extended methods, we have developed OCL crawlers to traverse and explore the explicit OCL expressions included in the large conceptual schema, and the implicit OCL expressions obtained by reification and conversion of graphical constraints (multiplicities, disjointness and completeness of generalization sets).

The second component of the service deals with the implementation and analysis of the closeness-computing method presented in Sect. 4.5 of Ch. 4. This component includes a distance-computing algorithm that incrementally obtains the topological distance between a pair of entity types through relationship types and generalization relationships.

The last component of the service deals with the implementation and analysis of the interest-computing method presented in Sect. 4.6 of Ch. 4. This component combines the resulting values of the aforementioned importance- and closeness-computing components in order to obtain the interest of an entity or event type with respect to a set of schema elements of focus, as presented in Ch. 5.

All three components extensively use the API provided by the schema manager service in order to obtain the elements included in the loaded conceptual schema, the relationships and information described between them, the schema rules in order to go through their tree structure of OCL expressions, among others. With all the functionality provided by this service, the three components can retrieve the required knowledge to compute the metrics that are the backbone of our filtering methodology

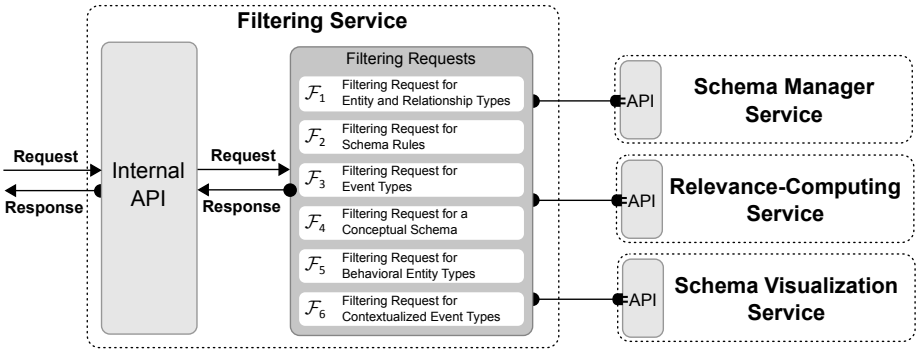


Figure 8.7. Internal structure of the filtering service.

8.3.3 Filtering Service

The filtering service deals with the guidance of the main filtering process to automatically extract the fragment of interest to the user of the large conceptual schema. Figure 8.7 presents the internal structure of this core web service. It contains the implementation of the catalog of filtering requests we studied in Ch. 6:

- $\mathcal{F}_1$: Filtering request for entity and relationship types, as described in Sect 6.3.1 of Ch. 6.
- $\mathcal{F}_2$: Filtering request for schema rules, as described in Sect 6.3.2 of Ch. 6.
- $\mathcal{F}_3$: Filtering request for event types, as described in Sect 6.3.3 of Ch. 6.
- $\mathcal{F}_4$: Filtering request for a conceptual schema, as described in Sect 6.3.4 of Ch. 6.
- $\mathcal{F}_5$: Filtering request for context behavior of entity types, as described in Sect 6.3.5 of Ch. 6.
- $\mathcal{F}_6$: Filtering request for contextualized types, as described in Sect 6.3.6 of Ch. 6.

Once a user requests to filter a large conceptual schema, the filtering service processes the request and passes the user input to the specific filtering request that must be used to obtain the desired knowledge that will be part of the resulting filtered conceptual schema, which conforms the expected feedback for the user. To produce the output, the filtering service makes use of the internal APIs of the other core web services.

The filtering service connects with the schema manager service in order to access and explore the elements of the large conceptual schema that may be part of the result. In addition to it, the relevance-computing service provides the necessary measures to complete the knowledge in the user input with additional knowledge with high importance and that is close to the user focus on the large schema. Finally, once the filtered conceptual schema is constructed, the filtering service makes use of the schema visualization service in order to graphically present it to the user.

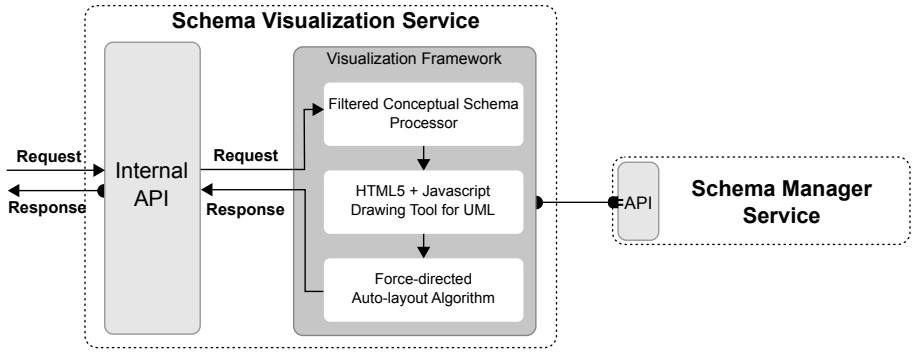


Figure 8.8. Internal structure of the schema visualization service.

8.3.4 Schema Visualization Service

The last web service that conforms the core of the filtering engine is the schema visualization service. Figure 8.8 presents the internal structure of this specific web service. It deals with the graphical representation of the filtered conceptual schema that results from the application of our filtering methodology to a large conceptual schema.

The visualization framework within the schema visualization service contains three components. First one processes the filtered conceptual schema in order to analyze how to present it to the user in a pleasant way to increase its understandability. The main idea is to reduce the effort a user needs to identify the elements of focus and explore the additional knowledge that complements them.

Then, the second component is a lightweight HTML5/Javascript library for UML 2 diagramming. It allows the developer to easily embed UML diagrams in web applications, just invoking a few javascript methods [96]. We have extended this library with additional features to cover a wide range of UML constructions and increased the documentation of the drawing tool [50]. Since UML has become a de-facto standard for modeling, many tools are available to allow the modeler to draw the structural and behavioral constructs that UML provides. However, these tools are mostly implemented to be run on your own computer. Our drawing tool however provides an online visualization of the filtered schema that can be remotely shown by freely accessing the filtering engine in a web browser.

The last component deals with the correct placement of the elements of the filtered schema. We see the schema as a graph whose entity and event types are the nodes, while the edges are the relationships between them. We use a force-directed auto-layout algorithm that computes the position of the set of edges and nodes [7]. The algorithm assigns forces as if the edges were springs and the nodes were electrically charged particles. The entire graph is then simulated as if it were a physical system. The forces are applied to the nodes, pulling them closer together or pushing them further apart (according to Hooke's attraction law and Coulomb's repulsion law [52]). This is repeated iteratively until the system comes to an equilibrium state, i.e. their relative positions do not change anymore from one iteration to the next. At that moment, the graphical representation of the filtered schema is ready.

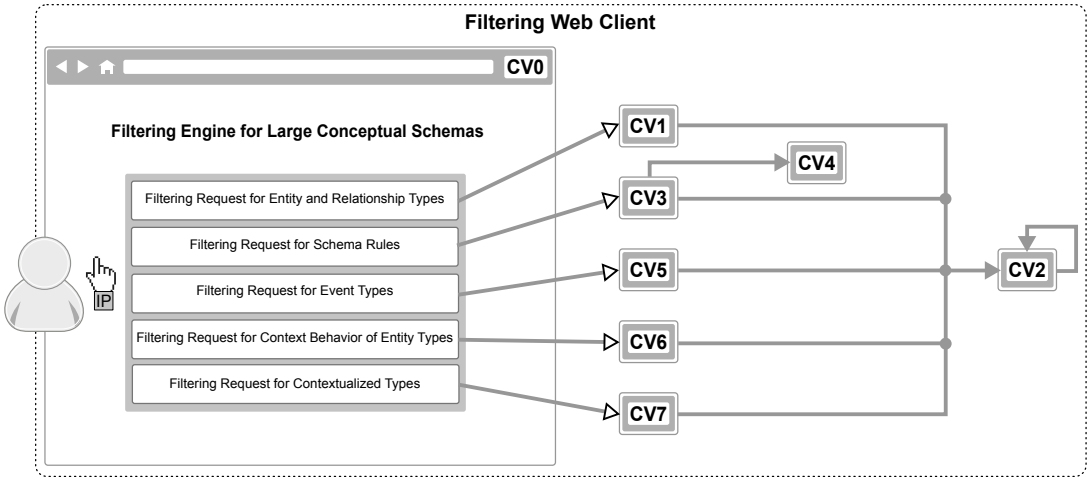


Figure 8.9. General structure of the filtering web client and interaction with client views.

8.4 User Interaction

The interaction of the user with our web-based filtering engine is performed through a web client that uses the corresponding API of the core web services within the filtering engine. As aforementioned, the user only requires a web browser to make use of the filtering client and then explore the knowledge specified in a large conceptual schema.

Our filtering web client provides a minimal consumer of the API provided by the back-end part of the filtering engine. Figure 8.9 presents the different views that are part of the client and the navigation relationships between them. It contains eight client views. Each one contains interaction points that change the behavior of the view in order to construct a focus set or to request the execution of a specific filtering request. We name CV to each client view contained in the filtering web client, and IP to each interaction point included within each of the CVs.

A user of our service starts with CV0 and selects one of the five alternatives to start the filtering process with a large conceptual schema. Each of the filtering proposals lead the user to the corresponding view —CV1, CV3, CV5, CV6, or CV7. These views are responsible of constructing the focus set and execute the filtering requests $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$, $\mathcal{F}_5$ and $\mathcal{F}_6$, respectively. Then, each view shows the resulting filtered conceptual schema in CV2, which can also start a new iteration of the filtering method by executing the filtering request $\mathcal{F}_4$. Additionally, CV4 shows the details of a specific schema rule selected in CV3.

In the following, we detail the user interface of each CV and the expected interaction pattern of a user of the web client. Section 8.4.1 presents the interaction of a user that focus on entity and relationship types. Section 8.4.2 deals with the interaction that focus on schema rules and Sect. 8.4.3 indicates the interaction centered on event types. In addition, Sect. 8.4.4 explains the functionality to filter from a conceptual schema of small size. Finally, Sect. 8.4.5 presents the interaction with the filtering request for context behavior of entity types, and Sect. 8.4.6 describes the interaction to contextualize types.

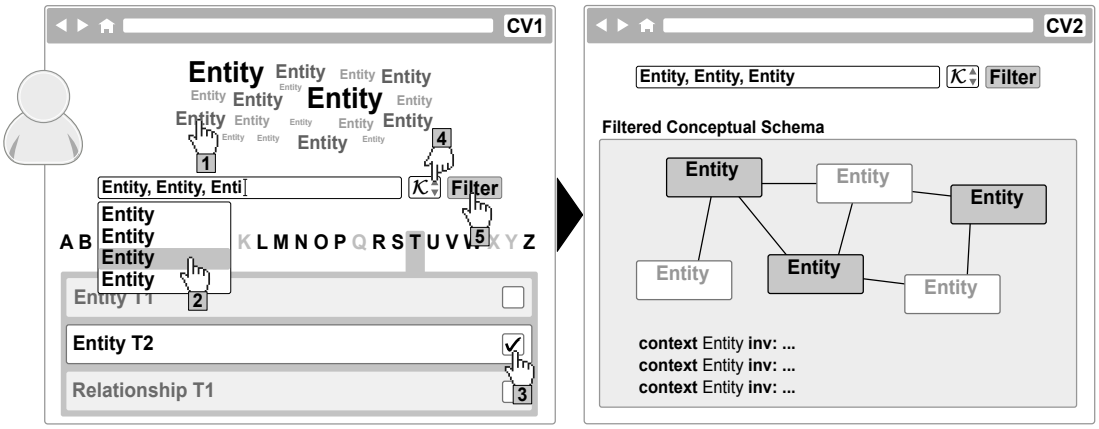


Figure 8.10. Interaction pattern of the filtering request for entity and relationship types ($\mathcal{F}_1$).

8.4.1 Filtering Request $\mathcal{F}_1$ – Interaction Pattern

The first filtering request deals with a filtering interaction centered on the entity and relationship types from a large schema. The method obtains a small-size filtered conceptual schema that includes the combination of the entity and relationship types in the user focus with the elements of interest gathered by our filtering methodology.

A user interested in using this request starts the interaction in CV1, as shown in Fig. 8.10. CV1 provides a word cloud [106] of the entity types of the large schema. A word cloud of entity types is a visual representation where more relevant entity types are depicted in a larger font. This format is useful for quickly perceiving the most prominent entity types by inexperienced users. IP1 indicates that selecting a single entity type name within the word cloud, includes such entity type in the focus set. The names of the entity and relationship types of focus are shown in the search bar of CV1.

Alternatively, the user may type the name of the desired entity or relationship types of focus in the search bar, which provides an auto-completing functionality that helps discovering the names of the existing types of the schema of context in IP2. In addition to it, CV1 provides an alphabetical list to explore the types of the schema. According to IP3, the user can select a single letter of the English alphabet from the list and then obtains an enumeration of those entity and relationship types whose name starts with such selected letter. The user may select the entity or relationship types of interest, which are then included in the search bar.

Also, the user may change the expected size $\mathcal{K}$ of the resulting filtered schema through the spin box widget of IP4. Note that the minimum value of $\mathcal{K}$ equals to the size of the set of selected entity types (including those entity types that are participants of selected relationship types). Therefore, this value changes whenever the user selects/deselects elements. Finally, once the user has selected the elements of focus, she presses the filter button (IP5) that completes the interaction and starts the request to the specific operation of the web service that implements this filtering request. The response is presented in CV2, which includes the graphical representation of the filtered conceptual schema.

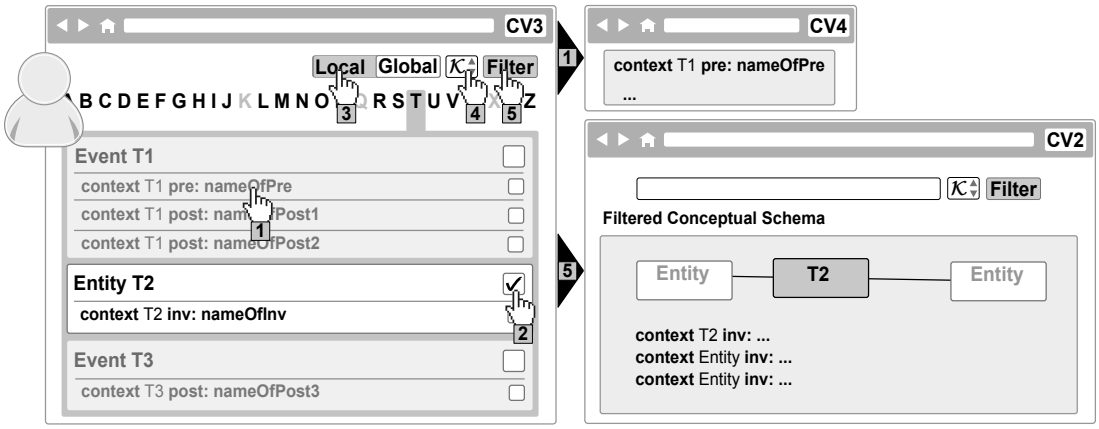


Figure 8.11. Interaction pattern of the filtering request for schema rules ($\mathcal{F}_2$).

8.4.2 Filtering Request $\mathcal{F}_2$ – Interaction Pattern

The second filtering request deals with a filtering interaction centered on the schema rules from a large schema. The method obtains a small-size filtered conceptual schema that includes the combination of the elements referenced by the schema rules in the user focus with the elements of interest gathered by our filtering methodology.

A user interested in using this request starts the interaction in CV3, as shown in Fig. 8.11. CV3 provides an alphabetical list to explore the schema rules of the schema. The user can select a single letter of the English alphabet from the list and then obtains an enumeration of those entity, event, and relationship types whose name starts with such selected letter. The schema rules defined in the context of each of these types are also enumerated. According to IP1, by clicking on the name of a specific rule our client provides its entire OCL specification shown in CV4. In addition to it, each schema rule can be selected to be part of the focus set, as indicated in IP2.

As indicated in Sect. 6.3.2 of Ch. 6, the scope of the filtering request for schema rules can be set to local or global. A *local* value for the scope implies that the resulting filtered conceptual schema will only contain those elements referenced by the schema rules of focus. On the other hand, A *global* value for the scope will include additional knowledge of interest to the user until reaching the size threshold $\mathcal{K}$. The user may change the scope through the switch indicated in IP4.

Also, the user may change the expected size $\mathcal{K}$ of the resulting filtered schema when the scope is set to global through the spin box widget of IP4. Note that the minimum value of $\mathcal{K}$ equals to the size of the set of referenced entity and event types by the schema rules of focus. Therefore, this value changes whenever the user selects/deselects schema rules. Finally, once the user has selected all the schema rules of focus, she presses the filter button (IP5) that completes the interaction and starts the request to the specific operation of the web service that implements this filtering request. The response is presented in CV2, which includes the graphical representation of the filtered conceptual schema.

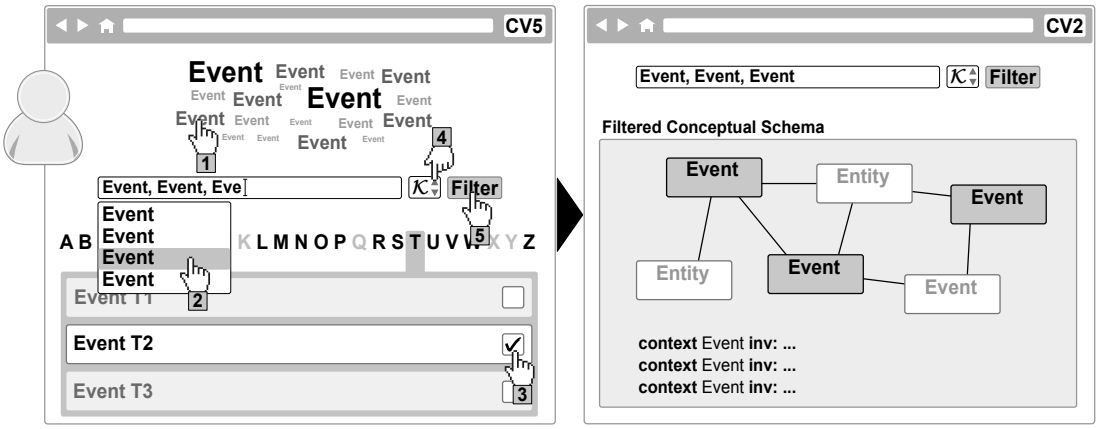


Figure 8.12. Interaction pattern of the filtering request for event types ($\mathcal{F}_3$).

8.4.3 Filtering Request $\mathcal{F}_3$ – Interaction Pattern

The third filtering request deals with a filtering interaction centered on the event types from a large schema. The method obtains a small-size filtered conceptual schema that includes the combination of the event types in the user focus with the elements of interest gathered by our filtering methodology.

A user interested in using this request starts the interaction in CV5, as shown in Fig. 8.12. CV5 provides a word cloud [106] of the event types of the large schema. A word cloud of event types is a visual representation where more relevant event types are depicted in a larger font. This format is useful for quickly perceiving the most prominent event types by inexperienced users. IP1 indicates that selecting a single event type name within the word cloud, includes such event type in the focus set. The names of the event types of focus are shown in the search bar of CV5.

Alternatively, the user may type the name of the desired event types of focus in the search bar, which provides an auto-completing functionality that helps discovering the names of the existing event types of the schema of context in IP2. In addition to it, CV5 provides an alphabetical list to explore the event types of the schema. According to IP3, the user can select a single letter of the English alphabet from the list and then obtains an enumeration of those event types whose name starts with such selected letter. The user may select the event types of focus, which are then included in the search bar.

Also, the user may change the expected size $\mathcal{K}$ of the resulting filtered schema through the spin box widget of IP4. Note that the minimum value of $\mathcal{K}$ equals to the size of the set of selected event types. Therefore, this value changes whenever the user selects/deselects events. Finally, once the user has selected all the event types of focus, she presses the filter button (IP5) that completes the interaction and starts the request to the specific operation of the web service that implements this filtering request. The response is presented in CV2, which includes the graphical representation of the filtered conceptual schema.

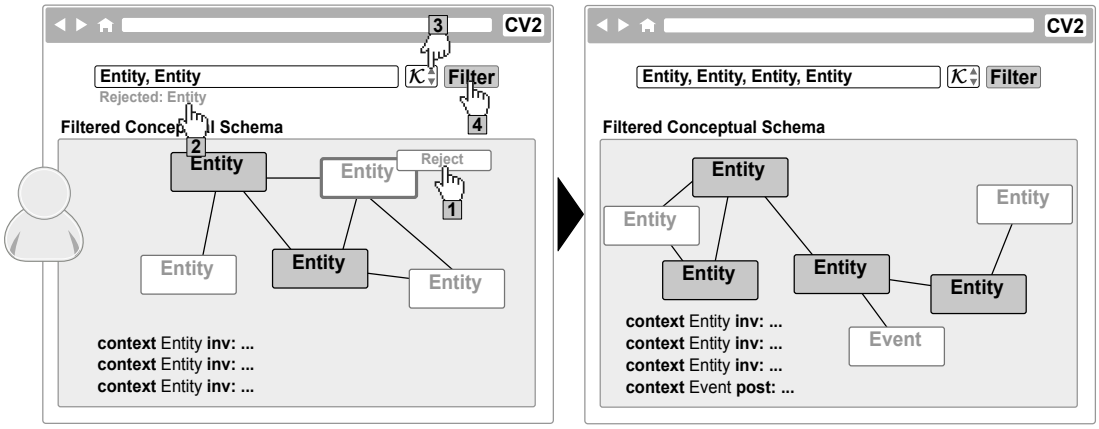


Figure 8.13. Interaction pattern of the filtering request for a conceptual schema ($\mathcal{F}_4$).

8.4.4 Filtering Request $\mathcal{F}_4$ – Interaction Pattern

The fourth filtering request deals with a filtering interaction centered on a small conceptual schema extracted from the large one. The method obtains more knowledge from the large schema with relation to the elements in the user focus. As output, the user obtains a small-size filtered conceptual schema that includes the combination of the small schema in the user focus with the elements of interest gathered by our filtering methodology.

A user interested in using this request starts the interaction with the filtering requests of CV1, CV3, CV5, CV6 or CV7 and obtains a filtered conceptual schema in CV2, as shown in Fig. 8.13. CV2 provides the graphical representation of the conceptual schema, including the OCL specification of schema rules. Then, the user may start a new iteration based on this schema. She can select an element of the filtered schema and include it into the rejection set for the new filtering request, as indicated in IP1. Therefore, the rejected element will not appear in the filtered conceptual schema of the response. To cancel this behavior, the user can delete the rejection of the element by clicking on its name below the search bar, as shown in IP2.

Alternatively, the user may type the name of the desired entity and event types of focus in the search bar, which provides an auto-completing functionality that helps discovering the names of the existing elements of the large schema in IP3. Note that if the user does not include the name of an entity or event type that appears in the current schema, it may appear in the resulting filtered schema according to our method —it does not mean that it is rejected.

Also, the user may change the expected size $\mathcal{K}$ of the resulting filtered schema through the spin box widget of IP4. Note that the minimum value of $\mathcal{K}$ equals to the size of the set of entity and event types that appear in the current schema minus the size of those entity or event types in the rejection set. Therefore, this value changes whenever the user rejects/selects elements. Finally, once the user has selected the elements of focus, she presses the filter button (IP5) that completes the interaction and starts the request to the specific operation of the web service that implements this filtering request. The response is also presented in CV2, which includes the graphical representation of the new filtered conceptual schema.

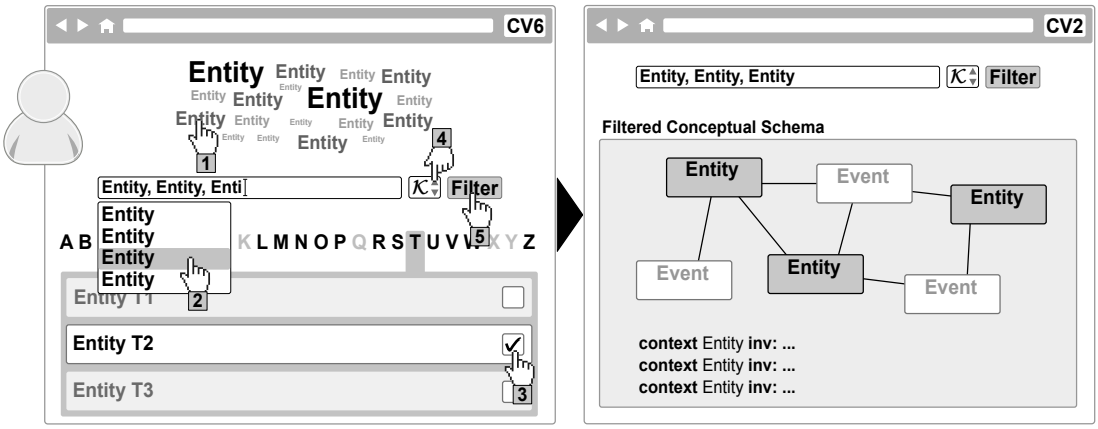


Figure 8.14. Interaction pattern of the filtering request for context behavior of entity types ($\mathcal{F}_5$).

8.4.5 Filtering Request $\mathcal{F}_5$ – Interaction Pattern

The fifth filtering request deals with a filtering interaction centered on the retrieval of the event types related to a subset of the entity types from a large schema. The method obtains a small-size filtered conceptual schema that includes the combination of the entity types in the user focus with the event types of interest gathered by our filtering methodology.

A user interested in using this request starts the interaction in CV6, as shown in Fig. 8.14. CV6 provides a word cloud [106] of the entity types of the large schema. A word cloud of entity types is a visual representation where more relevant entity types are depicted in a larger font. This format is useful for quickly perceiving the most prominent entity types by inexperienced users. IP1 indicates that selecting a single entity type name within the word cloud, includes such entity type in the focus set. The names of the entity types of focus are shown in the search bar of CV6.

Alternatively, the user may type the name of the desired entity types of focus in the search bar, which provides an auto-completing functionality that helps discovering the names of the existing entity types of the schema of context in IP2. In addition to it, CV6 provides an alphabetical list to explore the entity types of the schema. According to IP3, the user can select a single letter of the English alphabet from the list and then obtains an enumeration of those entity types whose name starts with such selected letter. The user may select the entity types of interest, which are then included in the search bar.

Also, the user may change the expected size $\mathcal{K}$ of the resulting filtered schema through the spin box widget of IP4. Note that the minimum value of $\mathcal{K}$ equals to the size of the set of selected entity types. Therefore, this value changes whenever the user selects/deselects entity types. Finally, once the user has selected the entity types of focus, she presses the filter button (IP5) that completes the interaction and starts the request to the specific operation of the web service that implements this filtering request. The response is presented in CV2, which includes the graphical representation of the filtered conceptual schema that shows the most interesting the event types according to the entity types of focus.

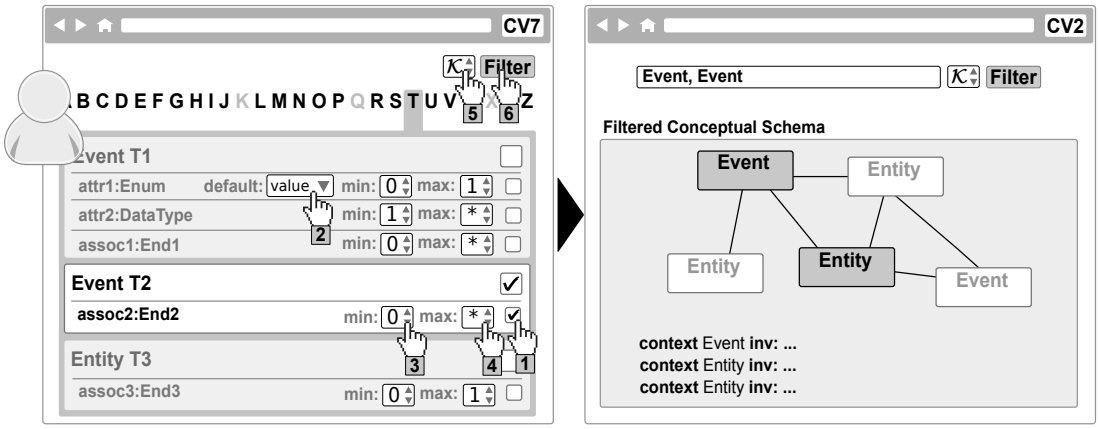


Figure 8.15. Interaction pattern of the filtering request for contextualized types ($\mathcal{F}_6$).

8.4.6 Filtering Request $\mathcal{F}_6$ – Interaction Pattern

The last filtering request deals with a filtering interaction centered on the entity and event types from a large schema. The user contextualizes the entity and event types of focus by means of reducing the characteristics defined over such types.The method obtains a small-size filtered conceptual schema that includes the combination of the entity and event types in the user focus with the elements of interest gathered by our filtering methodology taking into account the contextualization.

A user interested in using this request starts the interaction in CV7, as shown in Fig. 8.15. CV7 provides an alphabetical list to explore the entity and event types of the schema. The user can select a single letter of the English alphabet from the list and then obtains an enumeration of those entity and event types whose name starts with such selected letter. The attributes and relationship types defined in the context of each of these entities and events are also enumerated. According to IP1, the user can select which attributes and relationships are included in the filtering process. The user can delete a defined attribute, whenever the minimum multiplicity of such attribute equals to zero, by not marking the attribute in IP1. In the same way, The user can delete a relationship type whenever the minimum multiplicity in the opposite relationship end equals to zero. In addition to it, the user can select a default literal value for attributes with an enumeration type, as indicated in IP2. Consequently, the user can redefine the multiplicity of an attribute or redefine the multiplicity of the opposite relationship end of an entity or event type, in a relationship type between it and another schema element. To achieve this, the user may change the values of the spin boxes of IP3 and IP4.

Also, the user may change the expected size $\mathcal{K}$ of the resulting filtered schema through the spin box widget of IP5. Note that the minimum value of $\mathcal{K}$ equals to the size of the set of selected entity and event types. Finally, once the user has selected all the types of focus and performed the appropriate contextualization, she presses the filter button (IP6) that completes the interaction and starts the request to the specific operation of the web service. The response is presented in CV2, which includes the graphical representation of the filtered conceptual schema and takes into account the previous contextualization.

Figure 8.16. Screenshot of the main view of the filtering engine prototype tool.

8.5 Web-based Filtering Prototype Tool

We have developed a web-based filtering prototype tool following the guidelines presented in Sect. 8.4. The prototype includes the implementation of the filtering requests from the filtering catalog introduced in Ch. 6 with the aforementioned interaction patterns and a web-service oriented architecture. Our prototype provides functionality to extract a portion of the knowledge of the Magento conceptual schema [94], which is pre-loaded in the schema manager service.

The interaction of the user with our web-based prototype is performed through a web client that works as the entry point to our filtering engine. Figure 8.16 contains links to the five alternatives to start the filtering process with a large conceptual schema. Each alternative lead the user to the corresponding view to construct a focus set and execute the filtering requests $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$, $\mathcal{F}_5$ and $\mathcal{F}_6$, respectively. Note that in our prototype $\mathcal{F}_4$ only applies to filtered schemas obtained after the execution of any of the other filtering requests.

In the following we describe the details about the filtering prototype by showing examples of request/response interactions for any of the filtering requests that conform our filtering catalog.

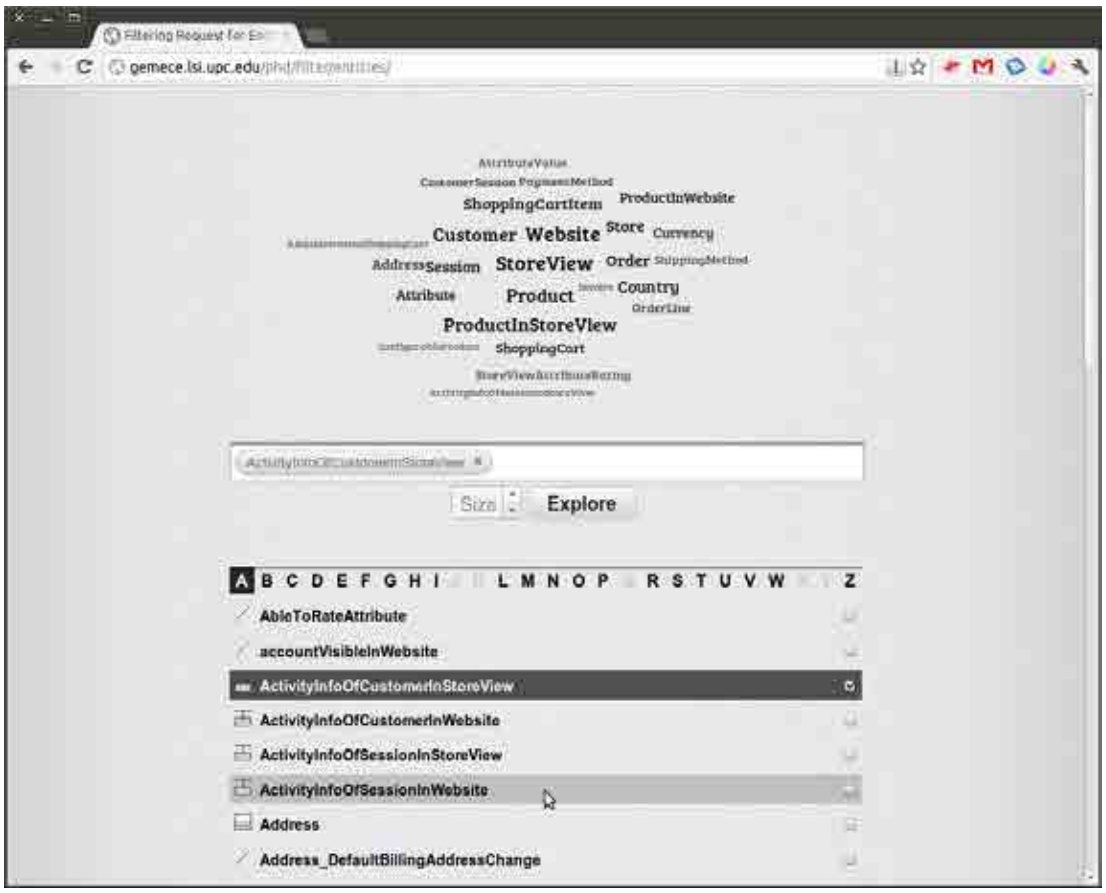


Figure 8.17. Screenshot of $\mathcal{F}_1$ (request) in the filtering engine prototype tool

8.5.1 Filtering Request $\mathcal{F}_1$ – Prototype

The first filtering request focuses on a set of (one or more) entity and relationship types from a large conceptual schema and returns the corresponding filtered schema with the knowledge of interest to the user.

In our implementation we provide users with a web-based interface to ease the construction of the focus set. To this end we follow the interaction pattern described in Sect. 8.4.1 where the user is able to include entity and relationship types into the focus set through three different components. Figure 8.17 shows the web-based interface of our prototype tool for $\mathcal{F}_1$.

First of all, we construct a word cloud with the top-25 most relevant entity types of Magento where more relevant ones are depicted in a larger font size. To compute the word cloud we make use of the importance-computing algorithms presented in Sect. 4.3 of Ch. 4. This format is useful for quickly perceiving the most prominent entity types by inexperienced users. Selecting a single entity type name within the word cloud, includes such entity type in the focus set.

Alternatively, the interface contains a search bar, which provides an auto-completing func-

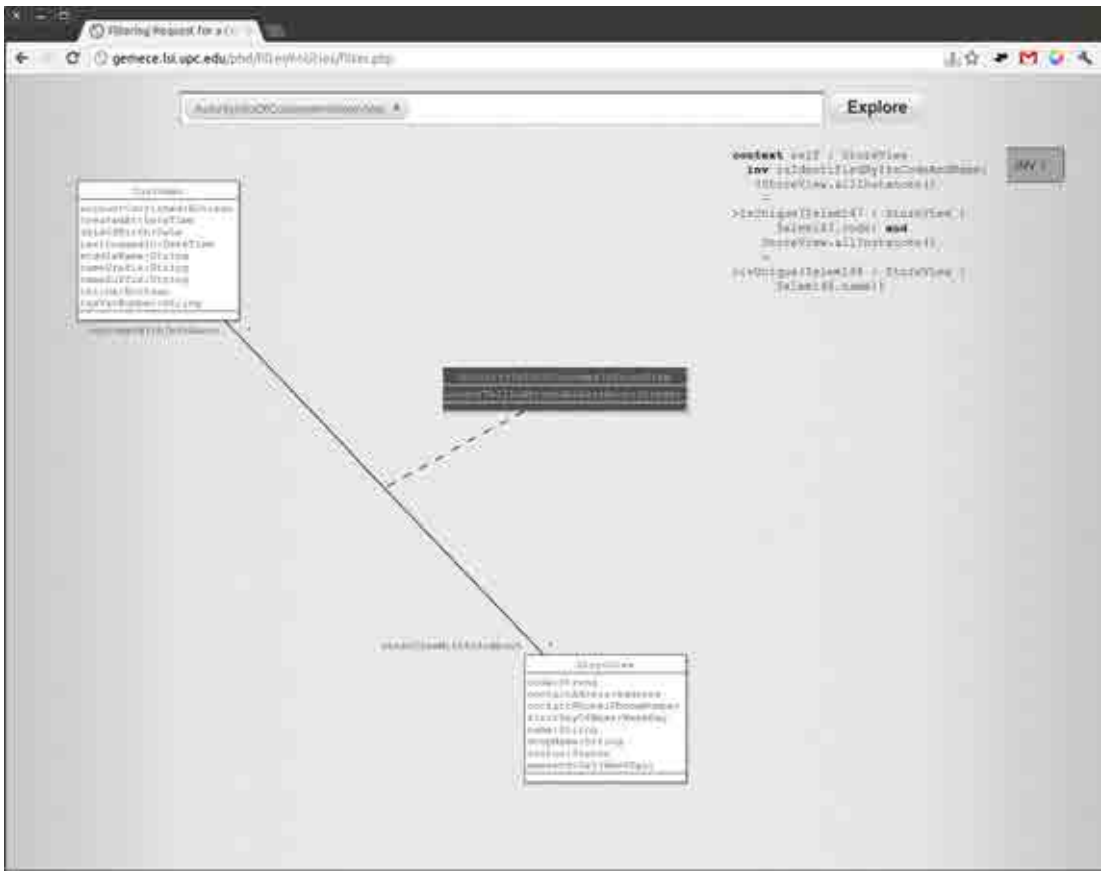


Figure 8.18. Screenshot of $\mathcal{F}_1$ (response) in the filtering engine prototype tool

tionality that helps discovering the names of the existing entity and relationship types of Magento. The names of the selected entity and relationship types are shown in the search bar.

In addition to it, we include an alphabetical list to explore all the entity and relationship types of Magento. The user selects a single letter of the English alphabet from the list and then obtains an enumeration of those entity and relationship types whose name starts with that letter. The user may select the entity or relationship types of interest, which are then included in the search bar. Note that we preceded each item in the alphabetical list with an icon showing the type of the schema element (entity type, relationship type, or association class).

The user may also select the final size of the filtered schema in order to indicate the amount of knowledge she wants to obtain as a result. As an example, Fig. 8.17 shows the filtering prototype when the user selects the association class ActivityInfoOfCustomerInStoreView as the input focus set. By using our prototype (see Fig. 8.18), the user discovers that such association class is a binary association, whose aparticipants are Customer and StoreView. Also, she obtains the rolenames (customerWithInfoAbout and storeViewWithInfoAbout) that may allow her construct additional OCL navigations, the multiplicities (0..* in both sides), the attributes of each element, and a referentially-complete invariant in the context of StoreView.

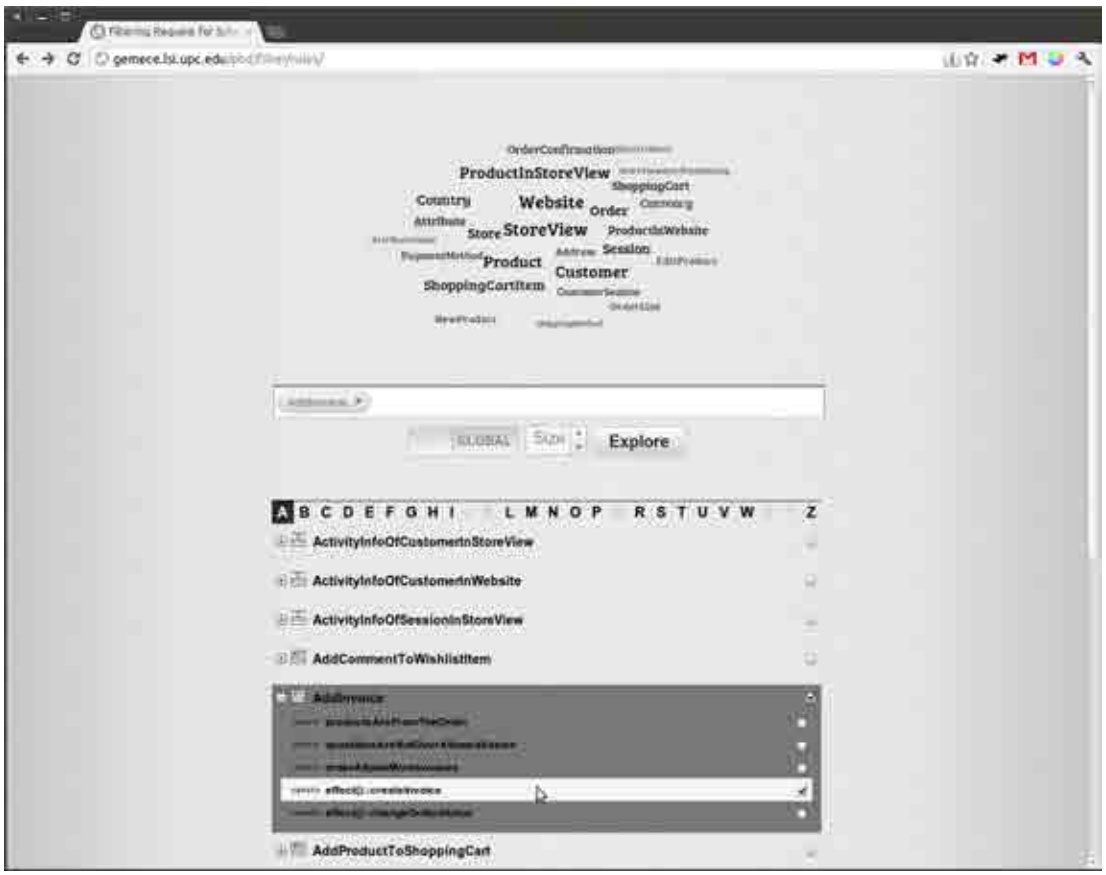


Figure 8.19. Screenshot of $\mathcal{F}_2$ (request) in the filtering engine prototype tool

8.5.2 Filtering Request $\mathcal{F}_2$ – Prototype

The second filtering request focuses on a set of (one or more) schema rules from a large conceptual schema and returns the corresponding filtered schema with the knowledge of interest to the user.

In our implementation we provide users with a web-based interface to ease the construction of the focus set. To this end we follow the interaction pattern described in Sect. 8.4.2 where the user is able to include schema rules into the focus set through three different components. Figure 8.19 shows the web-based interface of our prototype tool for $\mathcal{F}_2$.

First of all, we construct a word cloud with the top-25 most relevant entity types of Magento where more relevant ones are depicted in a larger font size. This format is useful for quickly perceiving the most prominent entity types by inexperienced users. Selecting a single entity type name within the word cloud, includes all the schema rules defined in the context of such entity type in the focus set.

Alternatively, the interface contains a search bar, which provides an auto-completing func-

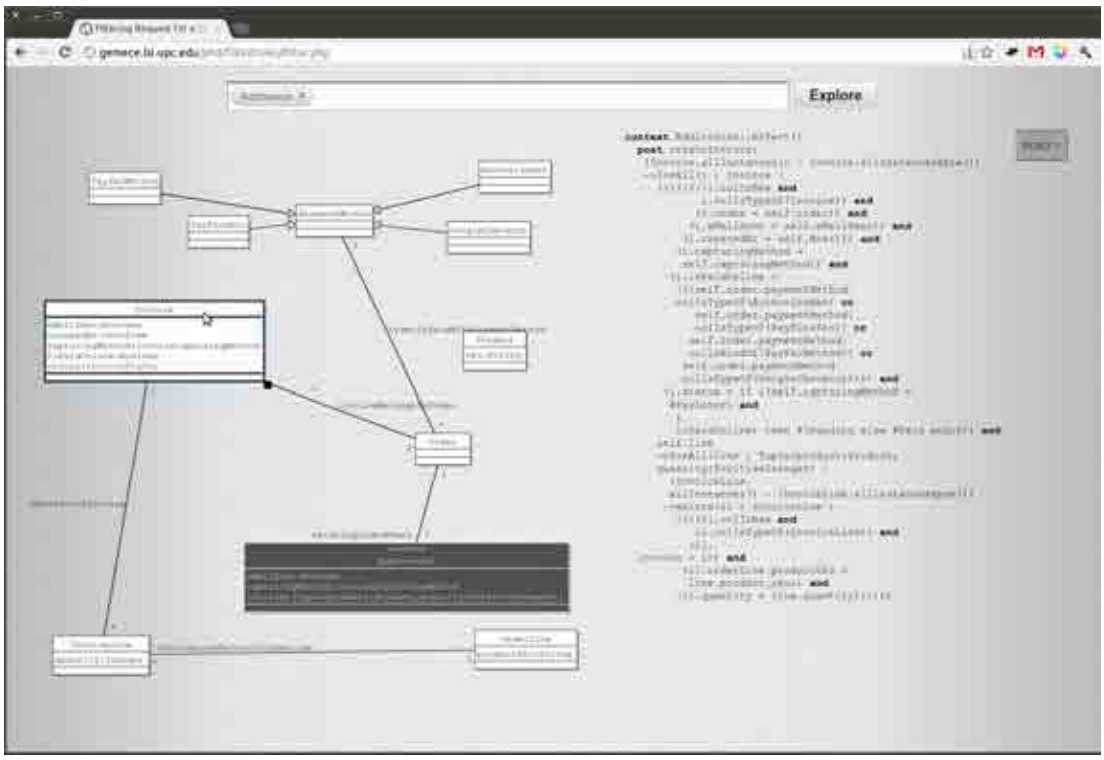


Figure 8.20. Screenshot of $\mathcal{F}_2$ (response) in the filtering engine prototype tool

tionality that helps discovering the names of the existing entity and event types of Magento. The names of the selected entity and event types are shown in the search bar, and all their schema rules are included in the focus set.

In addition to it, we include an alphabetical list to explore all the entity, event, and relationship types of Magento that define schema rules. The user selects a single letter of the English alphabet from the list and then obtains an enumeration of those entity and relationship types whose name starts with that letter. Note that we preceded each item in the alphabetical list with an icon showing the type of the schema element (entity type, event type, relationship type, or association class).

The user may also select the final size of the filtered schema in order to indicate the amount of knowledge she wants to obtain as a result, or alternatively, set the scope of the request to *local* in order to only obtain those elements referenced by the rules of focus. As an example, Fig. 8.19 shows the filtering prototype when the user selects the post-condition `createInvoice` of the `effect()` operation of the event type AddInvoice. By using our prototype (see Fig. 8.20), the user discovers the elements referenced by that schema rule taking into account that the scope was set to *local*. Concretely, the user may discover that the post-condition describes the behavior to generate a new instance of Invoice, including the way to set the values of the attributes of the new Invoice, and how to relate it with new instances of InvoiceLine that match the instances of OrderLine of the current Order of that event.

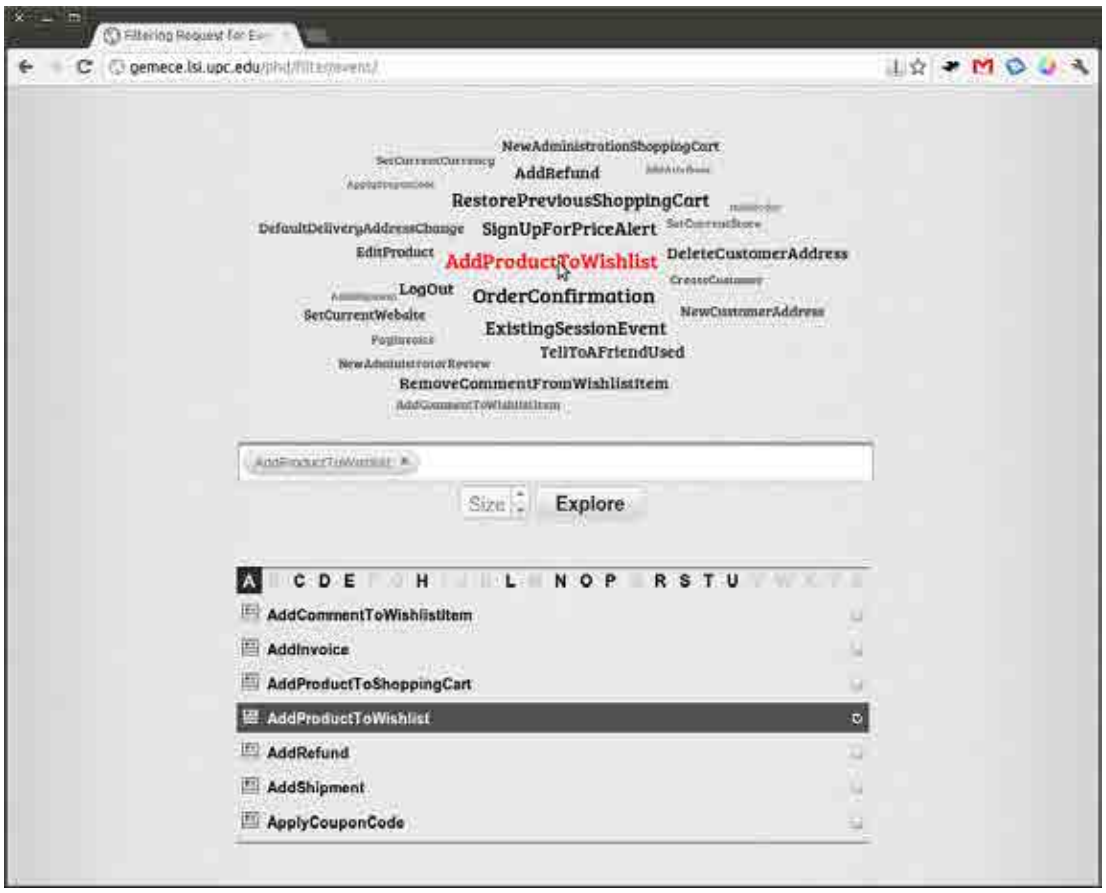


Figure 8.21. Screenshot of $\mathcal{F}_3$ (request) in the filtering engine prototype tool

8.5.3 Filtering Request $\mathcal{F}_3$ – Prototype

The third filtering request focuses on a set of (one or more) event types from a large schema and returns the corresponding filtered schema with the knowledge of interest to the user.

In our implementation we provide users with a web-based interface to ease the construction of the focus set. To this end we follow the interaction pattern described in Sect. 8.4.3 where the user is able to include event types into the focus set through three different components. Figure 8.21 shows the web-based interface of our prototype tool for $\mathcal{F}_3$.

First of all, we construct a word cloud with the top-25 most relevant event types of Magento where more relevant ones are depicted in a larger font size. This format is useful for quickly perceiving the most prominent event types by inexperienced users. Selecting a single event type name within the word cloud, includes such event type in the focus set.

Alternatively, the interface contains a search bar, which provides an auto-completing functionality that helps discovering the names of the existing event types of Magento. The names of the selected event types are shown in the search bar.

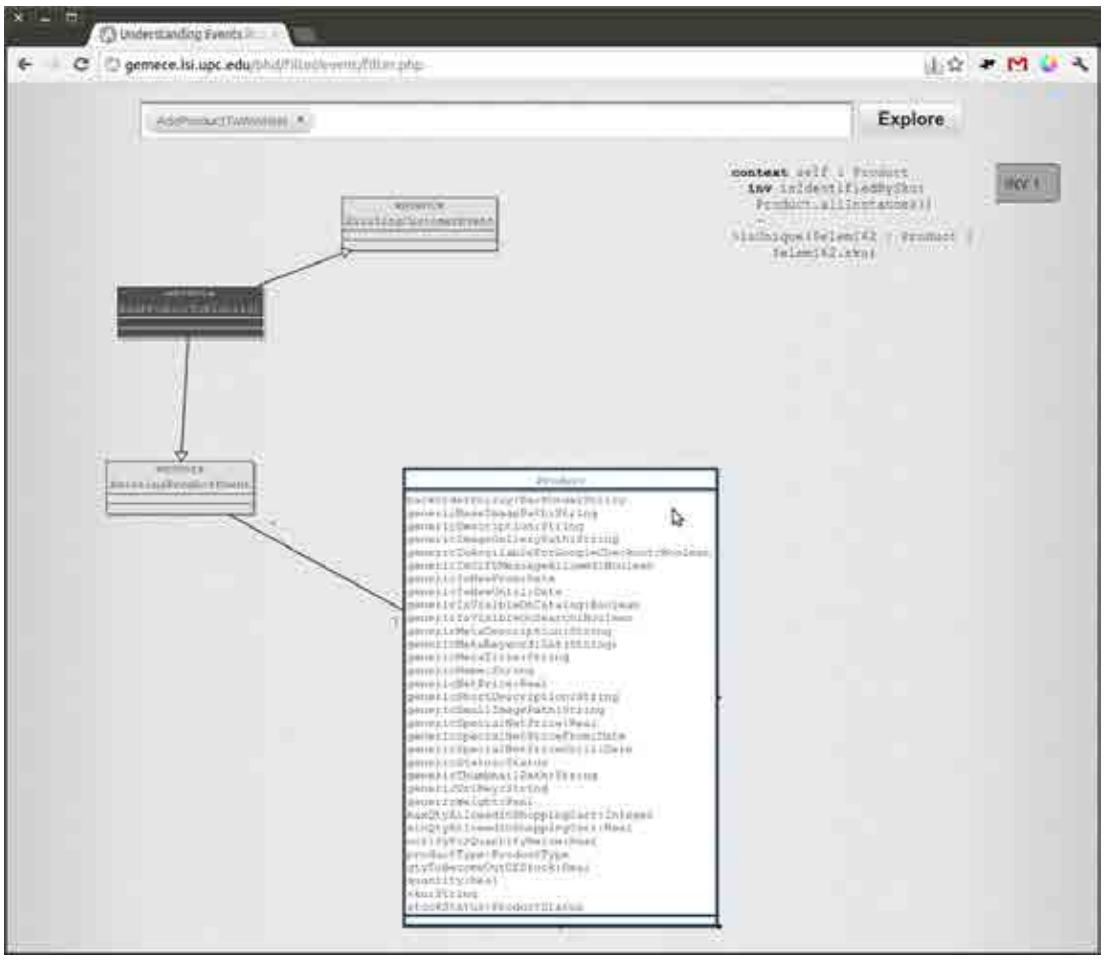


Figure 8.22. Screenshot of $\mathcal{F}_3$ (response) in the filtering engine prototype tool

In addition to it, we include an alphabetical list to explore all the event types of Magento. The user selects a single letter of the English alphabet from the list and then obtains an enumeration of those event types whose name starts with that letter. The user may select the event types of interest, which are then included in the search bar.

The user may also select the final size of the filtered schema in order to indicate the amount of knowledge she wants to obtain as a result. As an example, Fig. 8.21 shows the filtering prototype when the user selects the event type AddProductToWishlist as the input focus set. By using our prototype (see Fig. 8.22), the user discovers that such event type is a descendant from the abstract event types ExistingProductEvent and ExistingCustomerEvent, which indicates that AddProductToWishlist is an event type that deals with products and customers. In fact, the filtered schema shows that ExistingProductEvent is associated to an instance of the entity type Product, and that therefore AddProductToWishlist inherits that association. Also, the user obtains a referentially-complete invariant which states that each instance of Product is identified by the value of the attribute `sku`.

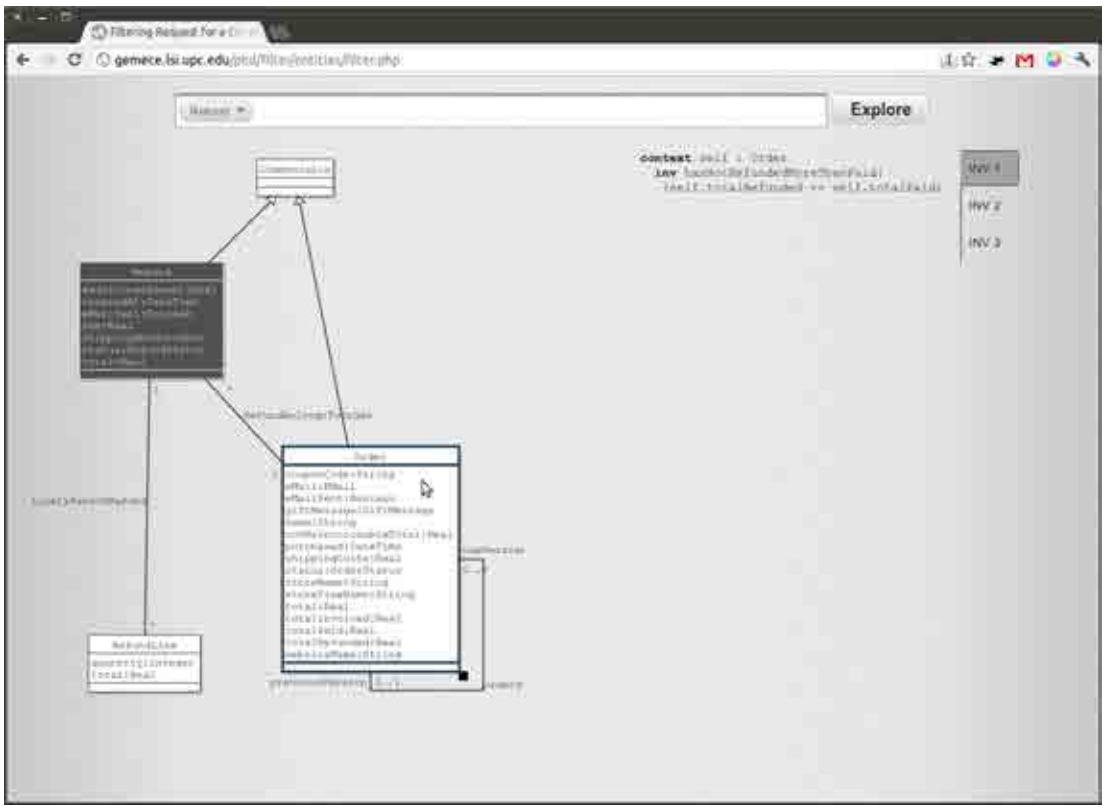


Figure 8.23. Screenshot of $\mathcal{F}_4$ (request) in the filtering engine prototype tool

8.5.4 Filtering Request $\mathcal{F}_4$ – Prototype

The fourth filtering request focuses on a small schema obtained through the application of any of the other filtering requests from the catalog. That small schema is a filtered conceptual schema that the user wants to extend in order to obtain additional knowledge from the original large schema with higher relation to the schema elements contained in the filtered schema.

In our implementation we provide users with a web-based interface to ease the construction of the focus set. To this end we follow the interaction pattern described in Sect. 8.4.4 where the user is able to include elements from a filtered schema into the focus set through two different components. Figure 8.23 shows the web-based interface of our prototype tool for $\mathcal{F}_4$.

First of all, we provide users with an interactive graphical visualization of the filtered conceptual schema through a HTML5/Javascript library. Users are able to directly interact with the filtered schema by changing the position of elements (drag & drop). Also, users may select a set of (one or more) schema elements from the filtered schema and include them into the focus set for the application of the fourth filtering request.

Alternatively, the user may type the name of the desired entity or event types of focus in the search bar, which provides an auto-completing functionality. Therefore, the user can refine

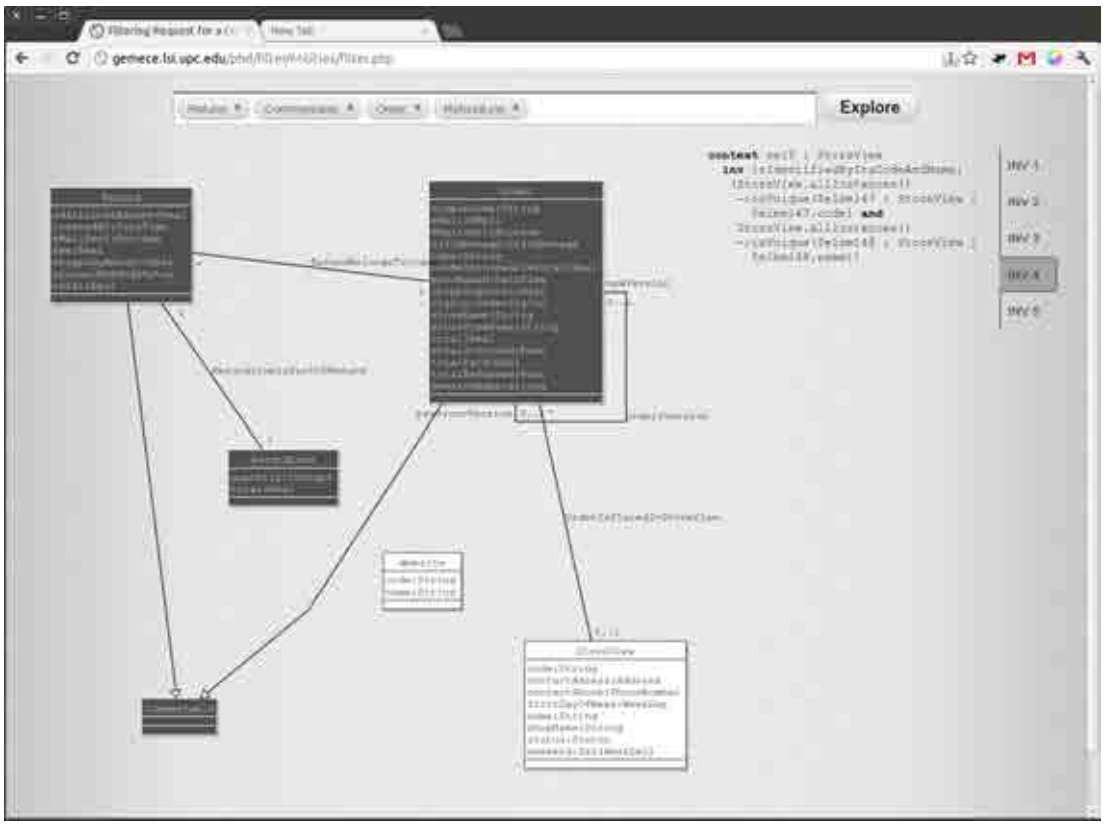


Figure 8.24. Screenshot of $\mathcal{F}_4$ (response) in the filtering engine prototype tool

the focus set in order to combine the selection of elements from the filtered schema with the selection of additional elements from the large schema according to specific information needs.

Once the focus set is complete, the user clicks the *Explore* button to start the new filtering request and obtains the corresponding filtered schema. In the example of Fig. 8.23, the interface shows a filtered schema where the focus was on the entity type Refund. The schema presents a Refund as a descendant of the abstract entity type Commentable, associated to instances of RefundLine, and that a Refund belongs to the Order to which it is related, which is also a descendant of Commentable. In addition to it, the filtered schema shows three invariants.

On the other hand, Fig. 8.24 shows the filtered conceptual schema that results when the user includes the four entity types from the filtered schema of Fig. 8.23 into the focus set of a new filtering request. That schema includes the same elements as in Fig. 8.23 (marked with a darker color) and two additional entity types of high relevance to them: Website and StoreView. Also, that schema contains two additional referentially-complete invariants.

Therefore, the interaction between the user and our proposed prototype results in an iterative process that must be applied as many times as required. The process ends when the user believes that she has obtained enough knowledge from the large schema to cover the specific information need, or wants to apply a different filtering request.

Figure 8.25. Screenshot of $\mathcal{F}_5$ (request) in the filtering engine prototype tool

8.5.5 Filtering Request $\mathcal{F}_5$ – Prototype

The fifth filtering request focuses on a set of (one or more) entity types from a large schema and returns the corresponding filtered schema with the event types of interest to the user.

In our implementation we provide users with a web-based interface to ease the construction of the focus set. To this end we follow the interaction pattern described in Sect. 8.4.5 where the user is able to include entity types into the focus set through three different components. Figure 8.25 shows the web-based interface of our prototype tool for $\mathcal{F}_5$.

First of all, we construct a word cloud with the top-25 most relevant entity types of Magento where more relevant ones are depicted in a larger font size. This format is useful for quickly perceiving the most prominent entity types by inexperienced users. Selecting a single entity type name within the word cloud, includes such entity type in the focus set.

Alternatively, the interface contains a search bar, which provides an auto-completing functionality that helps discovering the names of the existing entity types of Magento. The names of the selected entity types are shown in the search bar.

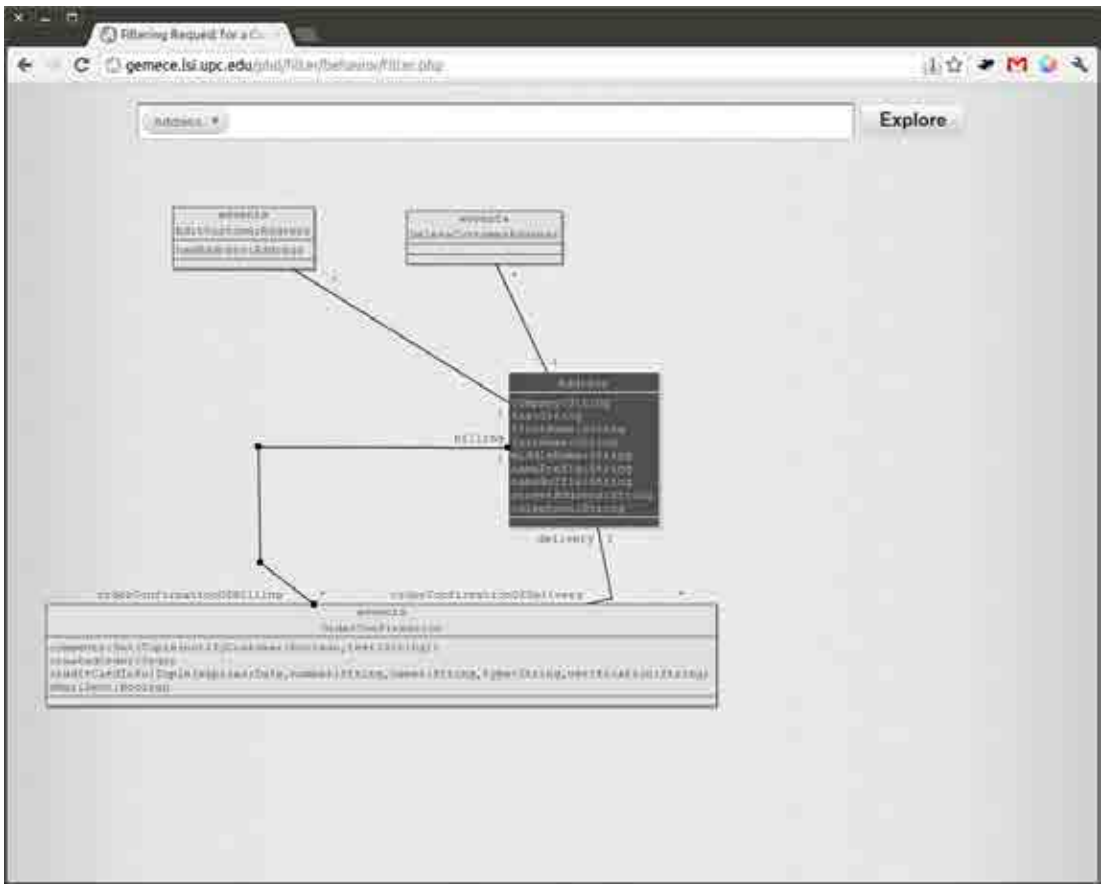


Figure 8.26. Screenshot of $\mathcal{F}_5$ (response) in the filtering engine prototype tool

In addition to it, we include an alphabetical list to explore all the entity types of Magento. The user selects a single letter of the English alphabet from the list and then obtains an enumeration of those entity types whose name starts with that letter. The user may select the entity types of interest, which are then included in the search bar.

The user may also select the final size of the filtered schema in order to indicate the amount of knowledge she wants to obtain as a result. As an example, Fig. 8.25 shows the filtering prototype when the user selects the entity type Address as the input focus set. By using our prototype (see Fig. 8.26), the user discovers the set of event types that are related to such entity type.

Therefore, we observe in the resulting filtered schema that the event types DeleteCustomer-Address and EditCustomerAddress make use of the entity type Address for their specification. Also, the event type OrderConfirmation is related to two instances of Address for the roles of billing and delivery, which indicates that in Magento orders of products require a billing address and a delivery address.

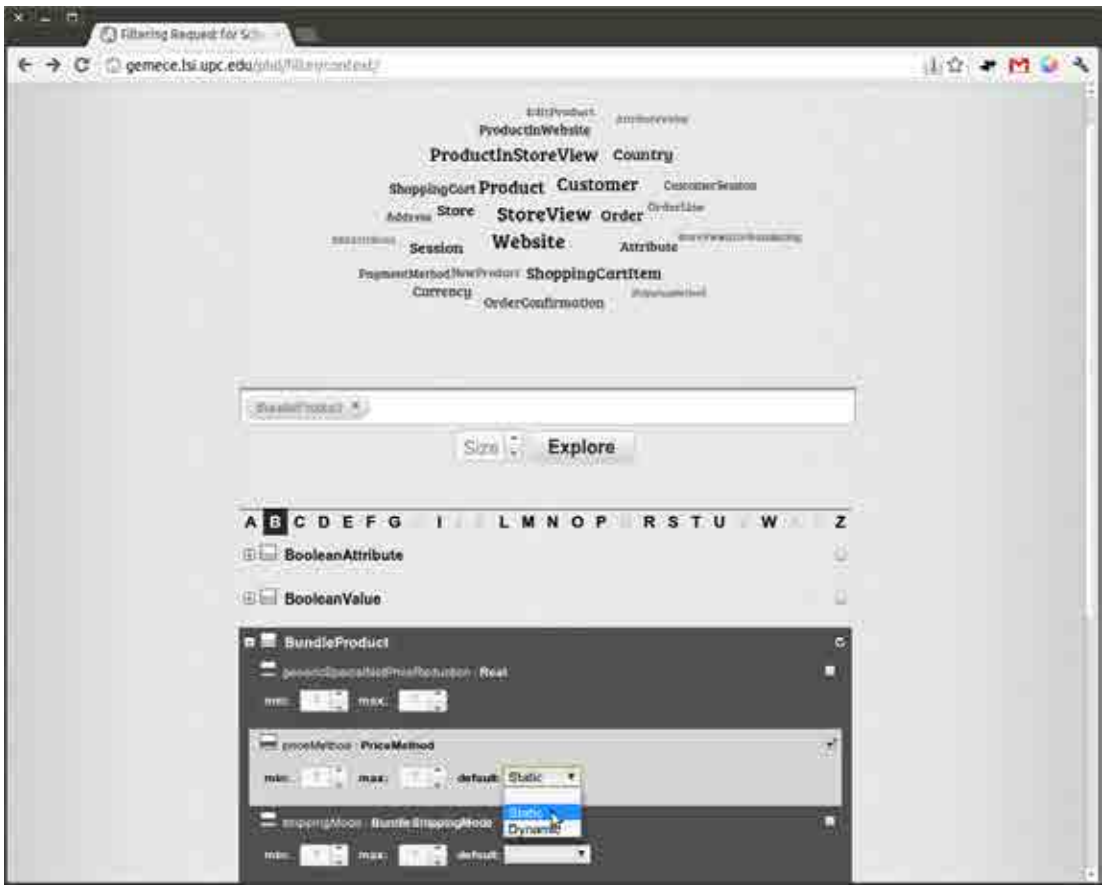


Figure 8.27. Screenshot of $\mathcal{F}_6$ (request) in the filtering engine prototype tool

8.5.6 Filtering Request $\mathcal{F}_6$ – Prototype

The sixth filtering request focuses on a set of (one or more) entity and event types from a large conceptual schema and returns the corresponding filtered schema with the knowledge of interest to the user when applying a contextualization function.

In our implementation we provide users with a web-based interface to ease the construction of the focus set. To this end we follow the interaction pattern described in Sect. 8.4.6 where the user is able to include entity and relationship types into the focus set through three different components. Figure 8.27 shows the web-based interface of our prototype tool for $\mathcal{F}_6$.

First of all, we construct a word cloud with the top-25 most relevant entity and event types of Magento where more relevant ones are depicted in a larger font size. To compute the word cloud we make use of the importance-computing algorithms presented in Sect. 4.3 of Ch. 4. This format is useful for quickly perceiving the most prominent entity and event types by inexperienced users. Selecting a single entity or event type name within the word cloud, includes such entity or event type in the focus set.

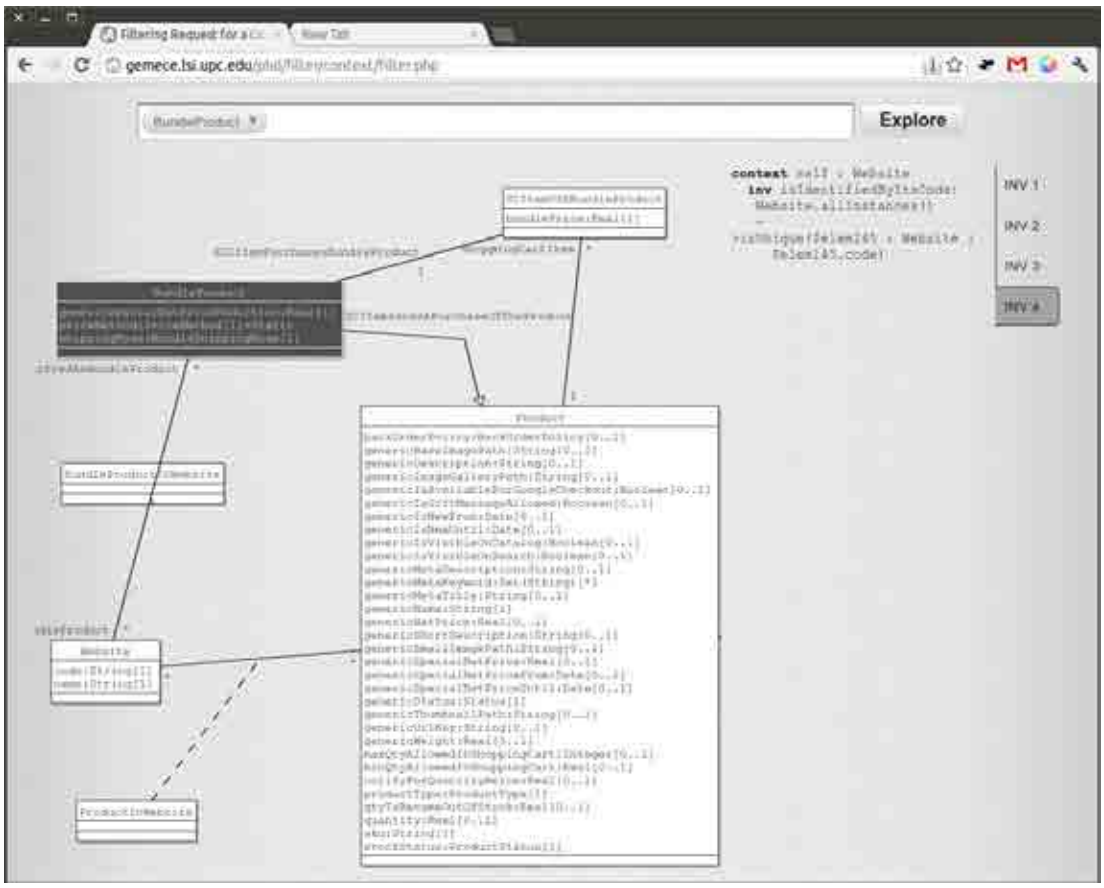


Figure 8.28. Screenshot of $\mathcal{F}_6$ (response) in the filtering engine prototype tool

Alternatively, the interface contains a search bar, which provides an auto-completing functionality that helps discovering the names of the existing entity and event types of Magento. The names of the selected entity and event types are shown in the search bar.

In addition to it, we include an alphabetical list to explore all the entity and event types of Magento. The user selects a single letter of the English alphabet from the list and then obtains an enumeration of those entity and event types whose name starts with that letter. Each item in the list shows also its attributes and participations in relationships. The user may select the entity or event types of interest, which are then included in the search bar, and additionally change the multiplicities of (some of) their attributes or participations in relationships, or even select default values for those attributes whose type is an enumeration. Note that we preceded each item in the alphabetical list with an icon showing the type of the schema element (entity type, event type, participation in relationship type, or owned attribute). By changing multiplicities of selecting default values the user applies a contextualization function, as indicated in Sect. 6.3.6 of Ch. 6 where we introduced the filtering request for contextualized types ($\mathcal{F}_6$).

The user may also select the final size of the filtered schema in order to indicate the amount of knowledge she wants to obtain as a result. As an example, Fig. 8.27 shows the filtering prototype

when the user selects the entity type BundleProduct as the input focus set and selects a default value for its owned attribute priceMethod. By using our prototype (see Fig. 8.28), the user discovers that such entity type descends from the entity type Product, and that is related to the entity types Website and SCItemOfBundleProduct. Note that the attribute priceMethod of BundleProduct shows its default value as *Static*.

8.6 Summary

In this chapter we have presented an implementation of the filtering methodology introduced in Ch. 5, including the development of a filtering web client and a set of core web services that deal with the management of the user interaction and the execution of the iterative process to filter a large conceptual schema by using the filtering requests of Ch. 6.

The implementation of our filtering methodology is not unique. There are several ways of designing and coding a filtering system following our catalog of filtering requests. However, our proposal provides a minimum working application that helps the user on her task of extracting fragments of knowledge from a large schema taking into account the specific point of view and interest of the user.

We based the development of our filtering system on a web-service approach due to the benefits of interoperability it provides. Any user interested in making use of our tool only requires a web browser to access it. Therefore, our web-based filtering engine allows us to reach a broader audience of users that can use the tool from any location and device. Furthermore, the component-based development we propose simplifies the maintenability and reusability of internal and external services that are part of our web architecture.

Finally, our service-oriented approach provides ways to easily extend the functionalities of the filtering engine (e.g., implement a new filtering request to fulfill specific needs of a particular domain, design a new client view that allows the user to explore additional knowledge of the schema, include a new interaction point to extend the user interaction with a particular view) without changing the existing implementation.

Computers are good at following instructions,
but not at reading your mind.

Donald Knuth

9

Adaptation of the Filtering Methodology to HL7 V3 Schemas

The ability to share information across systems and between care organizations is one of the major challenges in the healthcare business progress towards efficiency and cost-effectiveness. Healthcare standards play a key role in this area. We focus on the broadly used Health Level 7 Version 3 standard. It provides information models that define the structure and contents of clinical documents and messages for the exchange, integration, and retrieval of electronic health information on clinical information systems. These models, which are subsets of a large conceptual schema, contain huge amounts of knowledge, and are specified in a non UML-compliant modeling language, which difficults the understanding of the standard and its practical application by software engineering experts.

The chapter starts with an introduction to healthcare interoperability standards in Sect. 9.1. Section 9.2 presents the particularities of the HL7 V3 model-based healthcare standard, including its structure and development framework. Section 9.3 enumerates several approaches to improve the usability of the standard. In Sect. 9.4 we propose a transformation process to automatically translate the information models within HL7 V3 to the UML in order to benefit from collaboration, contributions, and existing tool-support in the software engineering area. We adapt our filtering method to be able to process the resulting set of UML schemas from the transformation as a large conceptual schema and, therefore, provide ways to explore the knowledge within them. Section 9.5 describes the details about this approach. Finally, Sect. 9.6 presents the results of the application of the filtering methodology to HL7 V3, and Sect. 9.7 summarizes the chapter.

9.1 Healthcare Interoperability Standards

Information systems play a key role in managing and exchanging clinical information. There is a growing need to connect complex applications from different vendors that handle and operate with large amounts of data from a wide range of healthcare domains. These applications aim to reduce the cost, save time, and eliminate redundancies and errors on transferring healthcare data—such as patient records, pharmacy orders, laboratory requests, or clinical reports. As a consequence, the design, implementation, and interoperability of clinical information systems becomes a challenge that has to be addressed [51].

Healthcare semantic interoperability is defined as the ability to exchange, understand, and act on clinical information among linguistically and culturally disparate health professionals, patients, and other actors and organizations —within and across healthcare jurisdictions— in a collaborative manner [28, 19]. The information transferred may be at the level of individual patients, but also aggregated information for quality assurance, policy, remuneration, or research [63]. Understandability of such information is one of the key aspects in semantic interoperability.

The inability to share information across systems and between care organizations is one of the major impediments in the health care business progress toward efficiency and cost-effectiveness. Using healthcare standards is a precondition for achieving semantic interoperability [86]. Healthcare standards play a key role in the battle against paper records, independent and autonomous systems, and manual interoperability. Also, there is a wide range of standards available for the integration and interoperability of applications and information systems, both on domain-specific and domain-neutral levels [82]. However, there exists a slow adoption of those standards in real industrial contexts.

The catalog of healthcare interoperability standards includes standards for the interchange and visualization in medical imaging —DICOM standard [16]; definition of healthcare codes and medical terminology in the electronic exchange and gathering of clinical results (such as laboratory tests, clinical observations, outcomes management and research) —LOINC standard [45]; sharing of comprehensive computerized clinical terminology covering clinical data for diseases, clinical findings, and procedures —SNOMED standard [110]; the representation and transmission of almost all medical data in form of clinical artifacts such as messages and documents between healthcare providers —HL7 V3 standard [9]; and additional areas of concern or interest of medical information [53].

All these healthcare interoperability standards are essential to progress toward shared healthcare knowledge, balancing quality of care with cost containment, improved care delivered to patients, and a more business process view of healthcare delivery. Their potential usefulness lies in the fact that they contain a huge amount of information of a wide range of clinical domains. However, this complexity makes difficult to integrate them and to train software engineering professionals to be able to master them. Doing so requires a thorough analysis and understanding of existing healthcare standards by software experts.

We focus on the HL7 V3 standard for exchanging messages and documents among information systems that implement healthcare applications. Information models are the backbone

of HL7 V3-based information systems. These models, which are considered in this thesis as subsets of a large conceptual schema, are artifacts provided by the standard that play a key role in the integration of heterogeneous databases and medical information systems in the healthcare domain [113]. We propose an adaptation of our filtering methodology to deal with the particular characteristics of HL7 V3 information models.

9.2 Health Level 7 Version 3

The Health Level Seven International (HL7)[1] is a not-for-profit, ANSI-accredited standards developing organization dedicated to providing standards for the exchange, integration, sharing, and retrieval of electronic health information that supports clinical practice and the management, delivery, and evaluation of health services. HL7 standards enable semantic interoperability between almost all institutions, domains, and fields of healthcare. With HL7, all important communication tasks of a healthcare provider (hospitals, clinics, primary care centers, and other service delivery points) can be handled and the efficiency of communications can be improved.

HL7 Version 3 is an specific healthcare interoperability standard within HL7 designed to handle most if not all healthcare communications, using a relatively small set of constructs [12, 108, 17]. The standard specifies storyboards, trigger events, and interactions between clinical applications to maintain a common representation of healthcare messaging tasks such as the update of a patient billing account, the registration of a new observation in a diagnosis, or the submission of a blood test request to a laboratory.

The V3 standard presents a model-based specification and a development framework covering the whole life cycle from design through adaptation and maintenance up to implementation, use, and testing of clinical messages and documents on the basis of a Reference Information Model (RIM) [100]. The standard indicates the information each kind of clinical message or document must contain and how it is structured through the definition of information models based on the RIM. An information model is a structured specification of the information within a specific domain of interest. It expresses the concepts of information required and the properties of these concepts, including attributes, relationships, constraints, and states. The standard defines different types of information models to represent documents and messages to exchange in several healthcare domains.

HL7 V3 and its model-based development framework enable semantic interoperability between different applications that adopt the same healthcare information models. Despite that, a successful integration of the HL7 V3 standard in an interoperability project to semantically interconnect clinical information systems truly depends on the expertise and knowledge of all the participants, including analysts, designers and developers, when aligning the specific project requirements with the information models defined in the standard. This problem has been widely explored and studied by many researchers in the field of software engineering and, specially, the area of conceptual modeling [141, 26]. The adoption of contributions from this field may increase the quality of the standard and its development framework.

[1] Health Level 7 International `http://www.hl7.org`

9.2.1 The HL7 V3 Development Framework

The HL7 V3 standard is based on a comprehensive development methodology where first techniques of modern software engineering have been deployed within a standard development process such as object-oriented analysis and object-oriented design as well as formal conceptual modeling. It presents a model-based specification and a development framework covering the whole life cycle from design through adaptation and maintenance up to implementation, use, and testing of clinical messages and documents on the basis of a top-level reference information model.

The HL7 Reference Information Model (RIM) [100] describes the core classes for the healthcare subject areas, the attributes for each of these classes as well as their associations. Figure 9.1 presents a summary with the six most important classes of the RIM. According to it, healthcare consists of a series of clinical *Acts* performed to, by, on behalf of, utilizing, or in some way involving, one or more instances of a Participating Entity-in-a-Role. For instance, the entity *John Smith* in the role of *Patient* participates as the *Subject* of an act of *ClinicalObservation*.

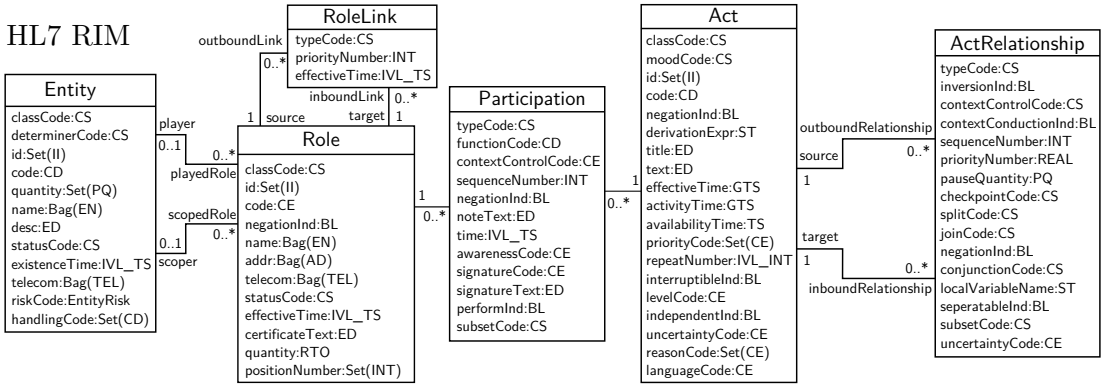


Figure 9.1. HL7 Reference Information Model (RIM).

Figure 9.2 shows an overview of the HL7 V3 development framework. Each V3 message or document specification is a view derived from the RIM (condensed in Fig. 9.2 A). Every particular healthcare domain defines their own Domain Message Information Model (D-MIM), which is a derived specialization of the RIM that includes a set of concepts, attributes, and relationships that can be used to create messages and structured clinical documents for a particular topic of interest in the domain (Fig. 9.2 B). For instance, the set of concepts that are used by the laboratory domain –observation, specimen, device– is quite different from that used by the accounting and billing domain –account, insurance policy. As a consequence, the D-MIMs for these two domains are quite different, although both contain classes that are specializations from the classes of the RIM, and follow the Act-Participation-Role-Entity structure.

Then, every single interaction –exchange of message or document– between healthcare applications implementing HL7 V3 (Fig. 9.2 D) is defined by a Refined Message Information Model (R-MIM). An R-MIM is a subset of the specific D-MIM for the domain in which the R-MIM is used (Fig. 9.2 C). The R-MIM contains only those classes, attributes, and associations of a D-MIM required to model specific case scenarios for a set of messages or documents.

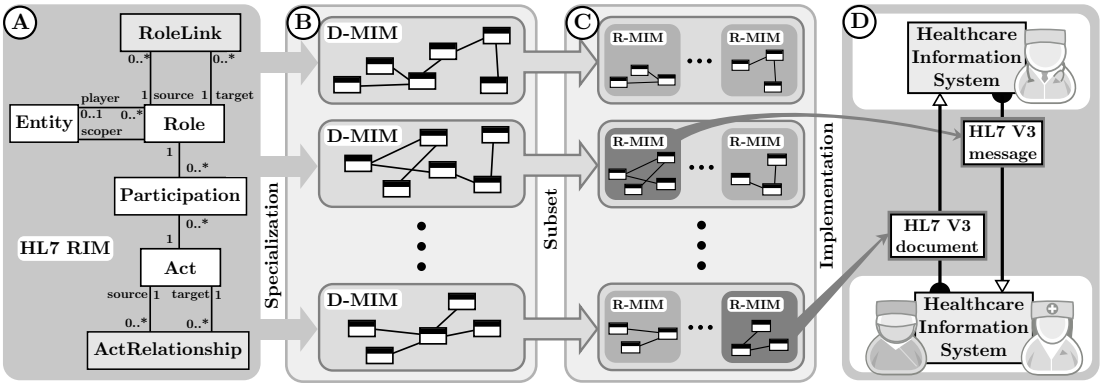


Figure 9.2. HL7 V3 Development Framework.

The HL7 V3 development framework requires that all information structures in derived models be traceable back to the RIM. Their semantic and related business rules should not conflict with those specified in the RIM. The RIM therefore is the ultimate source for all information content in HL7 V3. Each R-MIM is a subset of a D-MIM, and a D-MIM is a specialization of the RIM.

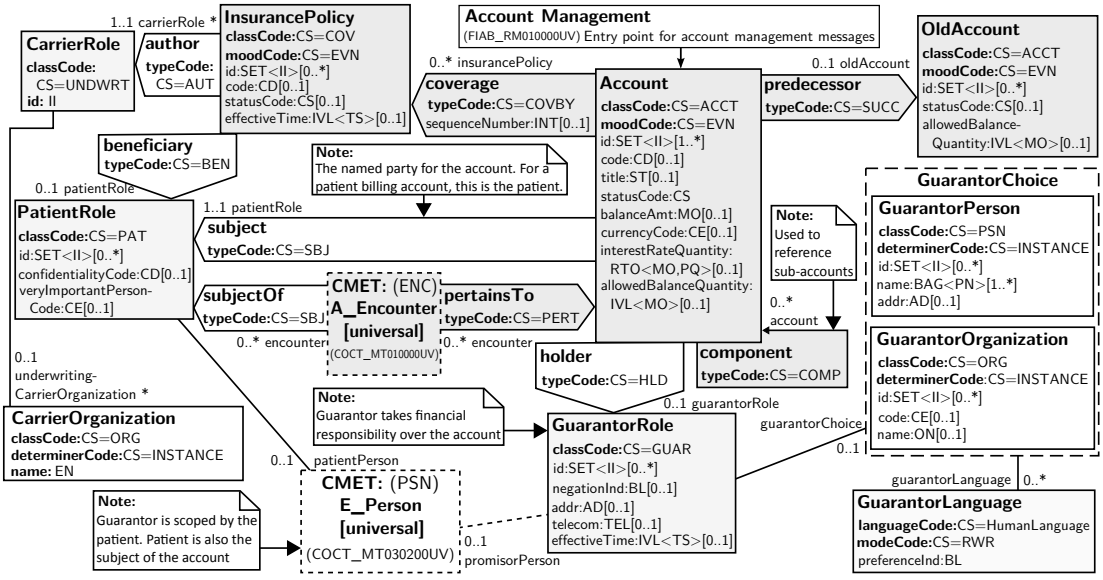


Figure 9.3. Patient Billing Account Event R-MIM (FIAB_RM010000UV02).

The HL7 V3 2010 normative edition of the standard contains 22 D-MIMs and 870 R-MIMs. Figure 9.3 presents the Patient Billing Account Event R-MIM, which serves as our running example. According to the standard, this message model includes the required concepts for the creation and management of patient billing accounts specified in a non UML-compliant modeling language. Central to the model of Fig. 9.3 is the concept of *Account*, with attributes such as identifier, code, balance, and the currency of the account. An account is an accumulator of financial and administrative information for the main purpose of supporting claims and reim-

bursement. Associations to the account include patient identification (subject of the account), insurance policy, encounter, and the optional specification of a guarantor for any outstanding balance in the account.

An HL7 V3 expert that wants to implement the message to create an account in a particular healthcare system starts exploring the RIM. Then, she selects the specific healthcare domain within the standard that describes the knowledge of interest for the message, and explores its D-MIM model. The last step consists in selecting the appropriate subset of the D-MIM content in order to obtain an R-MIM with the structure of the message for the purpose at hand.

The structure and contents of the specific message to create an account is defined by the Patient Billing Account Event R-MIM in Fig. 9.3, which is a subset of the Patient Billing Account D-MIM of the accounting and billing healthcare domain, which is a derived model from the HL7 RIM. This sequence of model derivation and model subsetting operations increases the complexity of the overall process to obtain and understand a particular information model from the HL7 V3 standard.

9.3 Improving the HL7 V3 Standard

Based on our previous contributions [132], our collaboration with HL7 members in projects related to the usability and learnability of the standard, and our experience with the information models and documentation from the HL7 V3 standard, we believe that even though the potential of HL7 V3 to enable semantic interoperability in the healthcare domain has been clearly stated, several difficulties emerge in its practice and functionality when used by software engineers and information systems developers.

In our opinion, the HL7 V3 standard is ideally suited to represent all the knowledge of interest for healthcare professionals in a wide range of domains such as care provision, pharmacy, medical records, laboratory, scheduling, account and billing, or public health reporting. However, software professionals that have to deal with the standard in real developments find it difficult to explore its documentation and to effectively construct and maintain HL7 V3-based applications.

The adequate management of the healthcare information models of the standard and the complexity to implement them in an interoperability scenario are central aspects for the success of HL7 V3. In the software engineering literature, most of the aspects related to information models have been extensively studied over the years [117]. Consequently, several existing tools and methodologies from this field may be applied and adapted to obtain, refine, understand, and use information models from the healthcare domain.

In real interoperability scenarios, software professionals are in charge of connect several clinical information systems in order to exchange, understand, and act on healthcare data. To do this, software analysts, designers and developers must know the structure of the HL7 V3 documentation, select from all the domains in the standard those that are closer to the requirements of the scenario, explore their storyboards and specifications to select the specific

messages or documents, understand their information models and the elements within them, including descriptions of classes, attributes, and relationships, and adapt them to a particular context. Moreover, professionals must complete this task manually — with no additional assistance apart from their own experience.

An information model is useless unless its purpose is properly understood by the specialists that have to work with it. We have identified a need to broaden the audience of the HL7 V3 models by providing a UML version to experts on model-driven development. The main benefits of providing a standard UML version of the healthcare information models within HL7 V3 are related to understandability, tool-support, and analyzability of the standard. The usage of UML as the modeling language of the HL7 V3 models allows all members of the software engineering community to understand the standard without requiring further training. Furthermore, the healthcare community can benefit from existing modeling tools and methodologies.

Nowadays, there is a wide range of methods and applications that support UML. There are assistants to generate most of the final code of a software system from its UML information model, or even interpret this model and make it executable [76]. It is possible to check the consistency of UML models in order to solve many issues in the software development process [13]. We can translate a UML model into a narrative description that must enable non-experts or business people to understand its semantics [23]. There is a large amount of tools to visualize and filter UML models that integrate other functions and help users to deal with them [132]. Also, there is a wide community of UML experts that may collaborate and contribute to the improvement of HL7 V3 models.

There exist several attempts to introduce UML for the specification of HL7 V3 models [90, 58] but up to now only the RIM (see Fig. 9.1) appears to be a consolidated UML-compliant model due to its simplicity and its reduced size. In the following section we propose a method to automatically obtain a UML version of the healthcare information models provided by HL7 V3 without requiring to change the current specification and modeling languages of these models. In Sect. 9.5 we use the resulting UML schemas from the transformation as the input of an adapted version of the filtering methodology described in Ch. 5 to increase the usability and understandability of the knowledge within the HL7 V3 standard.

9.4 Transformation from HL7 V3 to UML

As aforementioned, a present challenge of HL7 V3 is to provide a more abstract, formal, and consistent specification of its information models. These models contain special constructions specified in a non UML-compliant modeling language, which makes it difficult to learn the particularities of the standard by software engineering experts. In addition, there are few tools that support V3 models, and their maturity level is low.

In the context of Model-Driven Engineering [65, 37], information models (conceptual schemas) are the main development artifacts, and model transformations are among the most important operations applied to models. Model-to-model (M2M) transformations transform source models into target models. There exist different model transformation approaches [35, 138].

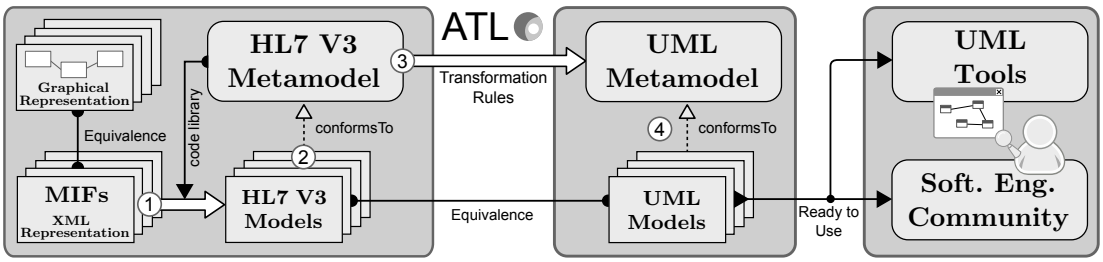


Figure 9.4. Overview of the automatic transformation process.

Figure 9.4 presents an overview of our transformation, which is based on the ATL framework [62], a model transformation language and toolkit developed on top of the Eclipse [38] platform that provides ways to produce a set of target models from a set of source models. ATL transformations are unidirectional, operating on read-only source models and producing write-only target models. In other words, during the execution of a transformation the source model may be navigated but changes are not allowed to it.

The transformation starts with the Model Interchange Format (MIF) [109] files of the HL7 V3 that contain the most detailed XML representation of the graphical information models within the HL7 V3 standard. The MIF files are a set of XML files used to support the storage and exchange of models as part of the the HL7 version 3 development process, as well as to support tooling and to provide documentation of HL7 methodology. The HL7 V3 editions of the standard are generated from the MIF files.

Those MIF files are processed and converted into instances of an HL7 V3 metamodel, i.e., into formal HL7 V3 models. For instance, an HL7 V3 model may describe the roles of *Guarantor* or *Patient* while the HL7 V3 metamodel describes the concept of *Role* and that it must be played by an *Entity* that has a *Participation* in an *Act*. Section 9.4.1 presents our HL7 V3 metamodel.

Next, the ATL engine executes a set of transformation rules to convert a HL7 V3 model to its UML equivalent. An ATL rule takes an element from the source model and transforms it into one or more elements of the target model. These transformation rules are defined at the metamodel level, i.e., they are specified in the context of the HL7 V3 metamodel and describe the translation to the UML metamodel. Therefore, whenever a HL7 V3 model changes, it can be translated to its UML equivalent as long as it conforms to the HL7 V3 metamodel, which means that there is no need to change the transformation rules unless we have to change the HL7 V3 metamodel. Section 9.4.2 presents the details of the transformation rules.

As a result, the transformation automatically produces a UML model, which is a valid instance of the UML metamodel, for each HL7 V3 model. The resulting UML models are ready to be used in existing UML tools by experts of the software engineering community.

Consequently, the overall M2M transformation process requires the specification of two metamodels —HL7 V3 and UML— and a set of transformation rules to translate the elements of the first metamodel into elements of the second one. We have developed a new HL7 V3 metamodel and the ATL transformation rules to translate between HL7 V3 to UML. The UML metamodel is already implemented in the Eclipse platform.

9.4.1 A Metamodel of HL7 V3

The HL7 V3 standard specification does not contain a formal metamodel, i.e., an information model that describes the characteristics and semantics of all the elements that are present in V3 models. We need a HL7 V3 metamodel because it is required for the ATL infrastructure to be the source of the transformation rules that translate to the target UML metamodel. We have constructed a HL7 V3 metamodel that defines the elements present in the HL7 V3 models.

We have studied the HL7 V3 documentation and performed a reverse-engineering study through all the V3 models in the standard in order to extract the general characteristics those models share. Since we want to translate V3 models into the standard UML our primary goal was to develop a HL7 V3 metamodel as conceptually close as possible to the UML metamodel. This will simplify the specification of the transformation rules of the overall process.

A simplified version of our HL7 V3 metamodel is presented in Fig. 9.5 (see [91] for the complete version). Those elements marked in gray are directly extracted from the UML metamodel, with minor changes. For instance, the metamodel shows that only two properties participate in any association since the HL7 V3 standard allows only binary associations. The hierarchy of *Class* represents the core element types of the RIM, including *Entity*, *Role*, *Participation*, and *Act*. Also, there are elements like *Choice*, *CMET* or *EntryPoint* that are specific constructs which only appear in HL7 V3 models and, therefore, they need to be transformed into standard constructs of the UML. As a consequence, our HL7 V3 metamodel indicates that in the HL7 V3 information models, a *Choice* must be associated with at least two *ChoosableElement*, and that a *ChoosableElement* must be a *Class*, a *CMET*, or another *Choice*. These are common rules in the standard.

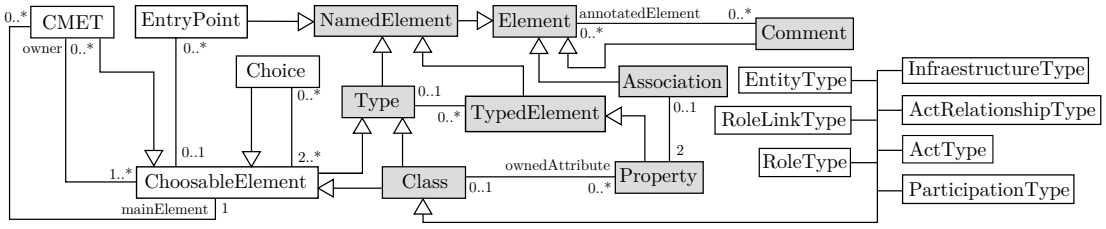


Figure 9.5. Simplified version of the HL7 V3 metamodel.

The effort to develop a HL7 V3 metamodel allows to easily process the MIF files from the standard and transform them into HL7 V3 models that satisfy our HL7 V3 metamodel. We have implemented the HL7 V3 metamodel as an Ecore model in the Eclipse Modeling Framework [111] . It allows to easily define the HL7 V3 meta-elements shown in Fig. 9.5 and automatically generates a complete Java code library to create instances of these meta-elements matching the characteristics defined in the HL7 V3 metamodel. We have processed the MIF files and created the corresponding HL7 V3 models through the generated code library. As a result, we can assure that the HL7 V3 models we obtain are a valid instance of the HL7 V3 metamodel and that each one contains the same knowledge as its source MIF file from the standard. The HL7 V3 models from the MIF files are the required input of the ATL transformation depicted in Fig. 9.4.

9.4.2 Transformation Rules

In the scope of the ATL framework, the generation of target model elements is achieved through the execution of transformation rules. Each rule is defined in the context of the source metamodel, deals with an element from it, and describes its equivalence in the target metamodel. In the following, we describe the effect of the transformation rules for the main elements of our HL7 V3 metamodel that are specific from the HL7 V3 standard and require a translation to common UML constructs.

Entry Points

An entry point indicates the central element of a particular HL7 V3 information model. Each D-MIM model will have at least one entry point. Since the D-MIM encompasses an entire domain, there may be several entry points, one for each R-MIM contained in the D-MIM of the domain. Entry points are represented on the D-MIM model as clear boxes with black borders (see Fig. 9.6 left). A black arrow originates from the entry point box and points to a class which is considered the root, or focal class in the model. Each entry point has a name, carries the identifier of the R-MIM which originates from it, and contains a brief description.

Algorithm 9.1. ATL transformation rule for Entry Points.

```
1  module HL7ToUML;
2  create OUT : UML2 from IN : HL7;
3
4  rule EntryPoint {
5   from ep: HL7!EntryPoint
6   to m: UML2!Model( name<-ep.identifier ),
7      c: UML2!Class( name<-ep.name, ownedAttribute<-desc, isAbstract<-true ),
8      desc: UML2!Property( name<-'description', default<-ep.description,
9                           type<-String )
10     e1: UML2!Property( lower<-0, upper<-1, type<-c )
11     e2: UML2!Property( lower<-0, upper<-* )
12     a: UML2!Association( ownedEnd<-Set{e1,e2} )
13  do {
14     m.packagedElement.add(c);
15     c.applyStereotype(c.getApplicableStereotype('EntryPoint'));
16     for (cl in HL7!Class.allInstancesFrom('IN'))
17         { thisModule.Class(m, cl); }
18     for (cm in HL7!CMET.allInstancesFrom('IN'))
19         { thisModule.CMET(m, cm);  }
20     for (ch in HL7!Choice.allInstancesFrom('IN'))
21         { thisModule.Choice(m, ch); }
22     for (a  in HL7!Association.allInstancesFrom('IN'))
23         { thisModule.Association(m, a); }
24     e2.type<-UML2!Class.allInstancesFrom('OUT')
25         ->select(c|c.name = ep.choosableElement.name).first();
26 }}
```

We transform each entry point element into an abstract UML class with the stereotype ≪`EntryPoint`≫ and an attribute named *description* with the original information of the entry point. The ATL rule 9.1 shows a simplified declaration of this transformation that starts the process. The header of the transformation (line 2) defines the source and target metamodels to be used in order to reference schema element types. The EntryPoint rule matches the entry point

of the source model (line 5) and creates a UML model whose name is the identifier of the entry point (line 6). It also creates the representation of the entry point as an abstract UML class with the same name, and a description attribute with the description of the entry point (lines 7-9). This class is included into the new model (line 14) with the «`EntryPoint`» stereotype (lines 15). The last part of the rule (lines 15-23) calls the appropriate rules for the transformation of the other elements within the source model. Note that the arrow that connects the entry point with the element of focus is transformed into an association (lines 10-12 and 24-25). Figure 9.6 graphically presents the transformation of the entry point *Account Management* of Fig. 9.3.

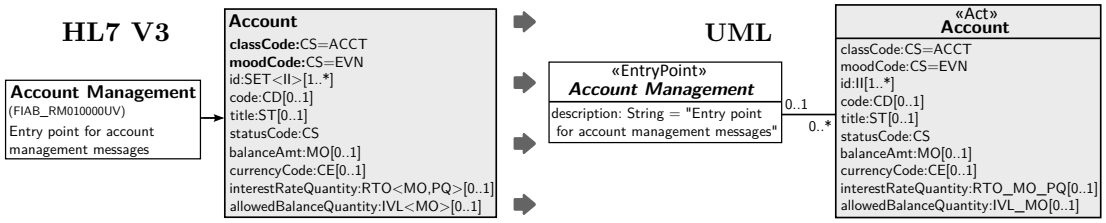


Figure 9.6. UML translation for the HL7 V3 entry point *Account Management*.

Classes, Associations, and Attributes

A class represents a concept of a particular domain and an association specifies a semantic relationship between classes. A HL7 V3 class can be classified as an *Act*, *ActRelationship*, *Participation*, *Role*, *RoleLink*, or *Entity* class —as defined in the RIM and in our HL7 V3 metamodel. We transform each HL7 V3 class into a UML class with the same name and attributes as the original but with an stereotype with the name of its kind of class derived from the RIM, as specified in the ATL rule 9.2. As aforementioned, this rule is invoked by the ATL rule 9.1. The first part of the rule creates a new UML class from the HL7 V3 one with the same name (line 2). Then, the rule applies the required stereotype to the UML class according to the type of the source HL7 V3 class (lines 5-11). Finally, we invoke a new rule to transform the attributes of the HL7 V3 class into UML attributes of the new class (line 12). The details of this rule can be found in [91].

Algorithm 9.2. ATL transformation rule for HL7 V3 classes.

```
1  rule Class(m: UML2!Model, hc: HL7!Class) {
2   to uc: UML2!Class( name<-hc.name )
3   do {
4      m.packagedElement.add(uc);
5      if (hc.oclIsTypeOf(HL7!ActType)) {
6          uc.applyStereotype( uc.getApplicableStereotype('Act') );
7      }
8      else if (hc.oclIsTypeOf(HL7!RoleType)) {
9          uc.applyStereotype(  uc.getApplicableStereotype('Role'));
10     }
11     ... -- same behavior for other class types
12     for (a in hc.ownedAttribute) { thisModule.Attribute(uc, a); }
13 }}
```

Similarly, a HL7 V3 association is transformed into a UML association with the same participants, role names, and multiplicities as the original one. Figure 9.7 graphically presents the transformation of the associations between three classes from the information model of Fig. 9.3.

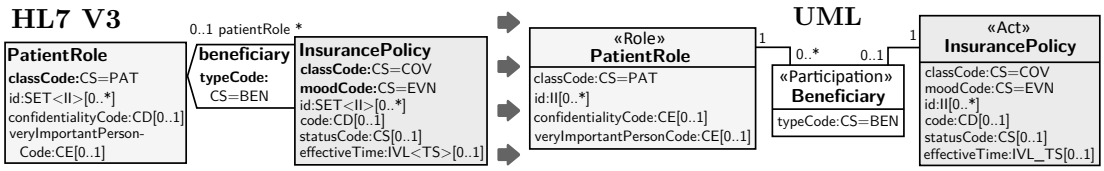


Figure 9.7. UML translation for HL7 V3 classes and associations.

The stereotype that represents the kind of class according to the RIM can be viewed as an implicit generalization relationship that connects a UML class with its parent class in the RIM. For instance, the *InsurancePolicy* class in left side of Fig. 9.7 is a descendant of the *Act* class in the RIM. In our transformation, we used stereotypes instead of generalization relationships as a way to reducing clutter in the resulting information models.

Choices

HL7 V3 information models may also contain *choice* boxes. These boxes are bordered by a dashed line and enclose two or more classes that are part of an inheritance hierarchy (e.g. two or more Roles, two or more Entities, etc.). It is important to note that any association connected to the choice box apply to all classes within it. Associations connected to a specific class within the choice box apply only to that class.

We transform each choice into an abstract class with the same name as the choice and the stereotype ≪`Choice`≫. The classes within the choice are transformed into regular UML classes sharing a generalization relationship with the choice class. Any association connected to the choice is now connected to the abstract class representing the choice in UML.

Algorithm 9.3. ATL transformation rule for HL7 V3 choices.

```
1  rule Choice(m: UML2!Model, ch: HL7!Choice) {
2   to c1: UML2!Class ( name <- ch.name, isAbstract <- true )
3   do {
4      m.packagedElement.add(c1);
5      c1.applyStereotype( c1.getApplicableStereotype('Choice'));
6      for (oe in ch.ownedElement) {
7           thisModule.createChoiceHierarchy(c1,
8              UML2!Class.allInstancesFrom('OUT')
9                ->select(c | c.name = oe.name).first());
10 }}  }
```

The first part of the the ATL rule 9.3 creates the UML class that represents the choice with the name of the choice and makes it abstract (line 2). Then the rule applies the stereotype ≪`Choice`≫ (line 5) and creates the hierarchy of the choice with generalization relationships to connect it with its owned elements, which are instances of *ChoosableElement* according to the HL7 V3 metamodel (see Fig. 9.5).

Figure 9.8 depicts the transformation of the choice of Fig. 9.3, which consists in the abstract class *GuarantorChoice* representing the source choice, and two generalization relationships connecting it with its descendants *GuarantorPerson* and *GuarantorOrganization*.

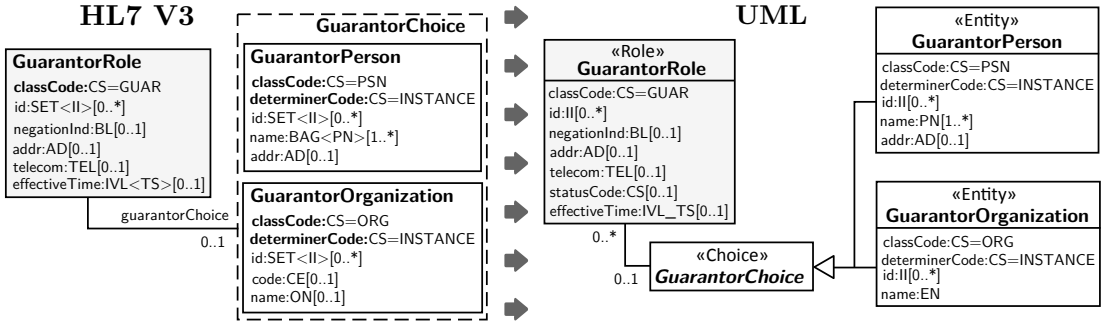


Figure 9.8. UML translation for the HL7 V3 choice *GuarantorChoice*.

CMETs

A Common Message Element Type (CMET) is a predefined container of common HL7 V3 elements that is re-used for several information models in order to avoid repetitions and simplify the model. CMETs can be seen as a hyperlinks to other models and are graphically denoted as boxes with a dashed border. The CMET box contains a name, the base class type, and the identifier of its own corresponding R-MIM where the contents wrapped by the CMET are defined. We transform each CMET into a UML class with the name of the CMET and a read-only attribute named *identifier* whose value is the original identifier of the R-MIM of the CMET. Also, this class is marked with the stereotype ≪`CMET`≫, and the stereotype of the base class type of the CMET. The details of the ATL rule for CMETs are similar to ATL rule 9.2 and can be found in [91].

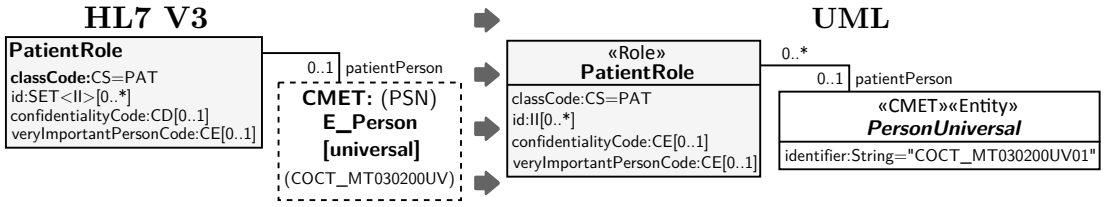


Figure 9.9. UML translation for the HL7 V3 CMET *PersonUniversal*.

Figure 9.9 depicts the HL7 V3 CMET of Fig. 9.3 named *E_Person[universal]* (left) and its representation as an abstract class in UML (right). It contains the identifier *COCT_MT030200UV01* of the model were the contents referenced by the CMET are defined, as an attribute named *identifier*. The stereotypes ≪`CMET`≫ and ≪`Entity`≫ represent that the base class type of the information model referenced by the CMET is an Entity. In this case, the focal class of the R-MIM referenced by this CMET is the entity *Person*.

9.4.3 Transformation Results

We have tested our automatic transformation process with the HL7 V3 2010 normative edition, which contains 870 MIF files. The overall process took around 3 minutes in a computer with a 2.8GHz Intel Core i7 processor with 8GB of DDR3 RAM. The full details can be found in [91]. Since new HL7 V3 normative editions appear once a year, and preliminary versions appear every four months, the execution time of our transformation is acceptable. The 870 resulting UML schemas contain 19815 classes, 55234 attributes, and 19053 associations. These models are UML-compliant, semantically identical to the original MIFs, and immediately available after the transformation process.

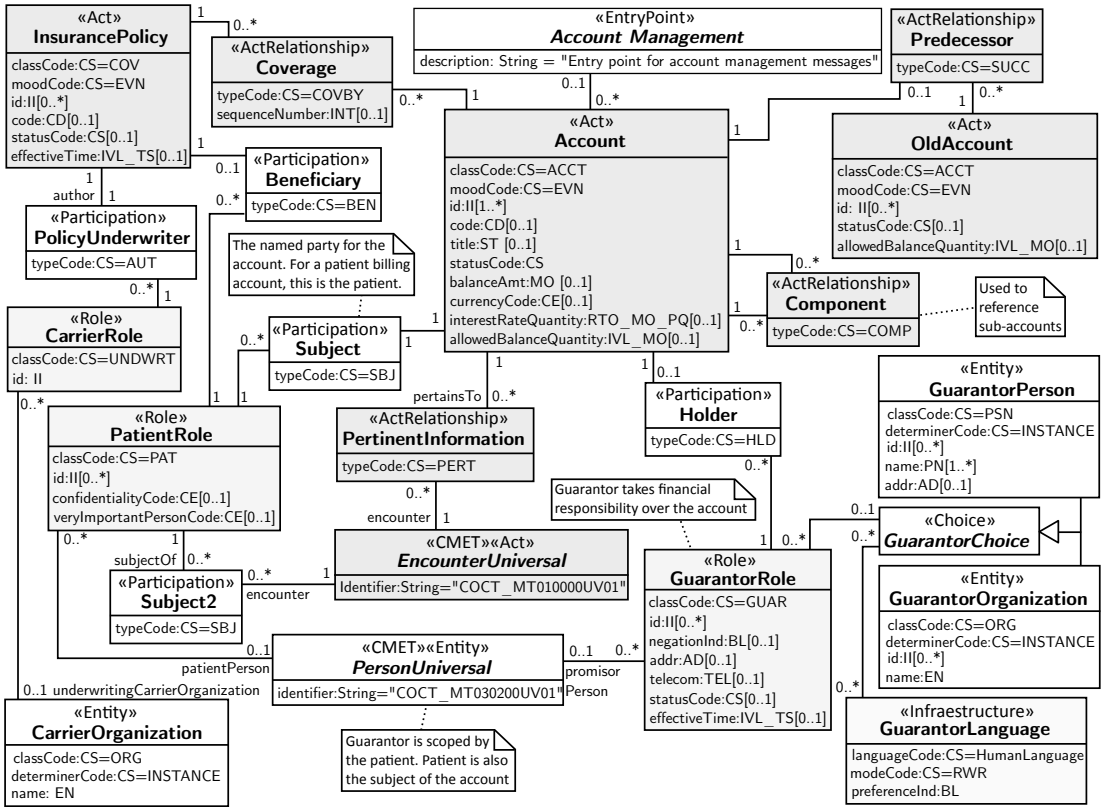


Figure 9.10. UML transformation of Patient Billing Account Domain Model (Fig. 9.3).

Figure 9.10 shows the complete UML version of the information model of Fig. 9.3 that holds the same original contents but using standard constructs. Note that the participations and act relationships of the HL7 V3 that were represented graphically as arrowed boxes are now common UML classes connected to other classes with binary associations.

Also, the choices and CMETs are processed following the aforementioned transformation rules. In the HL7 V3 version (see Fig. 9.3) there is a choice construction named *Guarantor-Choice* that includes the entities *GuarantorPerson* and *GuarantorOrganization*. In addition to it, there are two relationships with *GuarantorChoice* connecting it with *GuarantorRole* and

GuarantorLanguage. In the UML version (see Fig. 9.10) we have the same contents. The HL7 V3 choice is represented by an abstract class with two subclasses —the *GuarantorPerson* and the *GuarantorOrganization*. The *GuarantorRole* role is connected to the abstract class of the choice and the relationship with *GuarantorLanguage* is maintained. Consequently, the UML version holds the same contents but using standard constructs.

The analysis of the UML schemas that result from the transformation points out several modeling problems that should be addressed to improve the quality of the standard. By analyzing the resulting UML schemas, we have found modeling flaws with the names of concepts used in the HL7 V3 standard. For instance, the *Participation* named *author* of *InsurancePolicy* of Fig. 9.3 is named *PolicyUnderwriter* in Fig. 9.10. Similarly, there are two *Participation* classes, named *Subject* and *Subject2*, which indicate the same subject role of both *Account* and *EncounterUniversal* acts in Fig. 9.10. Since these participations are identical, only one of them should appear in the schema.

The adoption of common UML schemas benefits the usage of HL7 V3 with existing modeling tools[2] compatible with our proposed UML schemas —there is no need to invest on specific tools. Finally, there are lots of UML experts that may increase the final quality of the standard and solve the existing modeling flaws by analyzing the UML schemas and proposing improvements on their specification and development framework.

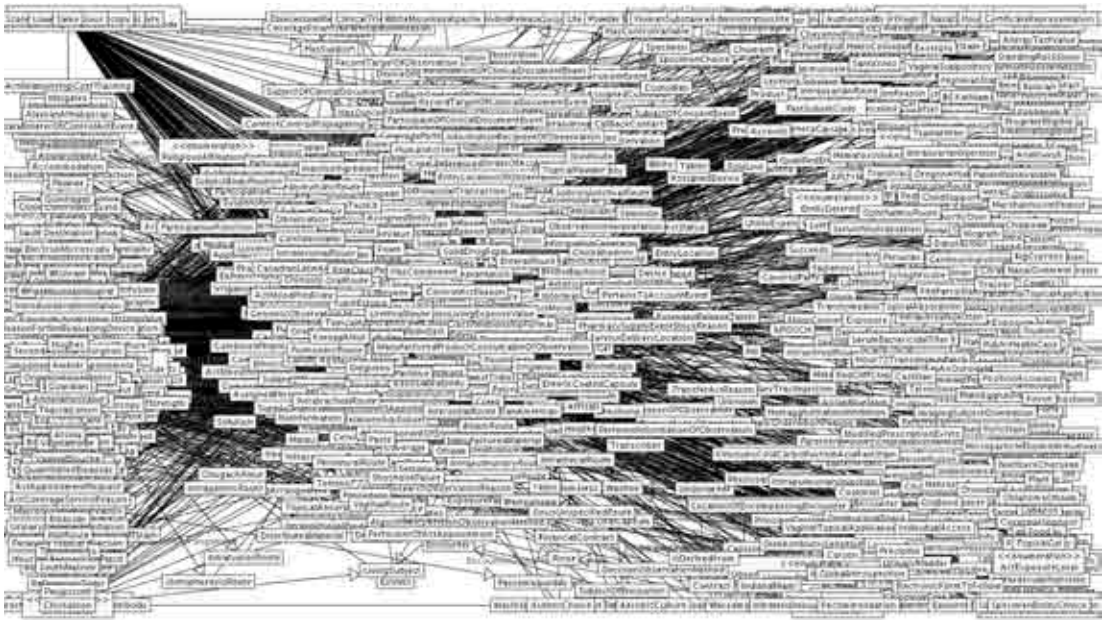


Figure 9.11. Conceptual Schema of the HL7 V3.

[2]Eclipse UML2 Compatibility List `http://wiki.eclipse.org/MDT-UML2-Tool-Compatibility`

9.5 The Filtering Methodology for HL7 V3

In this section we describe the changes our filtering methodology requires in order to be applied to the set of healthcare information models from the HL7 V3 standard. We review the characteristics of the new structure of the filtering method for HL7 V3, and analyze the definition of adapted metrics of relevance in order to compute the most interesting elements from HL7 V3 given a particular focus set. We also enumerate the filtering requests from our catalog that can be used to deal with these schemas, and show an example of application of one of the requests.

9.5.1 Structure of the Filtering Method for HL7 V3

As aforementioned in previous chapters, the main goal of the filtering methodology is to extract a reduced and self-contained view from the large schema, that is, a filtered conceptual schema with the knowledge of interest to the user. Figure 9.12 presents a comparison between the general structure of the filtering method —as explained in Ch. 5— and the structure of the filtering method when adapted to HL7 V3.

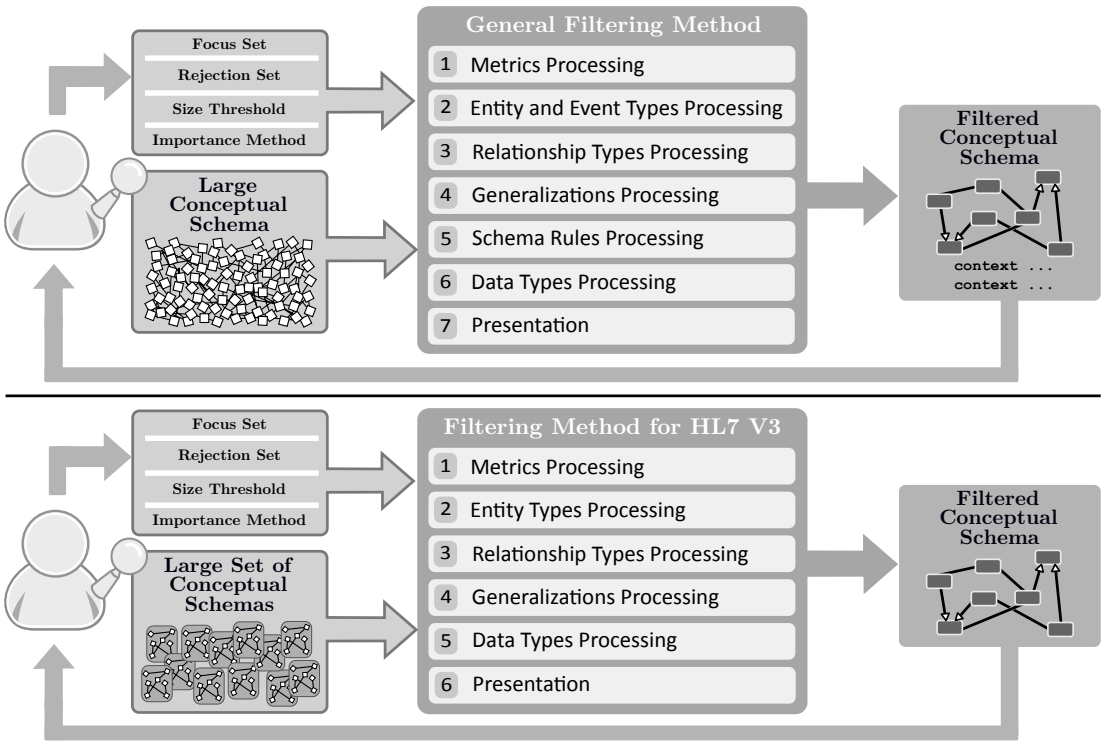


Figure 9.12. Comparison between general filtering method (top) and filtering method for HL7 V3 (bottom).

The main reason to adapt our filtering methodology to HL7 V3 is the topological structure of the conceptual schema of HL7 V3. Our general method is able to process a large conceptual schema, which is a single-file schema with large amounts of information including, among others,

entities, events, and relationship types. By contrast, the structure of the HL7 V3 is based on a large set of small-to-medium R-MIMs (Refined Message Information Model) that define the structure of healthcare messages and documents, and whose union creates a large conceptual schema.

These R-MIM models contain special constructions specified in a non UML-compliant modeling language, which difficults the learnability of the standard to software engineering experts, which are the ones that design and develop HL7 V3-based information systems. In addition, there are few tools that support V3 models, and their maturity level is low. Section 9.4 describes an automatic transformation to obtain UML schemas from each of the HL7 V3 R-MIMs of the standard (starting from their MIF representation). The union of the 870 transformed UML schemas results in a large conceptual schema, as shown in Fig. 9.11.

Consequently, we provide a UML version of the R-MIMs from the standard to broaden the audience of HL7 V3 and be able to load the UML schemas with existing tools. In addition, by using the UML version of the healthcare schemas we reduce the required changes in our filtering methodology and engine.

Therefore, we apply our filtering methodology to the resulting set of 870 UML schemas that are transformed from the HL7 V3 R-MIMs of the standard. The main issue here is to deal with a large set of small-to-medium schemas instead of working with a large one. Our filtering method needs to compute several relevance metrics in order to select and filter the most interesting elements from the set of transformed HL7 V3 schemas. The huge size of the union of 870 schemas makes it difficult to compute these metrics in an acceptable response time for a dynamic request/response environment. Since we want to construct an interactive filtering tool, computing the general importance of schema elements and their closeness to the focus set requires some special tuning in order to keep the efficiency and effectiveness levels of our proposal. To this end, Sect. 9.5.2 presents the definition of relevance metrics for the case of the UML schemas transformed from the HL7 V3 standard.

Additionally, Fig. 9.12 indicates that the number of filtering stages in the filtering method for HL7 V3 is lower than the number of stages in the general filtering method of Ch. 5. The R-MIMs behind HL7 V3 only define the structural subschema of a conceptual schema. Thus, the standard does not include the specification of event types in a behavioral subschema, and as a result, the second stage in the filtering method for HL7 V3 only deals with entity types from the structural subschema.

Also, HL7 V3 defines schema rules over the R-MIMs in order to constraint the graphical representation of entity and relationship types with additional textual conditions. The main problem here is that such schema rules are specified in an heterogeneous way. Exploring the R-MIMs it is possible to find schema rules defined in an OCL-like formal constraint language, and additional comments expressed in a non-processable natural language. Such schema rules should be corrected, normalized and rewritten in a standard constraint language in order to be easily included in our model transformation process. Since this task is out of the scope of this thesis, our HL7 V3-to-UML transformation does not take into account the schema rules, and as a result, the filtering stage number 5 in the general filtering method does not appear in the filtering method for HL7 V3 (bottom of Fig. 9.12).

In the following, we detail the internal modifications in our general filtering approach in order to take into account the specific characteristics of the large set of conceptual schemas obtained from the R-MIMs of HL7 V3.

9.5.2 Relevance Metrics for HL7 V3

As aforementioned in Sect. 9.4.3, the conceptual schema of the HL7 V3 standard is constructed as the union of a large set of small-to-medium schemas. Formally,

$$\mathcal{CS}_{HL7} = \biguplus_{i=1}^{n} \mathcal{CS}_i.$$

The 870 schemas ($n = 870$) that belong to HL7 V3 contain 19815 entity types, 55234 attributes, and 19053 relationship types. Furthermore, the specification of an entity type and their interrelationships with other entity types is spread out through several schemas, which means that there are entity types that are shared by more than one schema —that is the reason to use the multiset union $\uplus$.

The large size of the HL7 V3 schema $\mathcal{CS}_{HL7}$ makes it difficult to compute in real time the relevance metrics our filtering methodology requires. A common solution to this problem consists of pre-calculating the importance of all the entity types in the schema and all the minimum distances between each pair of entity types. However, if we are in a dynamic scenario where the schema evolves, this solution is not acceptable.

We propose the relevance computing approach depicted in Fig. 9.13, where the user selects a set of entity types of focus and our method automatically selects a subset of conceptual schemas of reduced size (pre-filtering). Concretely, the method selects those schemas $\mathcal{CS}_i$ from HL7 V3 that contain at least one entity type from the focus set. Thus, it is expected that the selected fragment $\mathcal{CS}_{\mathcal{S}} \subset \mathcal{CS}_{HL7}$ of the whole HL7 V3 is small enough to apply the relevance metrics in real time, and big enough to produce a good result.

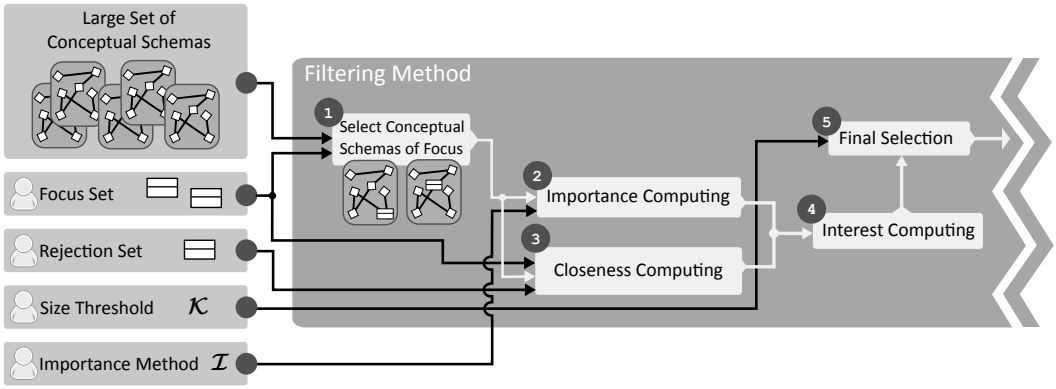


Figure 9.13. Relevance metrics processing when applied to HL7 V3.

In the following, we describe the changes in the relevance metrics in order to adapt them to the special case of HL7 V3. The rest of the method behaves as indicated in Sect. 9.5.1.

Importance computing

Our approach is based on the concept of importance. Second step consists of computing the importance of the entity types from the selected conceptual schemas of focus $\mathcal{CS}_{\mathcal{S}}$ with the importance method $\mathcal{I}$ selected in the input as shown in Alg. 9.4.

Algorithm 9.4. Compute Importance Ψ for HL7 V3.

```
1 for each e ∈ E_i ⊂ CS_S do
2   Ψ_HL7(e) = I(e)
3 end
```

In the general method introduced in Ch. 5 we compute the global importance Ψ of any entity of the large schema. Note that for the case of HL7 V3 the importance method $\mathcal{I}$ only takes into account the knowledge defined in the selected fragment $\mathcal{CS}_{\mathcal{S}}$ of $\mathcal{CS}_{HL7}$, as shown in Fig. 9.13.

Closeness computing

The importance metric is useful when a user wants to know which are the most important entity and event types, but it is of little use when the user is interested in a specific subset of elements of focus, independently from their importance. This step computes the measure of closeness of the entity types of the selected set of schemas $\mathcal{CS}_{\mathcal{S}}$ that are candidates to be included in the resulting filtered schema, with respect to the elements of the focus set $\mathcal{FS}$. A candidate entity type belongs to $\mathcal{CS}_{\mathcal{S}}$ and is neither in the focus set nor in the rejection set.

Algorithm 9.5. Compute Closeness Ω for HL7 V3.

```
1 for each e ∈ {{E_i ⊂ CS_S} \ {FS ∪ RS}} do
2   Ω_HL7(e, FS) = |FS| / Σ_{e'∈FS} δ(e, e')
3 end
```

Intuitively, the closeness of e should be directly related to the inverse of the distance δ of e to the entities of the focus set. Those entity types that are closer to more elements of $\mathcal{FS}$ will have a greater closeness. Note that the distance δ computes to the minimum topological distance between two entity types in $\mathcal{CS}_{\mathcal{S}}$. Given two entity types e_i, e_j, if there is no path in $\mathcal{CS}_{\mathcal{S}}$ between them, then we define $\delta(e_i, e_j) = \sum_{\mathcal{E}_i \subset \mathcal{CS}_{\mathcal{S}}} |\mathcal{E}_i|$.

Interest computing

What is needed then is a metric that measures the interest of a candidate entity type e with respect to the focus set $\mathcal{FS}$. This metric should take into account both the adapted importance $\Psi_{HL7}(e)$ and the adapted closeness measure of $\Omega_{HL7}(e, \mathcal{FS})$. For this reason, the filtering method computes such measure as shown in Alg. 9.6.

Algorithm 9.6. Compute Interest Φ for HL7 V3.

```
1 for each e ∈ {{E_i ⊂ CS_S} \ {FS ∪ RS}} do
2   Φ_HL7(e) = α × Ψ(e) + (1 − α) × Ω(e, FS)
3 end
```

9.5.3 Catalog of Filtering Requests for HL7 V3

Chapter 6 presents a catalog of filtering requests applicable to large conceptual schemas specified in UML/OCL. The catalog presents six filtering requests and describes the characteristics of each of them. The adaptation of the filtering methodology to the case of the conceptual schemas that belong to the HL7 V3 standard produces changes in the aforementioned catalog. The resulting set of filtering requests that belong to the adapted catalog of filtering requests for HL7 V3 contains the following:

- **Filtering Request for Entity Types ($\mathcal{F}_1$)**: The user focuses on a set of entity types from a large schema. The request obtains a filtered conceptual schema that includes the combination of the initial entity types with the elements of interest. The original request also focuses on relationship types, but since relationships in HL7 V3 schemas are purely defined as binary connectors between two concepts, without a specific semantic, we do not take them into account as members of the focus set. The rest of the request behaves identically as the original one, defined in Sect. 6.3.1 of Ch. 6.
- **Filtering Request for a Conceptual Schema ($\mathcal{F}_4$)**: The user focuses on a small fragment from the large schema. The user is aware of the elements that conform such fragment or she has accessed them via previous requests. As output, the user obtains a filtered schema that includes the combination of the elements of the fragment surrounded with elements of interest. The request behaves identically as the original one, defined in Sect. 6.3.4 of Ch. 6.
- **Filtering Request for Contextualized Entity Types ($\mathcal{F}_6$)**: The user focuses on a set of entity types. The user contextualize them by means of a function to reduce or limit the characteristics defined over such types. As output, the user obtains a filtered schema with the selected entity types and the elements of interest taking into account the contextualization. The original request also focuses on event types, but since HL7 V3 schemas are purely defined as structural schemas —there is no behavioral subschema—, there are no events specified in the schemas of the HL7 V3 standard. The rest of the request behaves identically as the original one, defined in Sect. 6.3.6 of Ch. 6.

The filtering request for schema rules ($\mathcal{F}_2$), which is defined in Sect. 6.3.2 of Ch. 6, is not useful to HL7 V3. The reason is that since the schema rules are specified in an heterogeneous way in HL7 V3 information models, they should be corrected, normalized and rewritten in a standard constraint language in order to be easily included in our model transformation process. Therefore, the UML conceptual schemas obtained from the HL7 V3 R-MIMs does not contain schema rules, and thus, the user cannot focus on them.

The filtering request for event types ($\mathcal{F}_3$), and the filtering request for context behavior of entity types ($\mathcal{F}_5$), which are correspondingly defined in Sect. 6.3.3 and Sect. 6.3.5 of Ch. 6, are not useful to the large set of conceptual schemas of the HL7 V3. The reason is that there is no event specification within the HL7 V3 R-MIMs, and thus, the user cannot focus on them using $\mathcal{F}_3$ nor obtain the set of event types of interest taking into account a set of entity types of focus using $\mathcal{F}_5$.

9.5.4 Example of Application of a Filtering Request to HL7 V3

The first step of our filtering method consists in preparing the required information to filter the HL7 V3 schemas according to the user preferences. Basically, the user focus on a set of entity types (focus set) she is interested in and our method surrounds them with additional knowledge from the HL7 V3 schemas. Therefore, it is mandatory for the user to select a non-empty initial focus set $\mathcal{FS}$.

In this section we show an example where the user wants to see what is the knowledge the HL7 V3 schemas have about the entity types *Patient* and *Appointment*. Therefore, the user defines $\mathcal{FS} = \{Patient,\ Appointment\}$. Starting from this focus set, our filtering method retrieves the knowledge represented in the schemas about *Patient* and *Appointment* that is likely to be of more interest to the user.

For the sake of efficiency, our method selects a subset of conceptual schemas of reduced size (pre-filtering) containing at least one entity type from the focus set. Concretely, for the case of $\mathcal{FS} = \{Patient,\ Appointment\}$ our method retrieves the following 151 HL7 V3 schemas, which represent a 17% of the total amount of schemas in the standard:

Patient
COCT_MT010000UV01
COCT_MT010004UV02
COCT_MT050004UV01
COCT_MT060000UV01
COCT_MT080000UV
COCT_MT080100UV
COCT_MT220300UV
COCT_MT230100UV
COCT_MT260003UV
COCT_MT290000UV06
COCT_MT290002UV06
COCT_MT290004UV06
COCT_MT300000UV04
COCT_MT300001UV04
COCT_MT510000UV06
COCT_MT510005UV06
COCT_MT530000UV
COCT_MT610000UV06
COCT_MT830120UV05
COCT_MT910000UV
COCT_MT930000UV
COCT_MT970000UV
FIAB_MT010101UV02
FIAB_MT010102UV02
FIAB_MT010103UV02
FIAB_MT010104UV02
FIAB_MT010105UV02
FIAB_MT020101UV02
MCAI_MT700201UV01
MCAI_MT705201UV01
MCAI_MT900001UV01
MFMI_MT700701UV01
MFMI_MT700702UV01
MFMI_MT700711UV01
MFMI_MT700712UV01
MFMI_MT700721UV01
MFMI_MT700722UV01
POCG_MT000010UV01
POCG_MT000040UV01
PORP_MT050032UV03
PORR_MT040002UV01
PORR_MT040003UV01
PORR_MT040004UV01
PORR_MT040005UV01
PORR_MT040011UV01
PORR_MT040012UV01
PORR_MT040061UV01
PORR_MT040062UV01
PORR_MT049006UV01
PORR_MT049007UV01
PORR_MT049008UV01
PORR_MT049009UV01
PORR_MT049010UV01
PORR_MT049011UV01
PORR_MT049012UV01
PORR_MT100001UV01
PORR_MT150101UV01
PORT_MT020001UV01
PORT_MT020002UV01
PORT_MT030001UV01
PORT_MT090001UV01
PORT_MT090002UV01
POTD_MT000001UV02
PRPA_MT101301UV02
PRPA_MT101302UV02
PRPA_MT101303UV02
PRPA_MT101305UV02
PRPA_MT101310UV02
PRPA_MT201301UV02
PRPA_MT201302UV02
PRPA_MT201303UV02
PRPA_MT201304UV02
PRPA_MT201305UV02
PRPA_MT201310UV02
PRPA_MT202301UV02
PRPA_MT202302UV02
PRPA_MT202303UV02
PRPA_MT202310UV02
PRPA_MT301011UV02
PRPA_MT302011UV02
PRPA_MT303011UV02
PRPA_MT401001UV02
PRPA_MT401002UV02
PRPA_MT401003UV02
PRPA_MT401004UV02
PRPA_MT401005UV02
PRPA_MT402001UV02
PRPA_MT402002UV02
PRPA_MT402003UV02
PRPA_MT402004UV02
PRPA_MT402008UV02
PRPA_MT402009UV02
PRPA_MT403001UV02
PRPA_MT403002UV02
PRPA_MT403003UV02
PRPA_MT403004UV02
PRPA_MT403005UV02
PRPA_MT404001UV02
PRPA_MT404002UV02
PRPA_MT404003UV02
PRPA_MT404004UV02
PRPA_MT404005UV02
PRPA_MT411001UV02
PRPA_MT411002UV02
PRPA_MT411003UV02
PRPA_MT411004UV02
PRPA_MT412001UV02
PRPA_MT414001UV02
PRPA_MT414002UV02
PRPA_MT414003UV02
PRPA_MT414004UV02
PRPA_MT900101UV02
PRPA_MT900102UV02
PRPA_MT900350UV02
PRPM_MT301010UV01
PRPM_MT303010UV01
PRPM_MT306110UV01
PRPM_MT309000UV01
PRPM_MT401010UV01
PRPM_MT403010UV01
PRPM_MT406110UV01
PRPM_MT409000UV01
PRSC_MT010101UV01
PRSC_MT010201UV01
QUQI_MT120001UV01
RCMR_MT000001UV02
RCMR_MT000002UV02
RCMR_MT010001UV01
RCMR_MT010002UV01
REPC_MT000100UV01
REPC_MT000120UV01
REPC_MT000130UV01
REPC_MT000200UV
REPC_MT000300UV01
REPC_MT000301UV
REPC_MT000322UV01
REPC_MT000323UV01
REPC_MT000324UV01
REPC_MT002000UV01
REPC_MT002600UV01
REPC_MT003000UV01
REPC_MT004000UV01
REPC_MT004005UV01
REPC_MT004009UV01
REPC_MT610001UV01
REPC_MT610002UV01

Appointment
COCT_MT020000UV01
PRSC_MT010101UV01
PRSC_MT010201UV01
PRSC_MT020101UV01
PRSC_MT020201UV01

Figure 9.14. HL7 V3 models that contain the entity types Patient and Appointment.

Consequently, our method computes the relevance metrics from Sect. 9.5.2 to the previous subset of HL7 V3 schemas. Once the method obtains the metrics of importance and closeness, the next step consists of computing the interest (Φ) for each entity type from the subset of schemas out of the focus set $\mathcal{FS}$. As previously shown, the interest $\Phi(e)$ of a candidate entity type e to be included in the resulting filtered schema is a linear combination of the importance $\Psi(e)$ and the closeness $\Omega(e, \mathcal{FS})$ taking into account the balancing parameter α.

Given that the user does not include any entity type in the rejection set ($\mathcal{RS} = \emptyset$) and she selects a size threshold $\mathcal{K} = 12$, Tab. 9.1 shows the top-10 entity types with a greater value of interest when the user defines $\mathcal{FS} = \{Patient,\ Appointment\}$ and $\alpha = 0.5$.

Results in Tab. 9.1 indicate that included within the top-10 there are entity types that are directly connected to all members of the focus set $\mathcal{FS} = \{Patient,\ Appointment\}$ as in the case of *Subject* ($\Omega(Subject, \mathcal{FS}) = 1.0$) but also entity types that are not directly connected to any entity type of FS (although they are closer).

Rank	Entity Type (e)	Importance $\Psi(e)$	Distance $d(e,\ Patient)$	Distance $\delta(e,\ Appointment)$	Closeness $\Omega(e, \mathcal{FS})$	Interest $\Phi(e, \mathcal{FS})$
1	Organization	1.72	1	3	0.5	1.11
2	Person	1.22	1	3	0.5	0.86
3	ServiceDeliveryLocation	0.79	2	2	0.5	0.65
4	AssignedPerson	0.72	2	2	0.5	0.61
5	Subject	0.11	1	1	1.0	0.56
6	ManufacturedDevice	0.55	2	2	0.5	0.53
7	Location	0.26	3	1	0.5	0.38
8	ReusableDevice	0.19	3	1	0.5	0.35
9	Performer	0.13	3	1	0.5	0.32
10	Author	0.12	3	1	0.5	0.31

Table 9.1. Most Interesting classes with regard to $\mathcal{FS} = \{Patient,\ Appointment\}$.

Finally, the last step of the method receives the top-interest set of entity types from the previous step and puts it together with the entity types of the focus set $\mathcal{FS}$ in order to create a filtered conceptual schema with the entities of both sets. The main goal of this step consists in filtering information from the HL7 V3 schemas involving entity types in the filtered schema. To achieve this goal, the method explores the associations, and generalization/specialization relationships in the HL7 V3 schemas that are defined between those selected entity types and includes them in the filtered conceptual schema to obtain a connected schema.

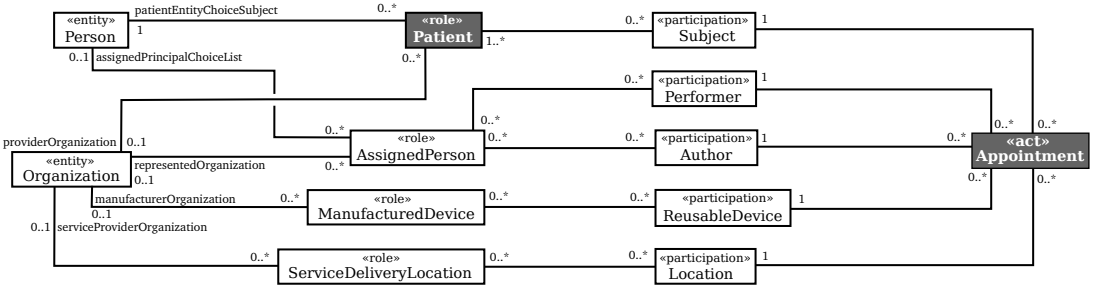


Figure 9.15. Filtered conceptual schema for $\mathcal{FS} = \{Patient,\ Appointment\}$.

The filtered schema for $\mathcal{FS} = \{Patient,\ Appointment\}$ is shown in Figure 9.15 (attributes are hidden for the sake of simplicity). This schema represents the arguments of a scheduling service. It has a central focus on the Appointment entity type, which represents the activity being scheduled. The subject (usually the patient) participates in the appointment. Note that the cardinality is 1..* in the side of patient to allow for group appointments. The performer, location, and reusable device entity types associate the resources being reserved in the appointment. Also, the author is the person who originated the appointment and who must confirm changes or substitutions.

9.6 Experimentation

Our filtering method and prototype tool provide support to the task of extracting knowledge from the HL7 models, which has been done manually or with little computer support.

Finding a measure that reflects the ability of our method to satisfy the user is a complicated task. However, there exists related work [5, 123] about some measurable quantities in the field of information retrieval that can be applied to our context:

- The ability of the method to withhold non-relevant knowledge (*precision*).
- The interval between the request being made and the answer being given (*time*).

9.6.1 Precision Analysis

A correct method must retrieve the relevant knowledge according to the user preferences. The precision of a method is defined as the percentage of relevant knowledge presented to the user. In our context, we use the concept of precision applied to HL7 universal domains (specified with D-MIM's). Each domain contains a main entity which is the central point of knowledge to the users interested in such domain. The other entities presented in the domain conform the relevant knowledge related to the main entity.

HL7 professionals interested in a particular domain decide about the knowledge to incorporate in it through ballots. Thus, a common situation for a user is to focus on the main entity of a healthcare domain and to navigate through the D-MIM to understand its related knowledge. To know the precision of our method, we simulate the generation of a D-MIM from its main entity. We define a single-entity focus set with such entity and set $\mathcal{K}$ with the size of the domain. This way, we will obtain a filtered conceptual schema with the same number of entities as such domain.

In one iteration of our method, we obtain two groups of entities within the resulting filtered conceptual schema: the relevant entities to the user, that is, the ones that were originally defined in the D-MIM by experts, and the non-relevant ones. The precision of the result is defined as the fraction of the relevant entities over the total $\mathcal{K}$.

To refine the obtained result, the non-relevant entities are included in a rejection set $\mathcal{RS}$ and the method is executed again taking into account $\mathcal{RS}$. It is expected that the filtered conceptual schema that results from this step will have a greater precision. This manner, at each iteration non-relevant entities to the user are rejected, and we know that in a finite number of steps our filtering method will obtain all the entities of the original domain. The smaller the number of required iterations until getting such domain, the better the method.

Figure 9.16 shows the number of iterations needed to reach the maximum precision for four of the HL7 V3 normative domains. Note that right side of Figure 9.16 zooms in the first five iterations. The test reveals that to reach more than 80% of the relevant entities of a domain, only three iterations are required. The results are significant and extensible to other domains.

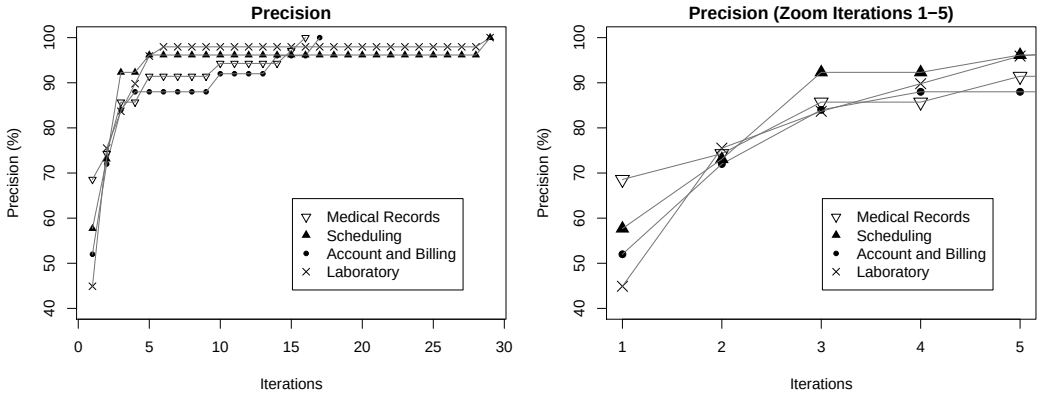


Figure 9.16. Precision analysis for a set of four significant HL7 V3 domains.

9.6.2 Time Analysis

It is clear that a good method does not only require precision, but it also needs to present the results in an acceptable time according to the user. To find the time spent by our method it is only necessary to record the time lapse between the request of knowledge, i.e. once a focus set $\mathcal{FS}$ has been indicated by the user, and the receipt of the filtered conceptual schema.

It is expected that as we increase the size of the focus set, the time will increase linearly. Our method computes the distances from each entity in the focus set to all the rest of entities. This computation requires the same time (in average) for each entity in the focus set. Therefore, the more entities we have in a focus set, the more time our method spends in computing distances.

In our experimentation, we set our prototype tool to apply the filtering method several times with an increasing number of entities in the focus set. The average results for sizes from a single-entity focus set up to a 40-entities focus set are presented in Figure 9.17. According to the expected use of our method, having a focus set $\mathcal{FS}$ of 40 entities is not a common situation (although possible). Sizes of focus set up to 10 entities are more realistic, in which case the average time does not exceed one second.

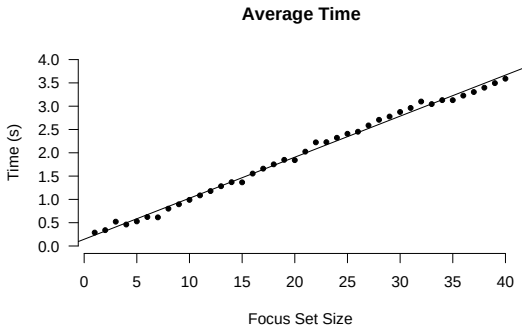


Figure 9.17. Time analysis for different sizes of $\mathcal{FS}$.

9.7 Summary

In Ch. 5 we introduced the filtering method that is the central contribution of this thesis. Furthermore, we presented a catalog of filtering requests to deal with large conceptual schemas in several filtering scenarios in Ch. 6, and we also showed their utility when applied to a set of real-world conceptual schemas, as described in Ch. 7. Consequently, in this chapter we have described a way our tool can be adapted to fulfill the requirements of HL7 V3, the well-known model-based standard from the healthcare domain.

HL7 V3 information models are very large. The wealth of knowledge they contain makes them very useful to their potential target audience. However, the size and the organization of these models make it difficult to manually extract knowledge from them. This task is basic for the improvement of services provided by HL7 affiliates, vendors and other organizations that use those models for the development of health systems.

What is needed is a tool that improves the usability of HL7 models for that task. We have adapted our filtering method from Ch. 5 in order to accept a large set of UML schemas as filtering target. We have also proposed an automatic transformation approach to convert the schemas from the HL7 V3 standard into equivalent UML schemas on order to benefit from contributions from the software engineering area.

Our method makes it easier to automatically extract knowledge from the UML version of the HL7 V3 models. Input to our method is the set of entity types the user is interested in. The method computes the interest of each entity type in the HL7 models with respect to that set taking into account its importance and closeness. Finally the method selects the most interesting entity types from those models, including their defined knowledge in the large schema (e.g. associations, attributes, generalization relationships).

The main question here is *Is it required to transform HL7 V3 R-MIMs into UML in order to be able to apply our filtering methodology?* The answer is clear: no, it is not. Our filtering approach can be applied to any object-oriented model based on a graph structure with a set of nodes connected through a set of edges. Since the HL7 V3 graphical modeling language follows these guidelines, our approach can be directly used (with minor adaptations in the filtering engine). However, we propose a transformation from HL7 V3 R-MIMs to equivalent UML schemas in order to improve the understandability of the standard for those interested on it, and the usability of the HL7 V3 information models with a wide range of existing UML tools that provide additional support and functionalities to adopt model-driven engineering techniques into the healthcare domain. Moreover, we have found modeling flaws in the standard that conceptual modelers can solve in the UML version and, therefore, help in the final development of HL7 V3.

The experiments we have done clearly show that the proposed method and its associated tool provides an easier way to extract knowledge from the models. Concretely, our filtering engine recovers more than 80% of the knowledge of a D-MIM in three iterations, with an average time per iteration that for common uses does not exceed one second. The conclusion we extract from the previous experimentation is that our filtering methodology helps users when exploring the large amount of schemas HL7 V3 contains.

I think and think for months and years.
Ninety-nine times, the conclusion is false.
The hundredth time I am right.

Albert Einstein (1879-1955)

10

Conclusions and Future work

The usability and understandability of large conceptual schemas has become a relevant issue in the field of conceptual modeling. Since organizations, and the information systems that support them, require an increasing amount of information processing and knowledge analysis, the size of their conceptual schemas has grown until reaching complexity levels that make it difficult to users to manage them. Throughout the different chapters of this thesis, we have provided methods and tools to help reduce the impact of this problem.

The chapter starts with a summary of the results of the thesis. Section 10.1.1 aligns our research with the problem of conceptual modeling in the large. Section 10.1.2 reviews the contribution on the analysis of relevance metrics defined in the context of large conceptual schemas. Section 10.1.3 indicates the benefits of the filtering approach and Sect. 10.1.4 describes the advantages our catalog of filtering requests provides to users that need to explore portions of the knowledge within a large schema. Section 10.1.5 reviews the results of the experimentation with three real filtering scenarios where a user may use our approach, and Sect. 10.1.6 shows the consequences of using a prototype to validate our research. Since our work opens new research directions, Sect. 10.2 enumerates extension points to our current research approach and further work that could provide additional benefits to the users of large conceptual schemas. Finally, Section 10.3 lists the publications and research impact of the thesis.

10.1 Summary of Results

This thesis presents a new approach to help and guide users on exploring the knowledge defined in both structural and behavioral components of large conceptual schemas by filtering. This section summarizes the main contributions and results of the research and approach presented in the previous chapters of this document.

10.1.1 The problem of conceptual modeling in the large

In Ch. 1 we stated that the main objective of this thesis is to provide a filtering engine of very large conceptual schemas to help users to easily extract from them the most relevant knowledge for a particular purpose. Also, we included the sub-goals of identifying, studying and describing several properties of relevance for elements of large conceptual schemas, presenting useful feedback to the user, and evaluating the usability of the filtering engine.

Following the guidelines of the design-science research paradigm, we studied the relevance of the problem of conceptual modeling in the large by analyzing (Ch. 3) the different approaches and contributions in the existing literature. Also, we found several requests for contributions in this area (see Sect. 3.2) that justify the existence of our work.

The major contributions in the scope of large conceptual schemas consist of classifying the elements of the schema into groups, or clusters, according to a similarity function (see Custering in Sect. 3.3.1). There are also methods that apply a ranking function to the elements of the schema in order to obtain an ordered list, also called ranking, according to the general relevance of the schema elements (see Relevance in Sect. 3.3.2). And the last family of methods compute a reduced general schema from the large original one with the elements that are more relevant but also with a a high degree of coverage of the schema (see Summarization in Sect. 3.3.3). The retrieved knowledge by all these families of methods is general, i.e., it is always the same —it does not change unless the schema changes— and it is not affected by specific user interests.

This thesis explores a new approach based on information filtering. As far as we know, there is no research study or contribution in the present literature about the application of specific information filtering methods and techniques to conceptual schemas. The aim of information filtering is to expose users to only information that is relevant to them. Apart from the schema itself, information filtering methods require as input a representation of the user information need or interest in form of knowledge request.

The filtering-based family of methods provides users with a dynamic request/response interaction that simplifies the knowledge extraction process. Other alternatives provide static access to their output, which in many cases contain only a fragment of the whole knowledge that may be out of the scope of interest to the user. Therefore, the development of a filtering approach to deal with large conceptual schemas covers a relatively unexplored area in the literature and helps to providing a more dynamic and flexible solution in comparison to the existing methodologies that contribute to this topic.

10.1.2 Analysis of relevance metrics

The filtering methodology of the thesis uses a core of relevance metrics to select the most interesting elements from the schema with respect to a set of user-selected elements of focus. In this thesis we focus on objective metrics, which are independent from subjective evaluations of users and modelers.

Intuitively, it seems that an objective metric of the importance of an element in a given schema should be related to the amount of knowledge that the schema defines about it. The more (less) knowledge a schema defines about an element, the more (less) important should be that element in the schema. Adding more knowledge about an element should increase (or at least not decrease) the relative importance of that element with respect to the others.

As far as we know, the existing metrics in the literature for entity and event type importance are mainly based on the amount of knowledge defined in the schema, but only take into account the number of attributes, relationship types and specialization/generalization relationships. Surprisingly, none of the methods we are aware of take into account additional knowledge about entity and event types defined in a schema that, according to the intuition, could have an effect on their relevance. A complete schema includes also cardinalities, general constraints, derivation rules and the specification of event effects, all of which contribute to the knowledge about entity and event types.

Our approach extends the definition of a set of seven existing methods from the literature with that additional knowledge (mostly obtained by reification of association classes and processing OCL expressions). We have implemented the seven methods described in Ch. 4, both the original and the extended versions. We have then evaluated the methods using three distinct case studies: osCommerce, the UML metaschema, and EU-Rent. The two main conclusions are: (1) among the original versions of the methods, the methods of choice are those based on the link analysis following the same approach than Google's PageRank; and (2) the extended versions of most methods produce remarkably similar results, which does not happen in the original version. In addition to it, we adapted these methods to be able to work with relationship types.

The relevance methods to compute the importance of entity and event types (and, possibly, other elements in the schema) produce a general ranking of elements that does not change unless the definition of the schema itself changes. Therefore, all the users interested in exploring a large conceptual schema will obtain the same ranking regardless of their specific knowledge requirements. The importance metric is useful when a user wants to know which are the most important entity or event types, but it is of little use when the user is interested in a specific subset of entity or event types, independently from their importance. What is needed then is a metric that measures the interest of a candidate entity or event type with respect to a user-selected set of elements of focus. Our approach defines this metric as a combination of any of the general relevance of an entity or event type in the schema, and its closeness to the set of focus. The interest metric is the key measure for our filtering methodology, which provides users with good-enough feedback without requiring the intervention of a domain expert.

10.1.3 The filtering approach

At present, conceptual schemas are gaining more presence in the software engineering field and beyond. Our proposal contributes to the expansion of conceptual schemas by the study of its characteristics and the description of the structure and components of a filtering methodology for large conceptual schemas. Even though various authors proposed similar solutions to the field of complex networks and graphs, the approach of using information filtering to explore large conceptual schemas has not been previously explicitly formulated as in this thesis.

According to the multi-layer model of Hanani et. al [102] to define and classify information filtering approaches, the characteristics of our method are:

- **Initiative of Operation:** our filtering method is *passive* because do not pushes results without the direct intervention of the user, who has to focus on a set of elements from the large schema to start the interaction.
- **Location of Operation:** our proposed filtering methodology is located at the *information source*. The server side of the filtering engine keeps the knowledge from the large schema an accepts the specific filtering requests of clients that indicate a focus set and want a small schema as the filtering result.
- **Filtering Approach:** the proposed filtering method takes a focus set of selected schema elements that represents the interest point of a user and indexes the large conceptual schema in order to extract a small fragment of interest. This interest-based approach uses the relevance metrics of Ch. 4 to construct the different stages of a *cognitive* filtering process, based on the correlation between the content of the data items and the user.
- **Method of Acquiring Knowledge on Users:** our filtering methodology follows an explicit process of acquiring knowledge on users based on *user interrogation*. The method expects the intervention of the user to obtain the filtering preferences that conform the input of the filtering process.

We formally define in Ch. 5 the inputs and output of a general filtering method for large conceptual schemas, and each of the seven stages the methods follows to obtain a filtered conceptual schema. We describe the particularities of projection of relationship types and indirect generalization relationships in order to reduce the final size of the resulting schema. The details of the different algorithms used are also provided in order to be able to replicate our results.

10.1.4 The catalog of filtering requests

The generic approach of our filtering methodology requires the definition of a focus set as input of the filtering process. Since different kinds of elements must be part of the focus set, it is necessary to have specific filtering requests to provide different filtering behaviors. Our filtering methodology describes the general characteristics of a regular and systematic way of accomplishing the user-driven process to extract knowledge from a large conceptual schema.

What is needed is to define a set of concrete filtering requests that inherit the guidelines and main organization proposed by the previous methodology but adapting it to the particular filtering needs a user may have when working with a large schema. A filtering request for a large conceptual schema is a specific knowledge extraction request that automatically obtains a portion of the entire knowledge of small size and high relevance to the user in relation to the schema elements in the user focus of interest. We consider each filtering request a user may take advantage of, as a concrete instantiation of our filtering method to be effectively used under certain filtering circumstances.

A catalog of filtering requests has not been previously explicitly formulated as in this thesis. Our catalog provides requests for the most common filtering situations a user may face when dealing with a large conceptual schema. A user that wants to extract knowledge from a large conceptual schema may focus on entity, event or relationship types and then obtain the most interesting fragment of the large schema with relation to the selection. Or maybe the user wants to obtain the event types where a set of entity type of focus participates. Or the user focuses on a set of schema rules in order to obtain the elements affected by their definition (in OCL). Or, even the user may want to focus on a small fragment of the large schema and then obtain additional elements of interest to know more about that fragment. Or, finally, the user may want to focus on a set of entity and event types and obtain a contextualized schema with the elements of interest taking into account some contextualization constraints. All these situations are covered through our catalog (Ch. 6).

10.1.5 Experimentation with real case studies

A design artifact is complete and effective when it satisfies the requirements and constraints of the problem it was meant to solve. We have experimented each of the filtering requests from the catalog in the following experimental case studies (Ch. 7):

- A comparison between two large conceptual schemas representing the knowledge of two frameworks for e-commerce applications, the osCommerce and the Magento.
- An exploration of the behavioral components of a large conceptual schema for a car rental system, the EU-Rent, in order to understand the specific functionality of the system.
- An exploration of the metaschema of the Unified Modeling Language, which is guided by the contents of the UML Superstructure specification document.

The overall goals of these case studies are: (1) analyzing the viability of using our filtering methodology to compare the specification of several concepts from the e-commerce domain in two different conceptual schemas, (2) characterizing the functionalities of a real conceptual schema through the application of filtering request to its behavioral subschema, (3) identifying filtering patterns in a formal exploration following the contents of a normative specification of a model-based standard, and (4) using the lessons learned to improve and refine the filtering method and filtering requests.

We have then evaluated the efficiency and effectiveness of these filtering requests by using the conceptual schemas of the previous case studies. The results show that in most cases our method achieves a size reduction greater than 70% in the number of schema elements to explore when using filtered schemas instead of manually exploring the large schema, with an average time per request that is short enough (milliseconds) for the purpose at hand.

In addition to it, we have adapted our filtering methodology to a real-case large conceptual schema from the healthcare domain: the HL7 V3 (Ch. 9). We have translated the HL7 V3 schemas into UML/OCL and propose modifications to our general information filtering methodology in order to take advantage of the particular aspects of HL7 V3 to improve the quality of the produced results.

The experimentation and evaluation of our filtering methodology ensures the benefits provided to users that use filtering instead of other existing techniques to deal with large conceptual schemas.

10.1.6 Implementation of the proposal

Throughout this thesis we have followed the design-science research methodology. The fundamental principle of design-science research is that knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact. Consequently, we have developed an artifact to evaluate the benefits of our research approach. Our artifact consists of a filtering engine that implements the specific filtering requests in a web-based environment (Ch. 8.

Our engine contains a core that is responsible for maintaining and access the characteristics of a large conceptual schema. In addition, the core is under control of the filtering requests that require querying the schema in order to serve the information needs of the user. The conjunction of the specific information filtering requests and the core of the engine is implemented as a web service. This architectural decision allows an easy interaction with web clients and increase the technology independence, and therefore, the usability of the overall system.

The implementation of our filtering methodology is not unique. There are several ways of designing and coding a filtering system following our catalog of filtering requests. However, our proposal provides a minimum working application that helps the user on her task of extracting fragments of knowledge from a large schema taking into account the specific point of view and interest of the user. In order to communicate the benefits of our filtering engine, we have demonstrated its functions in international conferences [133, 131].

Finally, our service-oriented approach provides ways to easily extend the functionalities of the filtering engine (e.g., implement a new filtering request to fulfill specific needs of a particular domain, design a new client view that allows the user to explore additional knowledge of the schema, include a new interaction point to extend the user interaction with a particular view) without changing the existing implementation.

10.2 Future Work

There are different questions arising from the results of this thesis and that can be treated as future work. Here we point out the most interesting ones.

10.2.1 Extend the filtering catalog

Our filtering catalog contains six filtering requests that cover a wide range of filtering scenarios where our approach can help users on dealing with the size and complexity of large conceptual schemas. However there could be situations in which a user requires a specific filtering request that is not part of our catalog. As an example, a user may want to explore the elements in the schema that reference or use a particular data type to analyze the impact of modifying it. In the same way, a user may want to obtain the list of schema rules that use a specific attribute of an entity or event type, in order to know whether the attribute is of high relevance or not. The formal definition of each of the stages that conform a filtering request allows to easily extend the catalog with new filtering requests that can extend the behavior of the existing ones. We plan to analyze additional filtering scenarios, prioritize them, and extract new filtering requests from that analysis.

10.2.2 Extend the relevance metrics

In this thesis, we have based the core of our filtering methodology on relevance metrics to obtain those elements that are of interest to the user at a given point in time. The relevance metrics we have studied are extracted from the existing literature. We have extended the knowledge from the large schema they take into account in their process, by using reification and analyzing the OCL expressions that specify schema rules.

Even though we have improved the definition of relevance metrics by covering a big amount of knowledge from the schema, it is possible to complete that definition with additional components that some large schemas contain. There are large conceptual schemas that define, apart from the structural and behavioral subschemas, a third component that contains the instances of the entity, relationship, and event types the large schema has at a given time. Such instances conform the information base of the large schema and could be used in the relevance process. It is possible to think that if a schema element that has a big amount of instantiations, it will be of high relevance. We plan to study the effect of instances in the general relevance of schema elements.

10.2.3 Validation with real users

Even though we have demonstrated that our filtering methodology is a valid approach when dealing with large conceptual schemas in a set of filtering scenarios, only real users can say whether it is useful or in the contrary if it is unusable for their purpose. The main question we

would like to address to users is *do you feel more confident using our method? or do you prefer another approach from the literature?*. We plan to carry out experimentation with several group of users. The task of users will be to explore the knowledge defined within a large conceptual schema, some of them will do so using our filtering approach and the others using other methods. Other experimentations with different conceptual schemas of several sizes are planned in order to identify the threshold from which the users may not explore the schemas properly because of their size.

10.2.4 Combine the filtering methodology with existing approaches from the literature

As aforementioned in several parts of this thesis, there are several approaches to deal with large conceptual schemas in the literature. Even though our filtering approach provides more dynamism to the user and allows a request/response interaction that produces different results according to the specific user requirements, it should be interesting to combine the filtering methodology with clustering or summarization techniques.

One of the weakest points of our filtering methodology is the initial request a user needs to construct in order to start the iterative process of filtering. An inexperienced user needs to know which are the elements from the schema she wants to focus on. At this point, it could be difficult to the user to obtain those elements. To reduce the complexity, we plan to explore the effect of providing a clustered or summarized schema to the user in order to point out which are the main components or elements that cover or summarize the contents of a large conceptual schema, and therefore, help the user to obtain a first general view of the semantics in the schema.

10.2.5 Automatic refactor of schema rules after contextualization

Our filtering methodology provides a filtering request to focus on a set of entity and event types of focus, and then construct a contextualization function that modifies multiplicities of attributes and relationship ends. As an example, a user can set the multiplicity of one of these elements to 0..0, and therefore the filtering method will not show that element in the resulting filtered schema. We have identified a need of future work to automatically refactor schema rules after the application of a contextualization function that changes multiplicities of elements referenced by those rules. In that context, we plan to study ways to modify the OCL expressions that conform a schema rule taking into account the contextualization function, in order to make the rule semantically correct in the filtered schema.

10.3 Thesis Impact

The impact of this thesis is argumented in this section on the basis of these criteria: the scientific publications that are related to the thesis (see Sect. 10.3.1), and the degree final projects in which I have participated (see Sect. 10.3.2).

10.3.1 Publications

The results in this thesis are documented in the following publications:

General Filtering Method for Large Conceptual Schemas

A Method for Filtering Large Conceptual Schemas [128]
Antonio Villegas and Antoni Olivé
Conceptual Modeling - **ER** (2010), vol. **6412** of Lecture Notes in Computer Science, Springer, pp. 247–260.

Catalog of Specific Filtering Requests for Large Conceptual Schemas

Understanding Constraint Expressions in Large Conceptual Schemas by Automatic Filtering [130]
Antonio Villegas, Antoni Olivé, and Maria-Ribera Sancho
Conceptual Modeling - **ER** (2012), vol. **7532** of Lecture Notes in Computer Science, Springer, pp. 50–63.

Filtering Engine for Large Conceptual Schemas

A Tool for Filtering Large Conceptual Schemas [133]
Antonio Villegas, Maria-Ribera Sancho, and Antoni Olivé
Advances in Conceptual Modeling - Challenging Perspectives, **ER** Workshops (2011), vol. **6999** of Lecture Notes in Computer Science, Springer, pp. 353–356.

A Web-based Filtering Engine for Understanding Event Specifications in Large Conceptual Schemas [131]
Antonio Villegas, Antoni Olivé, and Maria-Ribera Sancho
Advances in Conceptual Modeling, **ER** Workshops (2012), vol. **7518** of Lecture Notes in Computer Science, Springer, pp. 383–386.

Relevance Metrics for Large Conceptual Schemas

On Computing the Importance of Entity Types in Large Conceptual Schemas [126]
Antonio Villegas and Antoni Olivé
Advances in Conceptual Modeling - Challenging Perspectives, **ER** Workshops (2009), vol. **5833** of Lecture Notes in Computer Science, Springer, pp. 22–32.

Extending the Methods for Computing the Importance of Entity Types in Large Conceptual Schemas [127]
Antonio Villegas and Antoni Olivé
Journal of Universal Computer Science, **J.UCS** (2010), vol. **16**, num. **20**, pp. 3138–3162.

On Computing the Importance of Associations in Large Conceptual Schemas [129]
Antonio Villegas, Antoni Olivé, and Maria-Ribera Sancho
Conceptual Modelling and Its Theoretical Foundations (2012), vol. **7260** of Lecture Notes in Computer Science, Springer, pp. 216–230.

Adaptation of the Filtering Methodology to HL7 V3 schemas

Improving the Usability of HL7 Information Models by Automatic Filtering [132]
Antonio Villegas, Antoni Olivé, and Josep Vilalta
6th IEEE World Congress on Services, **SERVICES** (2010), IEEE Computer Society, pp. 16–23.

10.3.2 Degree Final Projects

I have co-directed two Degree Final Projects that are closely related to the work in my PhD thesis:

Transformación de modelos del estándar de salud HL7 a UML/OCL — Transformation of standard healthcare models from HL7 to UML [91]
David Ortiz — *co-directed by* Antoni Olivé and Antonio Villegas
Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya.

Visualización de esquemas UML con HTML5 — Visualization of UML schemas with HTML5 [50]
Jose Maria Gomez — *co-directed by* Maria-Ribera Sancho and Antonio Villegas
Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya.

Bibliography

[1] Akoka, J., and Comyn-Wattiau, I. Entity-relationship and object-oriented model automatic clustering. *Data & Knowledge Engineering 20*, 2 (1996), 87–117.

[2] Apache Software Foundation. Apache axis. `http://axis.apache.org/`.

[3] Apache Software Foundation. Apache tomcat. `http://tomcat.apache.org/`.

[4] Atkinson, C., and Kuhne, T. Model-driven development: a metamodeling foundation. *IEEE software 20*, 5 (2003), 36–41.

[5] Baeza-Yates, R., Ribeiro-Neto, B., et al. *Modern information retrieval*. Addison-Wesley Harlow, England, 1999.

[6] Baroni, A. L. Formal definition of object-oriented design metrics. Master's thesis, Vrije Universiteit Brussel, 2002.

[7] Battista, G., Eades, P., Tamassia, R., and Tollis, I. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall, 1998.

[8] Bauerdick, H., Gogolla, M., and Gutsche, F. Detecting OCL traps in the UML 2.0 Superstructure: An experience report. In *UML 2004 – The Unified Modeling Language. Modelling Languages and Applications*, vol. 3273 of *Lecture Notes in Computer Science*. Springer, 2004, pp. 188–196.

[9] Beeler, G. HL7 Version 3–An object-oriented methodology for collaborative standards development. *International Journal of Medical Informatics 48*, 1-3 (1998), 151–161.

[10] Belkin, N. J., and Croft, W. B. Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM 35*, 12 (1992), 29–38.

[11] Bender, M. A., Farach-Colton, M., Pemmasani, G., Skiena, S., and Sumazin, P. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms 57*, 2 (2005), 75–94.

[12] Benson, T. *Principles of health interoperability – HL7 and SNOMED*. Springer Verlag, 2010.

[13] Berrabah, D., and Boufarès, F. Constraints checking in UML class diagrams: SQL vs OCL. In *DEXA'07*, vol. 4653 of *LNCS*. Springer Berlin, 2007, pp. 593–602.

[14] Bézivin, J., Jouault, F., Rosenthal, P., and Valduriez, P. Modeling in the Large and Modeling in the Small. *Model Driven Architecture* (2005), 33–46.

[15] Bézivin, J., Jouault, F., and Valduriez, P. On the need for megamodels. In *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications* (2004).

[16] Bidgood, W., Horii, S., Prior, F., and Van Syckle, D. Understanding and using DICOM, the data interchange standard for biomedical imaging. *Journal of the American Medical Informatics Association 4*, 3 (1997), 199.

[17] Blazona, B., and Koncar, M. HL7 and DICOM based integration of radiology departments with healthcare enterprise information systems. *International Journal of Medical Informatics 76* (2007), S425–S432.

[18] Blobel, B., Engel, K., and Pharow, P. Semantic interoperability–HL7 Version 3 compared to advanced architecture standards. *Methods of Information in Medicine 45*, 4 (2006), 343.

[19] Blobel, B., Engel, K., and Pharow, P. Semantic interoperability–HL7 Version 3 compared to advanced architecture standards. *Methods of Information in Medicine 45*, 4 (2006), 343–353.

[20] Boger, Z., Kuflik, T., Shoval, P., and Shapira, B. Automatic keyword identification by artificial neural networks compared to manual identification by users of filtering systems. *Information Processing & Management 37*, 2 (2001), 187–198.

[21] Brin, S., and Page, L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems 30*, 1-7 (1998), 107 – 117. Proceedings of the Seventh International World Wide Web Conference.

[22] Brun, C., and Pierantonio, A. Model differences in the eclipse modelling framework. *UPGRADE, The European Journal for the Informatics Professional* (2008).

[23] Cabot, J., Pau, R., and Raventós, R. From UML/OCL to SBVR specifications: A challenging transformation. *Information Systems 35*, 4 (2010), 417–440.

[24] Campbell, L., Halpin, T., and Proper, H. Conceptual schemas with abstractions making flat conceptual schemas more comprehensible. *Data & Knowledge Engineering 20*, 1 (1996), 39–85.

[25] Castano, S., De Antonellis, V., Fugini, M. G., and Pernici, B. Conceptual schema analysis: techniques and applications. *ACM Transactions on Database Systems 23*, 3 (1998), 286–333.

[26] Castro, J., Kolp, M., and Mylopoulos, J. Towards requirements-driven information systems engineering: the Tropos project. *Information Systems 27*, 6 (2002), 365–389.

[27] Cerami, E., and St Laurent, S. *Web services essentials*. O'Reilly & Associates, Inc., 2002.

[28] Ceusters, W., and Smith, B. Semantic interoperability in healthcare. *Transatlantic Observatory for Meeting Global Health Policy Challenges through ICT-Enabled Solutions. Argos eHealth Project (November 2010), Published by the EUROREC Institute,* `http://argos.eurorec.org` (2010).

[29] Chen, P. The entity-relationship model–toward a unified view of data. *ACM Transactions on Database Systems (TODS) 1*, 1 (1976), 9–36.

[30] Cockburn, A., Karlson, A., and Bederson, B. A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Surveys 41*, 1 (2008).

[31] Conesa, J., Storey, V. C., and Sugumaran, V. Usability of upper level ontologies: The case of researchcyc. *Data & Knowledge Engineering 69*, 4 (2010), 343–356.

[32] Costal, D., and Gómez, C. On the use of association redefinition in UML class diagrams. In *Conceptual Modeling - ER 2006, 25th International Conference on Conceptual Modeling* (2006), vol. 4215 of *Lecture Notes in Computer Science*, Springer, pp. 513–527.

[33] Cowan, N. Evolving conceptions of memory storage, selective attention, and their mutual constraints within the human information processing system. *Psychological Bulletin 104*, 2 (1988), 163–191.

[34] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet Computing 6*, 2 (2002), 86–93.

[35] Czarnecki, K., and Helsen, S. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture* (2003), pp. 1–17.

[36] DeRemer, F., and Kron, H. Programming-in-the large versus programming-in-the-small. In *Proceedings of the international conference on Reliable Software* (1975), ACM, pp. 114–121.

[37] Domínguez, E., Lloret, J., Rubio, A., and Zapata, M. Model-driven, view-based evolution of relational databases. In *DEXA'08*, vol. 5181 of *LNCS*. Springer Berlin, 2008, pp. 822–836.

[38] Eclipse Foundation. Eclipse project. `http://www.eclipse.org`.

[39] Eclipse Foundation. Eclipse web tools platform project. `http://www.eclipse.org/webtools/`.

[40] Egyed, A. Automated abstraction of class diagrams. *ACM Transactions on Software Engineering and Methodology 11*, 4 (2002), 449–491.

[41] Erl, T. *Service-oriented architecture: a field guide to integrating XML and web services.* Prentice Hall, 2004.

[42] Estivill-Castro, V. Why so many clustering algorithms: a position paper. *ACM SIGKDD Explorations Newsletter 4*, 1 (2002), 65–75.

[43] Feldman, P., and Miller, D. Entity model clustering: Structuring a data model by abstraction. *The Computer Journal 29*, 4 (1986), 348–360.

[44] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext transfer protocol — http/1.1, 1999.

[45] Forrey, A., McDonald, C., DeMoor, G., Huff, S., Leavelle, D., Leland, D., Fiers, T., Charles, L., Griffin, B., Stalling, F., Tullis, A., Hutchins, K., and Baenziger, J. Logical observation identifier names and codes (LOINC) database: a public use set of codes and names for electronic reporting of clinical laboratory test results. *Clinical Chemistry 42*, 1 (1996), 81–90.

[46] Francalanci, C., and Pernici, B. Abstraction levels for entity-relationship schemas. In *Entity-Relationship Approach - ER'94, Business Modelling and Re-Engineering, 13th International Conference on the Entity-Relationship Approach* (1994), vol. 881 of *Lecture Notes in Computer Science*, Springer, pp. 456–473.

[47] Frias, L., Queralt, A., and Olivé, A. EU-Rent car rentals specification. Tech. rep., Universitat Politècnica de Catalunya, `http://www.lsi.upc.edu/~techreps/files/R03-59.zip`, 2003.

[48] Geerts, F., Mannila, H., and Terzi, E. Relational link-based ranking. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases* (2004), VLDB Endowment, pp. 552–563.

[49] Gogolla, M., Büttner, F., and Richters, M. Use: A uml-based specification environment for validating uml and ocl. *Science of Computer Programming 69*, 1-3 (2007), 27–34.

[50] Gomez, J. M. Visualización de esquemas UML con HTML5. Tech. rep., Universitat Politècnica de Catalunya, 2012.

[51] Grimson, J., Grimson, W., and Hasselbring, W. The SI challenge in healthcare. *Communications of the ACM 43* (2000), 48–55.

[52] Halliday, D. *Fundamentals of Physics*. Wiley, 2010.

[53] Hammond, W. The status of healthcare standards in the United States. *International Journal of Bio-Medical Computing 39*, 1 (1995), 87–92.

[54] Han, J., and Kamber, M. *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.

[55] Hanani, U., Shapira, B., and Shoval, P. Information filtering: Overview of issues, research and systems. *User Modeling and User-Adapted Interaction 11*, 3 (2001), 203–259.

[56] Hartley, J. Case study research. *Essential guide to qualitative methods in organizational research* (2004), 323–333.

[57] Hay, D., and Healy, K. Defining business rules – what are they really? Tech. rep., Business Rules Group, `http://www.businessrulesgroup.org/first_paper/br01c0.htm`, 2000.

[58] Health Level Seven International. R-MIM diagram representation. http://wiki.hl7.org/index.php?title=RMIM_Diagram_Representation.

[59] Hevner, A., March, S., Park, J., and Ram, S. Design science in information systems research. *Mis Quarterly* (2004), 75–105.

[60] Hsi, I., Potts, C., and Moore, M. Ontological excavation: Unearthing the core concepts of the application. In *Proceedings of the 10th Working Conference on Reverse Engineering* (2003), WCRE'03, IEEE Computer Society, pp. 345–352.

[61] Jaeschke, P., Oberweis, A., and Stucky, W. Extending ER model clustering by relationship clustering. In *Entity-Relationship Approach - ER'93, 12th International Conference on the Entity-Relationship Approach* (1994), vol. 823 of *Lecture Notes in Computer Science*, Springer, pp. 451–462.

[62] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. ATL: A model transformation tool. *Science of Computer Programming 72*, 1-2 (2008), 31–39.

[63] Kalra, D., Lewalle, P., Rector, A., Rodrigues, J., Stroetmann, K., Surjan, G., Ustun, B., Virtanen, M., and Zanstra, P. Semantic interoperability for better health and safer healthcare. *Research and Deployment Roadmap for Europe. SemanticHEALTH Project Report (January 2009), Published by the European Commission, http://ec.europa.eu/information_society/ehealth* (2009).

[64] Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., and Giannopoulou, E. Ontology visualization methods–a survey. *ACM Computing Surveys 39*, 4 (2007).

[65] Kent, S. Model driven engineering. In *Integrated Formal Methods*, M. Butler, L. Petre, and K. Sere, Eds., vol. 2335 of *LNCS*. Springer Berlin, 2002, pp. 286–298.

[66] Kiencke, U., Majjad, R., and Kramer, S. Modeling and performance analysis of a hybrid driver model. *Control Engineering Practice 7*, 8 (1999), 985–991.

[67] Kleinberg, J. Authoritative sources in a hyperlinked environment. *Journal of the ACM 46*, 5 (1999), 604–632.

[68] Kleppe, A. G., Warmer, J., and Bast, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing, 2003.

[69] Kosara, R., Miksch, S., and Hauser, H. Focus+context taken literally. *IEEE Computer Graphics and Applications 22*, 1 (2002), 22–29.

[70] Kuflik, T., Boger, Z., and Shoval, P. Filtering search results using an optimal set of terms identified by an artificial neural network. *Information Processing & Management 42*, 2 (2006), 469–483.

[71] Kuflik, T., and Shoval, P. Generation of user profiles for information filtering – research agenda. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval* (2000), ACM, pp. 313–315.

[72] Lindland, O. I., Sindre, G., and Sølvberg, A. Understanding quality in conceptual modeling. *IEEE Software 11* (1994), 42–49.

[73] Maidel, V., Shoval, P., Shapira, B., and Taieb-Maimon, M. Evaluation of an ontology-content based filtering method for a personalized newspaper. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems* (2008), ACM, pp. 91–98.

[74] March, S., and Smith, G. Design and natural science research on information technology. *Decision support systems 15*, 4 (1995), 251–266.

[75] Marois, R., and Ivanoff, J. Capacity limits of information processing in the brain. *Trends in Cognitive Sciences 9*, 6 (2005), 296–305.

[76] Mellor, S. J., and Balcer, M. *Executable UML: A foundation for Model-Driven Architectures*. Addison-Wesley Longman, 2002.

[77] Miller, G. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychol. Rev 63* (1956), 81–97.

[78] Moody, D. L. A decomposition method for entity relationship models: A systems theoretic approach. In *ICSTM, International Conference on Systems Thinking in Management* (2000), vol. 72 of *CEUR Workshop Proceedings*, CEUR-WS.org.

[79] Moody, D. L. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering 35* (2009), 756–779.

[80] Moody, D. L., and Flitman, A. A methodology for clustering entity relationship models - a human information processing approach. In *Conceptual Modeling - ER 1999, 18th International Conference on Conceptual Modeling* (1999), vol. 1728 of *Lecture Notes in Computer Science*, Springer, pp. 114–130.

[81] Musial, B., and Jacobs, T. Application of focus + context to UML. In *APVis '03: Proceedings of the Asia-Pacific symposium on Information visualisation* (2003), Australian Computer Society, Inc., pp. 75–80.

[82] Mykkänen, J., and Tuomainen, M. An evaluation and selection framework for interoperability standards. *Information and Software Technology 50*, 3 (2008), 176–197.

[83] Nieto, P., Costal, D., and Gómez, C. Enhancing the semantics of UML association redefinition. *Data & Knowledge Engineering 70*, 2 (2011), 182–207.

[84] Object Management Group (OMG). *Unified Modeling Language (UML) Superstructure Specification, version 2.2*, February 2009.

[85] Object Management Group (OMG). *Object Constraint Language Specification (OCL), version 2.0*, February 2010.

[86] Oemig, F., and Blobel, B. Semantic interoperability adheres to proper models and code systems. *Methods of Information in Medicine 45*, 4 (2010), 343–353.

[87] Olivé, A. *Conceptual Modeling of Information Systems*. Springer, 2007.

[88] Olivé, A., and Cabot, J. A research agenda for conceptual schema-centric development. *Conceptual Modelling in Information Systems Engineering* (2007), 319–334.

[89] Olivé, A., and Raventós, R. Modeling events as entities in object-oriented conceptual modeling languages. *Data & Knowledge Engineering 58*, 3 (2006), 243–262.

[90] Open Health Tools. Model-Driven Health Tools (MDHT) Project. `https://www.projects.openhealthtools.org/sf/projects/mdht`.

[91] Ortiz, D. Transformación de modelos del estándar de salud HL7 a UML/OCL. Tech. rep., Universitat Politècnica de Catalunya, `http://hdl.handle.net/2099.1/12477`, 2011.

[92] Papazoglou, M. P. Unraveling the semantics of conceptual schemas. *Communications of the ACM 38*, 9 (1995), 80–94.

[93] Rahm, E., and Bernstein, P. A. A survey of approaches to automatic schema matching. *The VLDB Journal 10* (2001), 334–350.

[94] Ramirez, A. Esquema conceptual de Magento, un sistema de comerç electrònic. Tech. rep., Universitat Politècnica de Catalunya, `http://hdl.handle.net/2099.1/12294`, 2011.

[95] Richters, M., and Gogolla, M. OCL: Syntax, semantics, and tools. *Lecture Notes in Computer Science 2263* (2002), 42–68.

[96] Romero, J. jsUML2 - A lightweight HTML5/javascript library for UML 2 diagramming. Tech. rep., Universidad de Córdoba, `http://code.google.com/p/jsuml2/`, 2011.

[97] Salton, G. Automatic text processing. *Addison-Wesley Series In Computer Science* (1988), 450.

[98] Salton, G., and McGill, M. *Introduction to modern information retrieval*. McGraw-Hill, Inc. New York, NY, USA, 1986.

[99] Sandakith, L. *Eclipse WTP Tutorials – Creating Bottom Up Web Service via Apache Axis2*. Eclipse Foundation `http://wiki.eclipse.org/Creating_a_bottom-up_Axis2_Web_service`, June 2007.

[100] Schadow, G., Mead, C., and Walker, D. The HL7 Reference Information Model under scrutiny. *Studies in Health Technology and Informatics 124* (2006), 151–156.

[101] Selic, B. The pragmatics of model-driven development. *IEEE software 20*, 5 (2003), 19–25.

[102] Shapira, B., Hanani, U., Raveh, A., and Shoval, P. Information filtering: A new two-phase model using stereotypic user profiling. *Journal of Intelligent Information Systems 8*, 2 (1997), 155–165.

[103] Shapira, B., Shoval, P., and Hanani, U. Stereotypes in information filtering systems. *Information Processing & Management 33*, 3 (1997), 273–287.

[104] Shoval, P., Danoch, R., and Balaban, M. Hierarchical ER Diagrams (HERD)-The Method and Experimental Evaluation. In *Advanced Conceptual Modeling Techniques - ER Workshops* (2002), vol. 2503 of *Lecture Notes in Computer Science*, Springer, pp. 264–274.

[105] Shoval, P., Danoch, R., and Balaban, M. Hierarchical entity-relationship diagrams: the model, method of creation and experimental evaluation. *Requirements Engineering 9*, 4 (2004), 217–228.

[106] Sinclair, J., and Cardew-Hall, M. The folksonomy tag cloud: when is it useful? *Journal of Information Science 34*, 1 (2008), 15–29.

[107] Sotoodeh, M., Pottinger, R., and Kruchten, P. Towards supporting users in semantic exploration of large distributed schemas. *Procedia Computer Science 5*, 0 (2011), 570–577.

[108] Spahni, S., Lovis, C., Mercille, R., Verdel, H., Cotten, M., and Geissbühler, A. Implementing a new ADT based on the HL7 version 3 RIM. *International Journal of Medical Informatics 76*, 2-3 (2007), 190–194.

[109] Spronk, R. The HL7 MIF - Model Interchange Format. `http://www.ringholm.de/docs/03060_en_HL7_MIF.htm`, February 2010.

[110] Stearns, M., Price, C., Spackman, K., and Wang, A. SNOMED clinical terms: overview of the development process and project status. In *Proceedings of the AMIA Symposium* (2001), American Medical Informatics Association, p. 662.

[111] Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2008.

[112] Streit, A., Pham, B., and Brown, R. Visualization support for managing large business process specifications. *Lecture Notes in Computer Science 3649* (2005), 205.

[113] Sujansky, W. Heterogeneous database integration in biomedicine. *Journal of Biomedical Informatics 34*, 4 (2001), 285–298.

[114] Takeda, H., Veerkamp, P., and Yoshikawa, H. Modeling design process. *AI magazine 11*, 4 (1990), 37.

[115] Tavana, M., Joglekar, P., and Redmond, M. An automated entity–relationship clustering algorithm for conceptual database design. *Information Systems 32*, 5 (2007), 773–792.

[116] Teorey, T. J., Wei, G., Bolton, D. L., and Koenig, J. A. ER model clustering as an aid for user communication and documentation in database design. *Commununications of the ACM 32*, 8 (1989), 975–987.

[117] Thalheim, B. The science of conceptual modelling. In *DEXA'11*, vol. 6860 of *LNCS*. Springer Berlin, 2011, pp. 12–26.

[118] Tort, A., and Olivé, A. The osCommerce conceptual schema. Tech. rep., Universitat Politècnica de Catalunya, http://hdl.handle.net/2099.1/5301, 2007.

[119] Tzitzikas, Y., and Hainaut, J. On the visualization of large-sized ontologies. In *Proceedings of the working conference on Advanced Visual Interfaces* (2006), ACM, pp. 99–102.

[120] Tzitzikas, Y., and Hainaut, J.-L. How to tame a very large er diagram (using link analysis and force-directed drawing algorithms). In *Conceptual Modeling - ER 2005, 24th International Conference on Conceptual Modeling* (2005), vol. 3716 of *Lecture Notes in Computer Science*, Springer, pp. 144–159.

[121] Tzitzikas, Y., Kotzinos, D., and Theoharis, Y. On Ranking RDF Schema Elements (and its Application in Visualization). *Journal of Universal Computer Science 13*, 12 (2007), 1854–1880.

[122] Vaishnavi, V., and Kuechler, W. Design research in information systems. *Order A Journal On The Theory Of Ordered Sets And Its Applications 48*, 2 (2007), 133–140.

[123] Van Rijsbergen, C. Information Retrieval. *Cataloging & Classification Quarterly 22*, 3 (1996).

[124] Varga, R. *Matrix iterative analysis*. Springer, 2000.

[125] Villegas, A. Computing the importance of schema elements taking into account the whole schema. http://hdl.handle.net/2099.1/11296. Master's thesis, Univeritat Politècnica de Catalunya, 2009.

[126] Villegas, A., and Olivé, A. On computing the importance of entity types in large conceptual schemas. In *Advances in Conceptual Modeling - Challenging Perspectives, ER Workshops* (2009), vol. 5833 of *Lecture Notes in Computer Science*, Springer, pp. 22–32.

[127] Villegas, A., and Olivé, A. Extending the methods for computing the importance of entity types in large conceptual schemas. *Journal of Universal Computer Science 16*, 20 (2010), 3138–3162.

[128] Villegas, A., and Olivé, A. A method for filtering large conceptual schemas. In *Conceptual Modeling – ER 2010* (2010), vol. 6412 of *Lecture Notes in Computer Science*, Springer, pp. 247–260.

[129] Villegas, A., Olivé, A., and Sancho, M.-R. On computing the importance of associations in large conceptual schemas. In *Conceptual Modelling and Its Theoretical Foundations* (2012), vol. 7260 of *Lecture Notes in Computer Science*, Springer, pp. 216–230.

[130] Villegas, A., Olivé, A., and Sancho, M.-R. Understanding constraint expressions in large conceptual schemas by automatic filtering. In *Conceptual Modeling – ER 2012* (2012), vol. 7532 of *Lecture Notes in Computer Science*, Springer, pp. 50–63.

[131] Villegas, A., Olivé, A., and Sancho, M.-R. A web-based filtering engine for understanding event specifications in large conceptual schemas. In *Advances in Conceptual Modeling - Challenging Perspectives, ER Workshops* (2012), vol. 7518 of *Lecture Notes in Computer Science*, Springer, pp. 383–386.

[132] Villegas, A., Olivé, A., and Vilalta, J. Improving the usability of HL7 information models by automatic filtering. *6th IEEE World Congress on Services 0* (2010), 16–23.

[133] Villegas, A., Sancho, M.-R., and Olivé, A. A tool for filtering large conceptual schemas. In *Advances in Conceptual Modeling - Challenging Perspectives, ER Workshops* (2011), vol. 6999 of *Lecture Notes in Computer Science*, Springer, pp. 353–356.

[134] Warmer, J., and Kleppe, A. *The object constraint language: precise modeling with UML*. Addison-Wesley Longman Publishing, 1998.

[135] World Wide Web Consortium (W3C). *Web Services Description Language (WSDL) 1.1*, March 2001.

[136] World Wide Web Consortium (W3C). *Web Services Activity*, 2002.

[137] World Wide Web Consortium (W3C). *Simple Object Access Protocol (SOAP) Version 1.2 Part 1: Messaging Framework (Second Edition)*, April 2007.

[138] Xiao, R., Dillon, T., Chang, E., and Feng, L. Modeling and transformation of object-oriented conceptual models into XML schema. In *DEXA'01*, vol. 2113 of *LNCS*. Springer Berlin, 2001, pp. 795–804.

[139] Yang, X., Procopiuc, C. M., and Srivastava, D. Summarizing relational databases. In *VLDB 2009, 35th International Conference on Very Large Data Bases* (2009), pp. 634–645.

[140] Yu, C., and Jagadish, H. V. Schema summarization. In *VLDB 2006, 32nd International Conference on Very Large Data Bases* (2006), pp. 319–330.

[141] Yu, E. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering* (1997), pp. 226–235.

Index