A Methodological Approach to Policy Refinement in Policy-based Management Systems

Javier Rubio Loyola

Departament de Teoria del Senyal i Comunicacions

Universitat Politècnica de Catalunya

Thesis Director – Dr. Joan Serrat Fernández

A thesis submitted for the degree of

Doctor per la Universitat Politècnica de Catalunya

April 2007

A Yanet, por ser mi mejor amiga y compañera incondicional

A María y Juan por regalarme lo más lindo que puedo tener, mi familia

Mi gratitud para Sara y Joel, siempre

Acknowledgements

I would like to thank gratefully to Prof. Joan Serrat, who gave me the opportunity to complete this step of my professional life and who has given me guidance and support throughout this long and difficult process.

Thanks to Prof. George Pavlou for the support and to all the crew of University of Surrey. Thanks to Paris Flegkas and special thanks to Marinos Charalambides who gave me incredible support during the last steps of the realisation of this Thesis, thanks a lot.

To all my friends in Barcelona, Baja California and Mexico City, who have encouraged me to complete this Thesis.

Amb el suport del Departament d'Universitats, Recerca i Societat de la Informació, Catalonia, Spain.

Supported in part by the National Council for Science and Technology (CONACYT) of Mexico

Supported in part by the Public Education Secretary (SEP) of Mexico

Special thanks to the National Technical Institue (IPN) of Mexico

Abstract

Current research efforts are being directed to commit with the long-term view of self-management properties for telecommunications networks. One of the key approaches that have been recognised as an enabler of such a view is policy-based management. Policy-based management has been mostly acknowledged as a methodology that provides flexibility, adaptability and support to automatically assign network resources, control Quality of Service and security, by considering administratively specified rules. The hype of policy-based management was to commit with these features in run-time as a result of changeable network conditions resulting from the interactions of users, applications and existing resources. Despite enormous efforts with policy languages, management architectures using policy in different application domains, standardisation and industrial efforts, policy-based management is still not a reality. One reason behind the reticence for its use is the difficulty to analyse policies that guarantee configuration stability. In addition to policy conflict analysis, a key issue for this reticence is the need to derive enforceable policies from high-level administrative goals or from higher level policies, namely the policy refinement process.

This Thesis deals with the critical nature of addressing the policy refinement problem. We provide a holistic view of this process, from formal analysis to its practical realisation, identifying the key elements involved in each step of such critical process.

We initially propose a policy refinement framework relying on Linear Temporal Logic (LTL), a standard logic that allows analysis of reactive systems. Based on the former logic, we lay down the process of representing policies at different levels of abstraction. Following on with this, we develop the mechanisms that enable the abstraction of enforceable policies from hierarchical requirements in a fully automatic manner, making use of Linear Temporal Logic-based state exploration techniques. In addition, we clarify and identify the activities and management tasks that the administrative parties should carry out during the life cycle of the policy-based management system, from the perspective of the policy refinement process.

This Thesis provides the guidelines to address policy refinement in network management contexts. Concretely, we take one step ahead in the materialisation of the policy refinement process by exploiting inherent containment properties of network management systems. For this purpose we provide the methodology to apply the concepts introduced in the policy refinement framework developed in this Thesis in the above context.

In this Thesis we also execute a complete and rather detailed policy refinement process for a successful policy-based management solution. Taking the intra-domain Quality of Service Management application domain as background, we clarify and present the implications of the policy refinement problem in such a concrete application domain.

Resumen

En la actualidad se están realizando diversos esfuerzos para realizar la visión futurista de las redes de telecomunicación autogestionadas. La gestión basada en políticas ha sido reconocida como una herramienta potencial para habilitar esta visión. Mayoritariamente, ésta técnica ha sido reconocida como proveedora de flexibilidad, adaptabilidad y soporte para asignar recursos, controlar Calidad de Servicio y seguridad, de una manera automática y de acuerdo a reglas administrativas. Adicionalmente, se ha considerado que la gestión basada en políticas proveería tal flexibilidad en tiempo de ejecución y como resultado de cambios en la red, interacciones entre usuarios, aplicaciones y disponibilidad de recursos. A pesar de enormes esfuerzos realizados con lenguajes de especificación de políticas, arquitecturas de gestión en diversos dominios y estandarización, la gestión basada en políticas aún no es una realidad. Una de las razones para la reticencia en su utilización es la dificultad para analizar políticas que garanticen estabilidad en el sistema. Además de la problemática asociada a la gestión de conflictos entre políticas, otro obstáculo para su utilización es la dificultad de derivar políticas ejecutables alineadas a objetivos administrativos o a otras políticas de alto nivel. Este último es el problema del refinamiento de políticas.

Esta Tesis aborda el problema crítico de refinamiento de políticas. Damos una visión completa del proceso de refinamiento, desde el análisis formal hasta su realización práctica, identificando los elementos que intervienen en cada paso de tal proceso.

Inicialmente, proponemos un marco de trabajo para refinamiento de políticas basado en Lógica Lineal Temporal, una lógica estándar que permite el análisis en sistemas reactivos. Esta técnica es utilizada para representar políticas a diferentes niveles jerárquicos de abstracción. Acto seguido desarrollamos mecanismos que habilitan la obtención de políticas ejecutables a partir de ciertos requerimientos mediante la utilización de técnicas de exploración de estados basados en Lógica Lineal Temporal. Adicionalmente, aclaramos e identificamos las actividades y tareas de gestión de las partes administrativas durante el ciclo de vida de un sistema de gestión basado en políticas, desde la perspectiva del proceso de refinamiento de políticas.

Esta Tesis presenta también directrices para abordar el proceso de refinamiento de políticas en contextos de gestión de red. Damos un paso adelante en la materialización de este proceso mediante la utilización de propiedades estructurales inherentes a sistemas de gestión de red. Proveemos, en fin, una metodología para aplicar los conceptos introducidos en el marco de trabajo desarrollado en esta Tesis en sistemas de gestión de red.

En esta Tesis también realizamos un proceso de refinamiento de políticas completo. Detallamos la realización de tal proceso en una solución exitosa de gestión basada en políticas. Tomando como base el dominio de Gestión de Calidad de Servicio, aclaramos y presentamos las implicaciones del problema de refinamiento en este dominio de aplicación.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
AI	Artificial Intelligence
API	Application Programming Interface
BP	Behavioural Property
CORBA	Common Object Request Broker Architecture
CSL	Component Specification Language
DiffServ	Differentiated Services
DRsM	Dynamic Resource Management
DRtM	Dynamic Route Management
DSC	Distributed Software Component
DiffServ	Differentiated Services
ECA	Event-Condition-Action
FSM	Finite State Machine
GORE	Goal-Oriented Requirements Engineering
GUI	Graphical User Interface
IDE	Integrated Development Environment
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IST	Information Society Technologies
KAOS	Knowledge Acquisition in autOmated Specification
LDAP	Lightweight Directory Access Protocol
LTL	Linear Temporal Logic
LTS	Labelled Transition System
LSP	Label Switched Paths
MBM	Model Based Management
MPLS	Multi-Protocol Label Switching
MTNM	Multi-Technology Network Management
ND	Network Dimensioning
QoS	Quality of Service
PEP	Policy Enforcement Point
PDL	Policy Definition Language
PDP	Policy Decision Point
PHB	Per-Hop-Behaviour
PROMELA	Process Meta-Language
PS	Policy Server
QoS	Quality of Service
RAB	Resource Allocation Buffer
RAM	Resource Availability Matrix

RP	Refinement Pattern
RPC	Resource Provisioning Cycle
RPR	Request for Policy Refinement
SLA	Service Level Agreement
SLS	Service Level Specifications
SPIN	Simple Promela Interpreter
TE	Traffic Engineering
TEM	Traffic Estimation Matrix
TEQUILA	Traffic Engineering for Quality of Service for the Internet at Large Scale
TMN	Telecommunications Management Network
TP	Transition Plan
TR	Temporal Relationship
TT	Traffic Trunk
UML	Unified Modelling Language
WINMAN	WDM and IP Network Management

Chapter 1 Introduction

In the recent years, the research community has put efforts in new paradigms that allow installing, configuring, optimising and maintaining next generation networks. In management terms, these efforts are directed to provide self-management features to current telecommunication infrastructures [Ibm01], [Kep03]. Future communications will be self-configuring, self-optimising, self-healing, and self-protecting. Autonomous communications is the commonly term accepted for this view and is a long term view challenge still under research [Kep07].

Nowadays, the research community is concentrating in developing the technologies that will enable autonomicity to future systems. Future communication systems will have to adapt their behaviour according to changing performance conditions, and looking after service level agreements. In management terms, a policy is a directive that is administratively specified to manage certain aspects of desirable system behaviour resulting from the interactions of users, applications and existing resources [Ver00]. Policies are implemented with the philosophy, if "*event*" and "*conditions*" then "*actions*", which in turn prescribes that if certain conditions are present under the occurrence of a specific event, then specific actions must be taken in a policy-controlled environment.

Policy-based Management is meant to provide support to automatically assign network resources, control Quality of Service and security, by considering the previously defined set of policies. Also, it has been proposed to allow system scalability and adaptability to changeable network conditions and different QoS requirements of multimedia applications, virtual systems and other complex application processes that take place in real time. This management methodology has been acknowledged as one of the key enablers of self-healing properties, i.e. a key enabler of the future autonomic communications.

1.1 Policy-based Management Architecture

Policy-based Management involves policy creation, the translation of these policies into device specific configuration, and its application to enforce network behaviour according to the specified policies [Ver00]. Policy-based management has been used in different research projects and prototypes and consequently, we could say that there are several architectures tailored to address the peculiarities of each application domain in which policy has been used. Moreover, the background architecture mostly accepted by the research community is the architecture proposed by the Internet Engineering Task Force (IETF) [IETFPol], [IETFRap] which is graphically shown in Figure 1. A brief description of this architecture is provided thereafter.

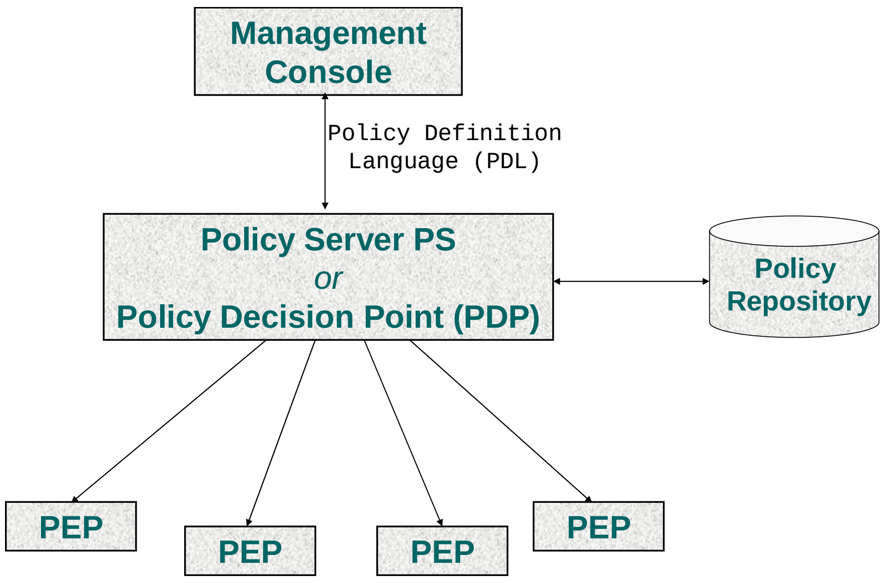

Management Console
Policy Definition Language (PDL)
Policy Server PS or Policy Decision Point (PDP)
Policy Repository
PEP
PEP
PEP
PEP

Figure 1. IETF policy architecture

The management of policies is executed through the Management Console. The latter acts as a user interface to allow constructing policies, deploying policies, and monitoring the status of the policy-managed environment. A Policy-based management system needs tools for policy specification. A Policy Definition Language (PDL) is used to define new policies in terms of policy rules with events, conditions and action lists. The language to use is very controversial, and the IETF [IETFPol], [IETFRap] has not reached consensus in standardising a Policy Definition Language.

The Policy Decision Point (PDP) is typically referred to as the Policy Server (PS). It is the entity that decides if the conditions of a policy are fulfilled and as a consequence, triggers the actions involved in that policy. Besides this function that is considered the main one, we can also attribute to this component the detection of policy conflicts, the retrieval of the relevant policy when required by an external trigger event and the interaction with the PEP component.

Policy Enforcement Points (PEPs) are basically network elements and are the entities that ensure that the actions ordered by the PDP are executed. It has also the role of metering and monitoring for auditing of policy compliance. The Policy Repository is used to store the policies with which the system works.

1.2 Policy Refinement Paradigm

The claim of policy-based management is that an ideal policy system might permit the definition of high-level policy description, enable their translation into lower-level ones and map them into commands that configure the managed devices properly. While the high-level policies would reflect the “business” criteria of the network administrator, the lower-level ones would mean to cope with device-level configuration.

Policy refinement is the process of transforming a high-level, abstract policy specification into low-level, concrete ones [Mof93]. The main objectives of this activity are:

- Determine the resources that are needed to satisfy the requirements of the policy
- Translate high-level policies into operational policies that the system can enforce
- Verify that the lower-level policies actually meet the requirements specified by the high-level policy.

1.3 Motivations of the Thesis

Despite having been introduced in different application domains through various research projects, several standardisation efforts and substantial interest from industry, policy-based management is still not widespread used. One reason behind the reticence for its use is the difficulty to analyse policies that guarantee configuration stability. Policies may have conflicts leading to unpredictable effects, and also, the number of policies necessary to control medium- to large-scale systems may be in the order of thousands. In this sense, in addition to policy conflict analysis, a key issue for the reticence to use Policy-based management is the need to derive policies from high-level administrative goals or from higher level policies, namely the policy refinement process.

Although policy refinement has been recognised as crucial for the success of policy-based management, it has been severely dismissed due probably to its inherent complexity. The main motivation of this Thesis is to identify and to address the key elements confronting the solution of the policy refinement process. We are moved by the current situation of the policy refinement process where for instance, it is still rather unclear how to address the refinement problem and the implications that it has on network management environments. To the time of the publication of this Thesis, there is not a clear understanding of the implications and future directions of the refinement problem, at least from what it is presented in the literature.

1.4 Contributions of the Thesis

In this Thesis we deal with the critical nature of addressing the policy refinement problem. We provide a holistic view of this process, namely from formal analysis to its practical realisation, identifying the key elements involved in each step of such critical process.

An initial contribution with this regard is a policy refinement framework relying on Linear Temporal Logic (LTL), a standard logic that allows analysis of reactive systems. Considering powerful analysis techniques from the Requirements Engineering area based on the former logic, we lay down the process of representing policies at different levels of abstraction. This way, the framework proposes the formalisation of high-level requirements and their translation into lower-level ones by means of LTL.

The Linear Temporal Logic foundations have made it possible to propose a framework that includes formal analysis techniques that enable the production of enforceable polices from formal hierarchical requirements in a fully automatic manner. Namely, the framework makes use of Linear Temporal Logic-based state exploration techniques to find restricted system behaviour that should commit to the previously formalised requirements.

Finally, the framework proposes ad-hoc machinery to abstract enforceable policies that should force the underlying system to behave the same way as the restricted system behaviour found in advance. In addition to laying down the above formalisms into a policy refinement framework, we clarify and identify the activities and management tasks that the administrative parties should carry out during the life cycle of the policy-based management system, from the perspective of the policy refinement process and the proposed framework.

Another contribution of the Thesis is a methodological approach to apply the policy refinement framework to network management contexts. The claim is that the techniques applied in the framework are mostly application-domain independent, and it is necessary to set up the guidelines to use them in network management contexts for practical use and to consider it a problem-solving approach. In addition, the techniques used in the framework are at some point novel for network management practitioners, and its utilisation may be unclear. With this regard, having defined a policy refinement framework, we provide general guidelines of its application to real-life management systems. The key contribution with this regard consists of a set of guidelines to drive the refinement process making use of the composition hierarchy of the management systems.

Finally, a substantial contribution of the Thesis is the execution of a complete refinement process in a successful policy-based management solution. To the best of our knowledge, no other work in the literature has provided a complete refinement scenario applied to real-life management situations and consequently it has been unclear at what extent the refinement problem is feasible in complex environments, or the overall implications for its assessment.

For this purpose we execute policy refinement for intra-domain Quality of Service Management, based on the principles developed in the context of the European IST project TEQUILA - Traffic Engineering for Quality of Service for the Internet at Large Scale [Tri01]. We make use of our framework and the methodological approach to address the policy refinement problem in this concrete application domain.

We must acknowledge that the policy refinement area has received very little attention from the research community. No other work in the literature has provided a holistic view of the refinement problem and consequently, we think that the ideas presented in this Thesis may encourage policy designers and researchers to address the policy refinement process in different application domains, We are not claiming to have solved the policy refinement problem because we think it is still at its initial stage and substantial efforts should be made to solve it.

1.5 Structure of the Thesis

This Thesis is composed by seven Chapters and three Appendixes.

Chapter 1 introduces the Thesis providing a general outline of policy-based management and the refinement problem. We also state the motivations of the Thesis and its main contributions to the state of the art.

Chapter 2 presents some background material that constitutes the foundations of the Thesis. Concretely, we present some crucial material on Requirements Engineering Techniques, namely the Goal-oriented Requirements Engineering foundations, methodology and practicality. Finally, we provide some technical background on Analysis of Reactive Systems techniques that include their objectives, analysis and tool support.

Chapter 3 describes one of the contributions of the Thesis, our policy refinement framework. We describe the rationale of the approach and present a functional prototype aligned to the former for proof of concept tests.

Chapter 4 presents another contribution of the Thesis. This Chapter provides the guidelines to address policy refinement in network management contexts. It provides a self-contained example that illustrates how to lay down the use of the policy refinement framework by making use of inherent features of hierarchical management systems.

Another contribution of the Thesis is provided in Chapter 5. This Chapter outlines a holistic policy refinement scenario. We make use of a Quality of Service Management solution and detail the refinement process as a whole.

Chapter 6 surveys the related work on policy refinement. We provide general analysis of goal-oriented management approaches, analysis techniques and functional prototypes targeting the refinement problem, and other efforts presented so far in the refinement area.

The Summary Conclusions of the Thesis are provided in Chapter 7. We review the contributions of the Thesis, provide additional discussion of relevant issues of the Thesis and provide some directions for future work.

Appendix A includes examples of PROMELA specifications exemplified throughout the body of the Thesis. Appendix B describes the Lucent Technologies Distributed Software Components toolkit used for the implementation of our functional prototype. Appendix C includes a detailed description of the main methods, classes and interfaces of our policy refinement prototype. Finally, Appendix D references the author’s publications related to this Thesis.

Chapter 2 Technological Background

2.1 Introduction

In this Chapter we provide some background material that constitutes the foundations of the work. Specifically, we present the Requirements Engineering Techniques and the Reactive Systems Analysis.

The formalization of the rationale to identify, organise and manage the capabilities of a system is of pivotal importance in our analysis. In the context of system design and operation, a lot of efforts have been directed to formalise the requirements a system should handle. For this reason, we outline the rationale of requirements engineering techniques and highlight the foundations and the practicality of this view of requirements formalisation.

Reactive systems have been traditionally represented by means of Finite State Machines (FSM) that are based on a strong formal support [Mea55]. The general "on-event and if-condition then action" structure of policy rules makes it possible to consider policy-based systems as reactive systems and hence use formal methods to analyze their behaviour. More importantly policy is represented as a means to control when a managed object transitions to a new state [Str04]

In the context of policy analysis, the representation of individual or several managed objects is possible by defining finite state machines (FSMs) that describe the multiple states in which such managed objects can be [Str04]. In this sense, it is possible to relate the behavior of an object or a set of objects to the value of one or more attributes that are used to characterize the states of the system. State transitions hence, are directly related to changes of attributes, which policies configure and control. The general "on-event and if-condition then action" structure of policy rules makes it possible to consider policy-based systems as reactive systems and hence use formal methods to analyze their behaviour. For this reason, we describe the objectives and the rationale of reactive systems analysis techniques.

2.2 Requirements Engineering Techniques

The requirements engineering area has been focused on formalising and documenting system requirements as the achievements of the envisioned systems. The system achievements have been traditionally been acknowledged as goals. For this, a lot of efforts have been made to develop the so-called Goal-Oriented Requirements Engineering (GORE) [Lam01] methods.

This Section provides some background material in the area of GORE methods. We initially provide general issues of these methodologies and then describe the scope of specific areas like goal modelling, goal specification and goal-based reasoning. Due to the importance of goal-based reasoning in this Thesis we give some details on the foundations of goal elaboration processes and their potential use. We finally describe the practicality of Goal-oriented methods, we give a general description of the facilities of Objectiver [Obj], a tool that provides support for GORE methods.

2.2.1 Scope of Goal-oriented Requirements Engineering

Goals capture the various objectives the system under consideration should achieve [Lam01]. Goals may be formulated at different levels of abstraction ranging from high-level concerns such as "*number of real-time service subscriptions maximised* " for a service management system, to lower-level technical concerns such as "*conservative thresholds set*" for a subscription control system.

The reasons for focusing on goals, found in the GORE literature [Lam01] are manifold, these include:

- Achieving requirements completeness; goals provide a precise criterion for sufficient completeness of a requirements specification.
- Explaining requirements to stakeholders; goals provide the rationale for requirements in a way similar to design goals in design process.
- Goal refinement provides a natural mechanism for structuring complex requirements documents for increased readability. A goal refinement tree provides traceability links from high-level strategic objectives to low-level technical requirements.
- During the goal elaboration process the requirements engineer is faced with many alternatives that provide some level of abstraction to validate choices or to suggest alternatives.
- Separating stable from volatile information. A requirement represents one way of achieving a specific goal, but there may be other ways of achieving the same goal. In this view, High-level Goals re-use is pivotal to formulate evolving requirements. This may result in systems sharing single goal models i.e. goal generalisations.
- Goals drive the identification of requirements to support them, they have been shown to be the driving force for systems requirements elaboration.

GORE methods are concerned with the use of goals for eliciting, elaborating, structuring, specifying, analysing, negotiating, documenting, and modifying requirements [Lam01]. Several GORE methods have been defined which give more attention to one or more of these aspects [Reg05] but in general, major efforts are devoted to the Goal Modeling, Goal Specification and Goal-based Reasoning areas.

2.2.2 Goal Modeling and Specification

Goals are modeled and specified to provide support to formal reasoning schemes during the requirements engineering process.

2.2.2.1 *Goal Modelling*

Goals are generally modelled by intrinsic features such as their *type*, *attributes*, and by their *links* to other goals and to other elements of a requirements model.

Regarding goal *types*, the literature [Lam01] has identified two main *types*:

- A classification considering functional and non-functional goals. Functional goals [Dar93] underlie services that the system is expected to deliver, e.g. satisfaction and information goals. Non-functional goals [Chu00] refer to expected system qualities such as security, safety, performance, usability, flexibility, customisability, interoperability, and so forth.
- A second classification considering the temporal behaviour prescription. This classification identifies four different types of goals: Achieve, Cease, Maintain and Avoid. While Achieve and Cease goals obey to system behaviours that require some target property to be eventually satisfied or denied respectively, Maintain and Avoid goals restrict behaviours in that they require some target property to be permanently satisfied or denied respectively.

Regarding *attributes of goals*, goals can be characterised by attributes such as the name and other specification. Also, priority is another important attribute that can be attached to goals. Qualitative values for this attribute allow mandatory or optional goals to be modelled with various degrees of optionality. Other goal attributes that have been proposed include goal utility and feasibility.

Regarding *goal links*, these are introduced to relate goals with each other and with other modelling elements. Such links form the basis for defining *goal structures*. Links between goals are aimed at capturing situations where goals support other goals. Directly borrowed from problem reduction methods in Artificial Intelligence, AND/OR graphs may be used to capture goal refinement links [Dar93]. The latter results in the formalisation of AND/OR goal graph structures that may be useful to represent alternative goal refinements, to identify potential conflicts of goals, and to prove the correctness of goal refinements.

GORE methods could be applied to any discipline that may take advantage of formalising requirements. In consequence, we could find Goal Modelling processes targeting the formalisation of systems of any kind. GORE projects have been undertaken in various industrial domains that include telecom, aerospace, automotive industry, press, pharmaceutics, health care, air traffic control, etc [Lam04]. In this sense, the precise types, attributes and links that may result in the elaboration of goal graph structures depends on the domain in which goals are modelled and consequently there is not a generic goal information model that captures the requirements for systems of any kind.

2.2.2.2 *Goal Specification*

The target of Goal Specification is to provide an alternative to formalise requirements amenable to Goal-based Reasoning during the requirements specification itself and for further management tasks like verification, validation, conflict management and so forth. For this reason, semi-formal specifications of goals can be attached to more formal notations that enable systematic Goal-based Reasoning.

Semi-formal specifications [Dar98] generally declare goals in terms of their *type*, *attribute*, and *links*. Such declarations may in general be provided alternatively using a textual or graphical syntax. For example, consider the need to specify the requirement that a passenger in a lifting system will be satisfied once the elevator is called and the passenger is transported to the destination. A textual semi-formal Goal Specification for the latter requirement may be integrated by three goals; "Passenger Satisfied", "Elevator Called" and "Passenger Transported". This basic Goal Specification may be completed by indicating that the "Passenger Satisfied" goal is the parent goal of the refinements "Elevator Called" and "Passenger Transported". A graphical syntax of a Goal Specification becomes a *goal graph structure*. The Figure 2 shows the goal graph structure of our basic Goal Specification. Obviously, a complete specification for a lifting system may be integrated by more requirements and consequently, the graphical syntax may result in a more complex goal graph structure than the one shown in the Figure 2.

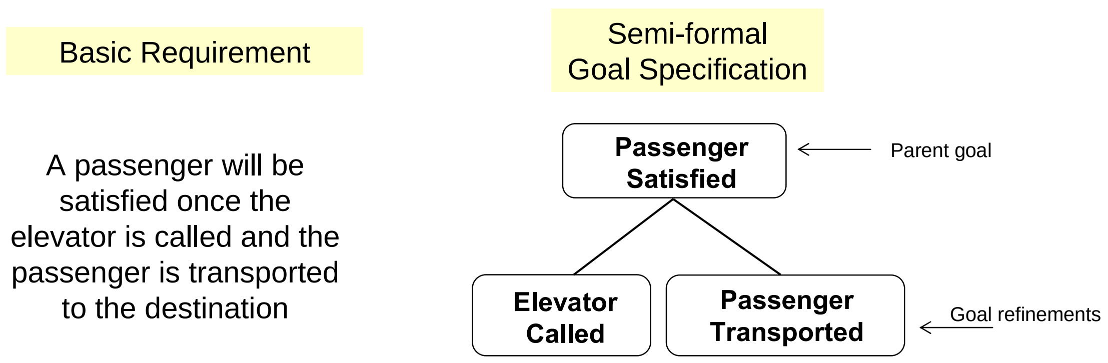

Basic Requirement
Semi-formal Goal Specification
A passenger will be satisfied once the elevator is called and the passenger is transported to the destination
Passenger Satisfied
Parent goal
Elevator Called
Passenger Transported
Goal refinements

Figure 2. Semi-formal Goal Specification for a Basic Requirement

In order to enable formal notations from semi-formal notations, goals may include keyword verbs with some predefined semantics or may be formalised with logical expressions. For example, a formal Goal Specification for the above requirement would be formalised by the following logical expression:

"Passenger Satisfied is true" IF "Elevator Called is fulfilled" AND "Passenger Transported is fulfilled"

The most relevant approach combining semi-formal and formal specifications of goals is the KAOS (Knowledge Acquisition in autOmated Specification) methodology [Dar96]. Due to the relevance of the KAOS methodology in our work the next sub-Section provides the principles of Goal-based Reasoning with the KAOS methodology.

2.2.3 KAOS Goal-based Reasoning

Goal-based reasoning techniques are applied to all system requirements activities: requirements elaboration, verification, validation, conflict management, negotiation, explanation and evolution. In terms of relevance, requirements elaboration plays a crucial role given that the remaining activities rely on it. KAOS is a methodology to implement goal-based reasoning. In particular, it supports requirements elaboration. In this context, requirements are elaborated as a two-step process. First of all, a goal graph structure (semiformal specification) is elaborated from which its formal notations are obtained making use of Linear Temporal Logic [Man92]. Therefore, we first provide some background material on Linear Temporal Logic (LTL) and then describe the principles for the elaboration of KAOS goal graph structures. We finally give a general outline of the potential use of KAOS goal graph structures.

2.2.3.1 Background on Linear Temporal Logic

The branch of logic that allows one to reason about causal and temporal relations of system properties is called temporal logic [Man92]. Temporal logic allows to formalise the properties of a system execution with the help of temporal operators. In this work we use the classical temporal operators: $\lozenge$ eventually in the future, $\square$ always in the future, U always in the future until and W always in the future unless. We also use the classical logic connectors $\wedge$ and, $\vee$ or, $\neg$ not, $\rightarrow$ logical implication, $\leftrightarrow$ equivalence, and so forth.

Linear Temporal Logic is used to express system properties where observations are extended with temporal connections such as “eventually in the future” or “always in the future”. The following are typical examples of frequently used LTL formulae describing system properties or temporal prescriptions:

- “$\lozenge p$” captures the notion that the system property p is guaranteed to eventually become true at least once throughout the system execution.
- “$\square p$” captures the notion that the system property p remains invariantly true throughout the system execution.
- “$p \rightarrow \lozenge q$” captures the notion that if the system property p holds at some point of the system execution, the system property q will eventually hold in the future of the system execution.
- “$\lozenge p \rightarrow \lozenge q$” captures the notion that if the system property p eventually holds, the system property q will eventually hold as well.

In conclusion, Linear Temporal Logic (LTL) enables to establish formal properties of systems and hence enables to carry out formal reasoning of systems features with respect to their temporal relationships. The next sub-section provides a view of the KAOS Support for Requirements Elaboration in terms of goals. For a complete summary of Linear Temporal Logic the interested reader may refer to [Man92] and for a complete summary of typical system properties formulae to reference [Dwy98].

2.2.3.2 KAOS Support for Requirements Elaboration

In general the KAOS methodology can be used upstream in the specification process as it supports formal reasoning about goals. It suggests ways of refining goals to make up correct goal graph structures at reasonable cost, as it hides proofs and their underlying mathematics [Dar96].

In KAOS, there is a two-level specification process; namely, semi-formal and formal. While the semi-formal approach copes with graphical and textual definitions, the formal approach is inspired in Linear Temporal Logic (LTL) [Man92]. In this view, goals are defined with a specific temporal prescription that can be represented with LTL logic formulae.

2.2.3.2.1 Basic definitions for goal refinement and refinement pattern

Definition 1: The goal assertions G1, G2,…,Gn are complete *goal refinements* of a goal assertion G iff the following conditions hold:

1. $G1 \wedge G2 \wedge \ldots \wedge Gn \models G$ (entailment)
2. $\forall\, i,j: j \neq i \rightarrow Gj \not\models Gi$ (minimality)
3. $G1 \wedge G2 \wedge \ldots \wedge Gn \not\models \text{false}$ (consistency)
4. $n > 1$ (nonequivalence)

The “entailment” condition refers to the fact that the intersection of the fulfillment of every goal refinement (G1, G2,…,Gn) should imply the fulfillment of the parent goal (G). The “minimality” condition refers to the fact that every goal refinement should be different in comparison with the rest. The “consistency” condition refers to the fact that the intersection of the temporal prescriptions of each goal refinement (G1, G2,…,Gn) should not conflict. Finally, the “nonequivalence” condition is set to avoid trivial refinements consisting in rewriting G into logically equivalent forms.

Definition 2: A refinement pattern is a one-level AND-tree of abstract goal assertions such that the set of leaf assertions is a complete refinement of the root assertion. In other words, a *refinement pattern* is a proposition of how a root goal assertion should be fulfilled by a complementary set of goal assertions. In the KAOS method a refinement pattern is propositional in the sense that it suggests the temporal prescription of the parent goal and that of its refinements. A refinement pattern in KAOS is applied at different levels during the formalisation of the goal graph structures, namely refinement patterns are applied to decompose a parent goal into goal refinement which in turn may be refined by applying a refinement pattern.

2.2.3.2.2 Domain independent goal refinement patterns

The basic idea of the KAOS methodology is to provide formal support for building goal graph structures by the use of domain-independent refinement patterns. For this purpose the methodology provides a set of domain independent refinement patterns [Dar96] that have been previously proved to be correct. The refinement patterns are grouped by the behaviour prescription of four High-level Goals, namely *Achieve, Cease, Maintain* and *Avoid*. For example, Table 1 shows some KAOS Refinement Patterns (RPs) that represent different ways to decompose the high-level *Achieve* parent goal into their respective sub-goals i.e. different goal refinements.

RP	Sub-goals		
RP1	$P \wedge R \rightarrow \Diamond Q$	$P \rightarrow \Diamond R$	$P \rightarrow P \, \mathcal{W} \, Q$
RP2	$P \rightarrow \Diamond R$	$R \rightarrow R \, \mathcal{U} \, Q$	
RP3	$P \rightarrow \Diamond R$	$R \rightarrow \Diamond Q$	
RP4	$P \wedge P1 \rightarrow \Diamond Q1$	$P \wedge P2 \rightarrow \Diamond Q2$	$\Box(P1 \vee P2)$ $Q1 \vee Q2 \rightarrow Q$
RP5	$P \wedge \neg R \rightarrow \Diamond R$	$P \wedge R \rightarrow \Diamond Q$	$P \rightarrow \Box P$
RP6	$\neg R \rightarrow \Diamond R$	$P \wedge R \rightarrow \Diamond Q$	$P \rightarrow \Box P$

Table 1 Some refinement patterns for the *Achieve* goal

The parent *Achieve* goal is formally expressed as $P \rightarrow \Diamond Q$. This expression states that *"If a property P occurs at some point, then property Q would eventually hold in the future"*. Here, the different refinement patterns provide some guidelines on how to decompose this parent goal. For example, let us establish a differentiation between the instantiations of RP3, RP4, RP5 and RP6:

- RP3 states that a parent goal $P \rightarrow \Diamond Q$ could be refined into sub-goals with the following temporal prescriptions: $P \rightarrow \Diamond R$, $R \rightarrow \Diamond Q$.
    - $P \rightarrow \Diamond R$ implies that *"under the occurrence of a state satisfying P, the state satisfying R must eventually be reached"*.
    - $R \rightarrow \Diamond Q$ implies that *"under the occurrence of a state satisfying R, the state satisfying Q must eventually be reached"*.

  The semantics of the intersection of the temporal prescriptions "$P \rightarrow \Diamond R$" and "$R \rightarrow \Diamond Q$" is as follows: "*on the occurrence of a state P, an intermediate state satisfying R must first be reached from which a goal state Q must eventually be reached"*. In other words, the instantiation of RP3 would formalise the fact that a *milestone* refinement tactic of fulfillment for the two sub-goals should be applied as to consider that our parent goal is properly fulfilled.

- RP4 states that a parent goal $P \rightarrow \Diamond Q$ could be refined into sub-goals with the following temporal prescriptions: $P \wedge P1 \rightarrow \Diamond Q1$, $P \wedge P2 \rightarrow \Diamond Q2$, $\Box(P1 \vee P2)$, and $Q1 \vee Q2 \rightarrow Q$.
    - $P \wedge P1 \rightarrow \Diamond Q1$ implies that *"on the occurrence of a state satisfying P and an alternative state satisfying P1, the state satisfying Q1 must eventually be reached"*.
    - $P \wedge P2 \rightarrow \Diamond Q2$ implies that *"on the occurrence of a state satisfying P and an alternative state satisfying P2, the state satisfying Q2 must eventually be reached"*.
    - $\Box(P1 \vee P2)$ implies that *"either a state P1 or a state P2 must always be satisfied"*
    - $Q1 \vee Q2 \rightarrow Q$ implies that *"either a state satisfying Q1 or a state satisfying Q2 implies that Q is reached"*

  The semantics of the intersection of the temporal prescriptions "$P \wedge P1 \rightarrow \Diamond Q1$", "$P \wedge P2 \rightarrow \Diamond Q2$", "$\Box(P1 \vee P2)$", and "$Q1 \vee Q2 \rightarrow Q$" is as follows: *"on the occurrence of P, an alternative sate satisfying either P1 or P2 will eventually satisfy the goal states Q1 or Q2 respectively which in turn suffices to satisfy the state Q"*. In other words, the instantiation of RP4 would formalise a *case-driven* refinement tactic of fulfillment in which either one sub-goal P1 or the other sub-goal P2 suffices to consider our parent goal as fulfilled.

- RP5 states that a parent goal $P \rightarrow \Diamond Q$ could be refined into sub-goals with the following temporal prescriptions: $P \wedge \neg R \rightarrow \Diamond R$, $P \wedge R \rightarrow \Diamond Q$, and $P \rightarrow \Box P$.
    - $P \wedge \neg R \rightarrow \Diamond R$ implies that *"on the occurrence of a state satisfying P, if a state satisfying R does not hold, the state R must eventually hold".*
    - $P \wedge R \rightarrow \Diamond Q$ implies that "*on the occurrence of a state satisfying P, if a state satisfying R holds, the goal state Q must eventually be reached".*
    - $P \rightarrow \Box P$ implies that *"the occurrence of a state satisfying P must always hold".*

  The semantics of the intersection of the temporal prescriptions "$P \wedge \neg R \rightarrow \Diamond R$", "$P \wedge R \rightarrow \Diamond Q$", and "$P \rightarrow \Box P$" is as follows: *"on the occurrence of P and the absence of the state R, the latter must hold together with P so that the goal state Q eventually holds".* In other words, RP5 requires that on the occurrence of state P, if the state R does not hold, the goal state Q will not be satisfied unless states P and R are both reached.

- RP6 states that a parent goal $P \rightarrow \Diamond Q$ could be refined into sub-goals with the following temporal prescriptions: $\neg R \rightarrow \Diamond R$, $P \wedge R \rightarrow \Diamond Q$, and $P \rightarrow \Box P$.
    - $\neg R \rightarrow \Diamond R$ implies that *"if a state satisfying R does not hold it must eventually hold".*
    - $P \wedge R \rightarrow \Diamond Q$ implies that "*on the occurrence of a state satisfying P, if a state satisfying R holds, the goal state Q must eventually be reached".*
    - $P \rightarrow \Box P$ implies that *"the occurrence of a state satisfying P must always hold".*

  The semantics of the intersection of the temporal prescriptions "$\neg R \rightarrow \Diamond R$", "$P \wedge R \rightarrow \Diamond Q$", and "$P \rightarrow \Box P$" is as follows: *"on the absence of a state satisfying R, the latter must hold when P occurs so that the goal state Q can be eventually reached".* In other words, RP6 proposes that it is mandatory that the property R holds by the time of the occurrence of P so that the goal state Q is satisfied.

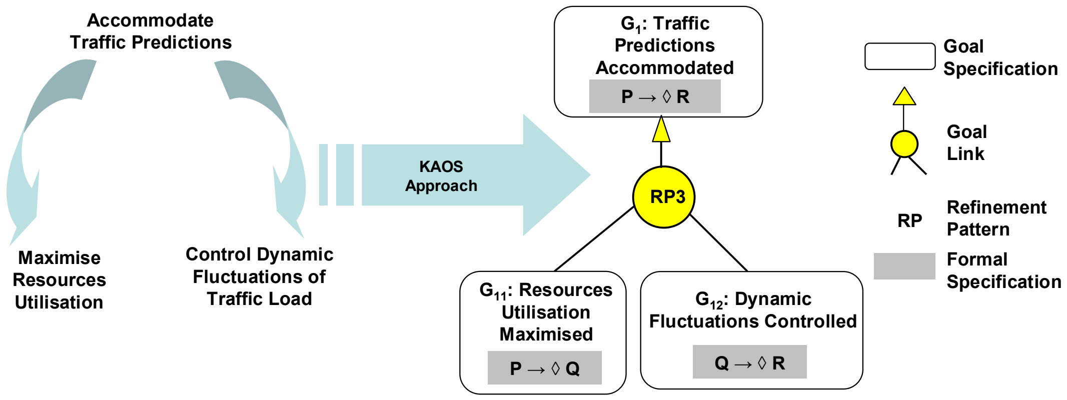
Any of these refinement patterns [Dar96] may be used in accordance with user requirements.

2.2.3.2.3 Using domain-independent refinement patterns

The KAOS methodology proposes the general principle of using the above domain-independent refinement patterns and instantiate them with appropriate domain-dependent information. In addition, all these patterns have been proved to be correct and complete. In this sense, KAOS provides the necessary support to build up hierarchical goal graph structures in which the lower-level sub-goals i.e. goal refinements, logically entail the parent goals. In this view the user would be enabled to build goal graph structures that are complete and correct without the need of carrying out logical proofs. The remaining of this section provides a brief example of this approach. A more extended description of this approach can be found in [Dar93], [Lam95].

Let us consider the case where a user defines the requirements for a system in charge of accommodating traffic load predictions to network resources. For the envisioned system the user considers two requirements: (1) Maximise the Resources Utilisation; and (2) Control Dynamic Fluctuations of Traffic Load. This basic scenario is shown in the left part of Figure 3. The intent is to formalise these requirements into a KAOS goal graph structure in which this information is modeled, specified and amenable for goal-based reasoning. In order to meet these requirements, Achieve prescriptions will be used to specify the goals in the graph structure. Let the administrator identify the parent goal "Traffic Predictions Accommodated" and link it to two sub-goals "Resources Utilisation Maximised" and "Dynamic Fluctuations Controlled". The administrator should identify then the appropriate refinement pattern that would formalise the aforementioned information into a goal graph structure. The refinement patterns shown in Table 1 are suggestions of how to refine an Achieve parent goal. Moreover the selection of one refinement pattern depends on administrative criteria.

Taking the patterns of Table 1 into account in this example, it is appropriate that the administrator selects the instantiation of RP3 since it considers that the parent goal "G_1: Traffic Predictions Accommodated" is refined into the sub-goal refinements "G_{11}: Resources Utilisation Maximised" and "G_{12}: Dynamic Fluctuations Controlled", making use of the aforementioned *milestone* refinement tactic. This formalises the goal graph structure of the right part of Figure 3, which includes the formal temporal logic formulae (shaded in Figure 3) for every goal of the graph structure. The meaning of all this goal graph structure would be "Traffic Predictions would be accommodated by maximising the resources utilisation and controlling dynamic fluctuations of traffic load".



Accommodate Traffic Predictions
Maximise Resources Utilisation
Control Dynamic Fluctuations of Traffic Load
KAOS Approach
G1: Traffic Predictions Accommodated
P → ◊ R
RP3
G11: Resources Utilisation Maximised
P → ◊ Q
G12: Dynamic Fluctuations Controlled
Q → ◊ R
Goal Specification
Goal Link
RP Refinement Pattern
Formal Specification

Figure 3. Goal Modeling, Specification and Goal-based Reasoning

At this stage the administrator has formalised the first level of a goal graph structure. Going further in the process and in a similar manner as for G_1, the administrator may formalise a second level of the goal graph structure by instantiating the appropriate patterns to refine G_{11} and G_{12}. A third level of the graph may be formalised by refining the resulting goal refinements of G_{11} and G_{12} as graphically illustrated in Figure 4.

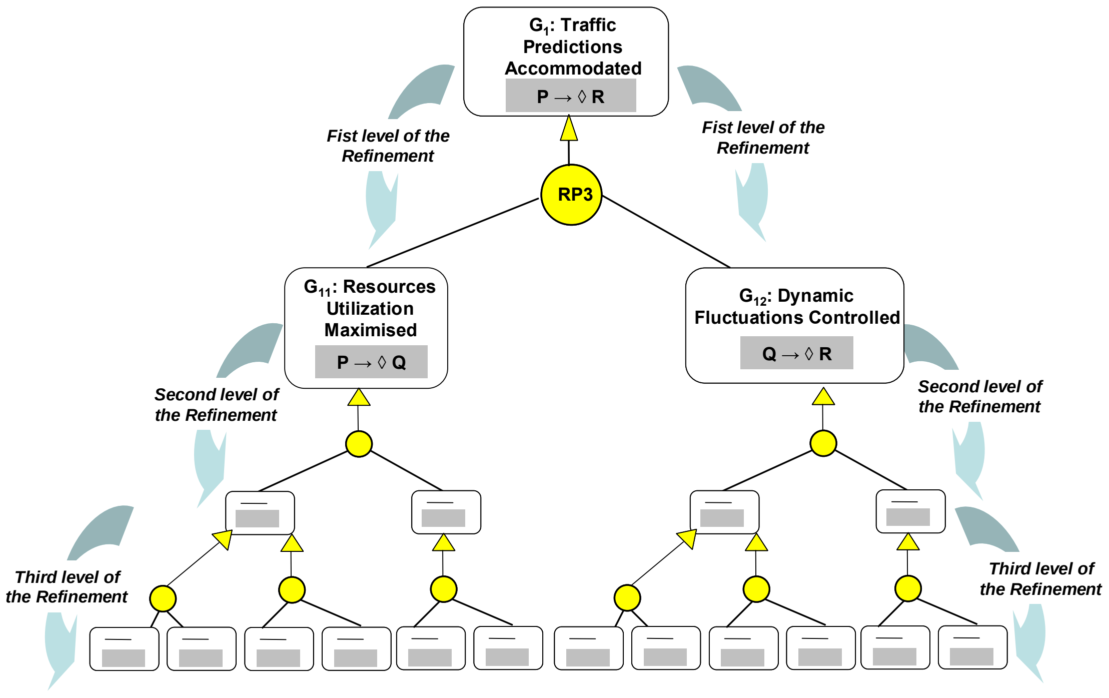

G_1: Traffic Predictions Accommodated
$P \rightarrow \Diamond R$
Fist level of the Refinement
Fist level of the Refinement
RP3
G_{11}: Resources Utilization Maximised
$P \rightarrow \Diamond Q$
G_{12}: Dynamic Fluctuations Controlled
$Q \rightarrow \Diamond R$
Second level of the Refinement
Second level of the Refinement
Third level of the Refinement
Third level of the Refinement

Figure 4. Levels of a goal graph structure

2.2.3.3 Potential Use of KAOS Goal-graph Structures in the Requirements Engineering area

The goal graph structures are elaborated until implementable constraints are reached, namely until the lowest-level goals are represented by actions. Once this is fulfilled, the next step is how to use the goal model. Goals are prescriptive statements of intentions whose satisfaction requires the cooperation of managed entities or active components. In this sense, goal graph structures are potential source for two key activities of the Requirements Engineering field [Lam01]: Responsibility Assignment and Operationalisation.

- The aim of the Responsibility Assignment activity is to explore the underlying system and define the precise responsibilities that each managed object should take as to fulfill the goal specifications. In addition to the underlying system specification or model, the input of this activity are the goal graph structures for which additional analysis is necessary to identify the managed entities, assign responsibilities to them, derive the managed entities' interfaces with other managed entities for mutual collaboration, and so forth. The analysis applied for the Responsibility Assignment is domain-dependent which means that there is not a unified method or procedure relying on a generic goal-based reasoning technique for this activity.
- The aim of the Operationalisation activity is to identify the precise actions that the managed entities of the underlying system should take as to fulfill the goal specifications. Same as for the Responsibility Assignment, the Operationalisation activity handles with the specification of the underlying system and the goal graph structures. The analysis carried out with this activity should evaluate the pre-conditions, the

triggering conditions and post-conditions of the operations from the goal specifications to make sure that the operations are complete and consistent with the operation of the underlying system.

In summary, the goal graph structures are potential source of information to apply goal-based reasoning techniques that allow defining the managed entities and the precise operations that such managed entities should take as to fulfill the requirements of a target system. For this purpose the applied techniques should handle the model of the target system and the goal graph structures.

In the context of our example the goal-based reasoning analysis for the Responsibility Assignment activity should target the deduction of managed entities in charge of accommodating traffic predictions. On the other hand, the analysis applied for the Operationalisation activity should cope with the identification of the precise operations that the aforementioned managed entities should take as to fulfill the goal specifications. The context of these two activities is graphically presented in Figure 5. A more detailed description of the Responsibility Assignment and Operationalisation activities can be found in [Lam01].

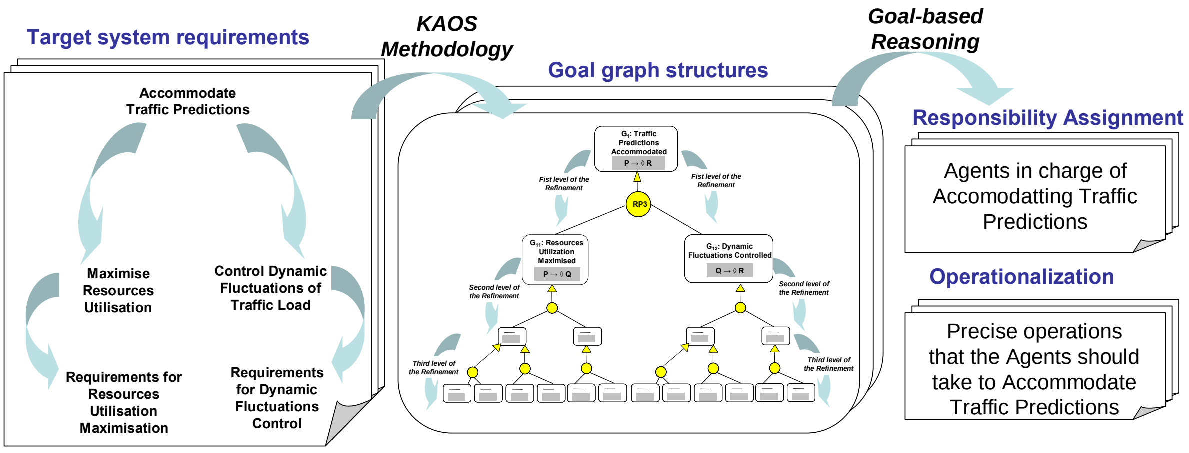
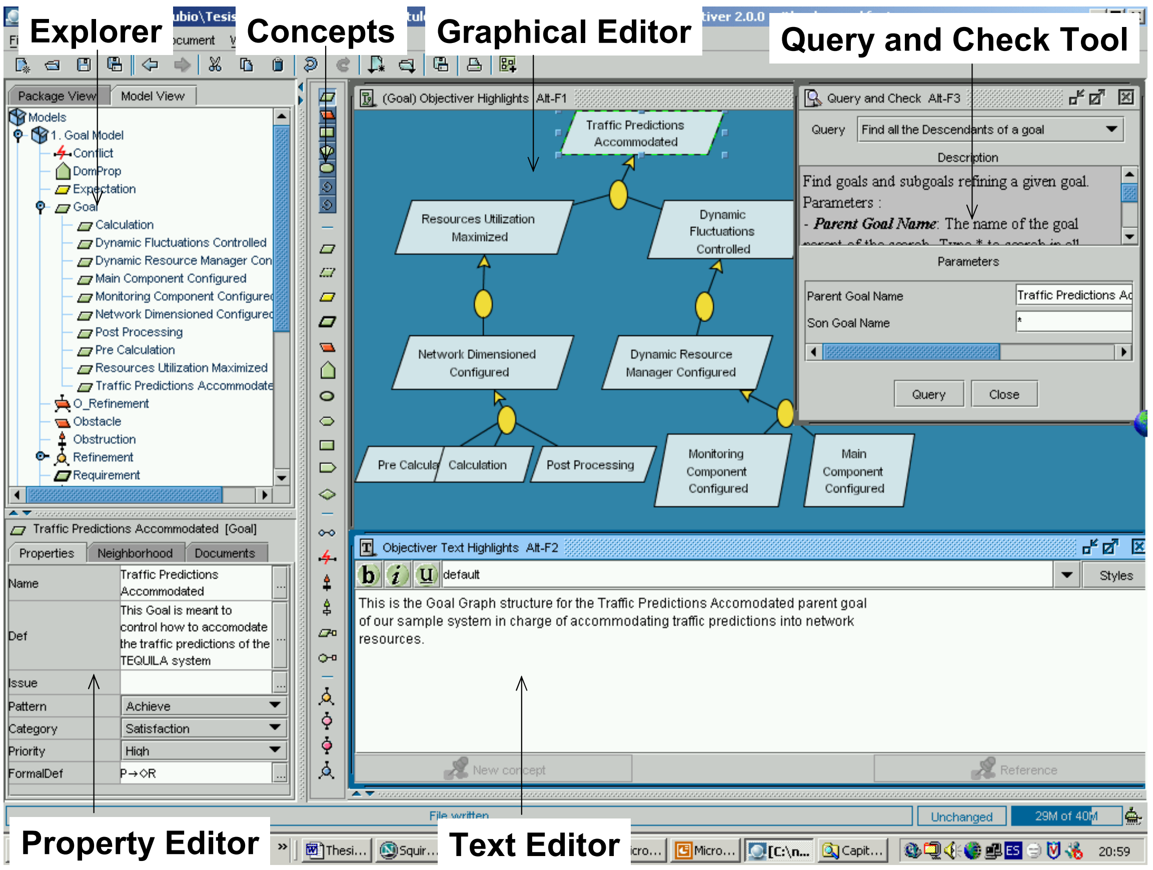

Target system requirements
Accommodate Traffic Predictions
Maximise Resources Utilisation
Control Dynamic Fluctuations of Traffic Load
Requirements for Resources Utilisation Maximisation
Requirements for Dynamic Fluctuations Control
KAOS Methodology
Goal graph structures
G1: Traffic Predictions Accommodated
P → ◊ R
Fist level of the Refinement
Fist level of the Refinement
RP3
G11: Resources Utilization Maximised
P → ◊ Q
G12: Dynamic Fluctuations Controlled
Q → ◊ R
Second level of the Refinement
Second level of the Refinement
Third level of the Refinement
Third level of the Refinement
Goal-based Reasoning
Responsibility Assignment
Agents in charge of Accomodatting Traffic Predictions
Operationalization
Precise operations that the Agents should take to Accommodate Traffic Predictions

Figure 5. Potential use of KAOS goal graph structures

2.2.4 Objectiver. A supporting tool for goal-oriented processes

The successful use of Goal-oriented Requirements Engineering techniques requires tool support that scale up to the size of large-scale systems throughout the specification and evolution of the requirements. Developing high-quality requirements specifications is mandatory for a number of critical industrial processes. The KAOS goal-driven methodology has been successfully implemented in Objectiver [Obj] and has been validated in many industrial projects [Del03].

Objectiver is a tool that supports the KAOS notation and has been specially tailored for the creation of requirements models and documents. It contains the same components found in most IDE (Integrated Development Environment) tools such as an explorer, a graphical editor, a query and check tool, a property editor and a text editor. Figure 6 is a snapshot of the the Objectiver environment.


Explorer
Concepts
Graphical Editor
Query and Check Tool
Property Editor
Text Editor
Traffic Predictions Accommodated
Resources Utilization Maximized
Dynamic Fluctuations Controlled
Network Dimensioned Configured
Dynamic Resource Manager Configured
Pre Calcula
Calculation
Post Processing
Monitoring Component Configured
Main Component Configured

Query: Find all the Descendants of a goal

Description: Find goals and subgoals refining a given goal. Parameters : - *Parent Goal Name*: The name of the goal

Parameters	
Parent Goal Name	Traffic Predictions Ac
Son Goal Name	*

This is the Goal Graph structure for the Traffic Predictions Accomodated parent goal of our sample system in charge of accommodating traffic predictions into network resources.

Traffic Predictions Accommodated [Goal]

Name	Traffic Predictions Accommodated
Def	This Goal is meant to control how to accomodate the traffic predictions of the TEQUILA system
Issue	
Pattern	Achieve
Category	Satisfaction
Priority	High
FormalDef	P→◇R

Figure 6. Tool support for Goal-oriented methods

The *Graphical Editor* is the space used by the stakeholder to represent the concepts (e.g. goals) and their relationships (e.g. link refinements). This is the space used to elaborate goal graph structures following the KAOS methodology. The *Text Editor* allows the developer to record design notes or to associate descriptive texts to diagrams. It allows foreign texts (such as interview transcripts or other source material) to be integrated, edited and hyperlinked to model elements. The *Property Editor* is used to specify predefined attribute values or user-defined attribute-value pairs of the concepts. For example, for a goal concept, these attributes include the name, the purpose of the goal, the refinement pattern used to decompose it, and the formal definition or temporal prescription of the goal. The *Explorer* is used to manage hierarchical views with drag-and-drop facilities. It is used to retrieve goal graph structures, text documents, and concepts by names, types or occurrences from the internal goal database. The *Query and Check Tool* is a cross-reference navigator that goes back and forth through all traceability and reference links between existing concepts and documents. It is used to carry out analysis on the goal graph structures by allowing queries about the details of the goal graph structures (e.g. find goals not refined, find concepts by name, find all the descendants of a goal, etc).

One of the major advantages of Objectiver is that foreign components can be tightly integrated through a meta-model based open Application Programming Interface (API). This makes Objectiver an excellent platform for structuring models in a user-readable way and exchanging these with the most powerful third parties formal tools. This capability makes it possible to develop extensions to Objectiver to allow formal analysis of the goal-oriented requirements.

One of the most prominent efforts in integrating formal tools in Objectiver is the FAUST toolbox [Pon05]. By the time of the publication of this thesis, the FAUST toolkit is still under development. The aim of FAUST is to allow formal modelling activity with user-oriented interfaces, easy understanding and that allows user validation.

2.3 Analysis of Reactive Systems

In this section we describe the objectives of reactive systems analysis techniques and the principles and usage of finite-state Behavioural Properties (BP). We finally provide some background on automated support for reactive systems analysis.

2.3.1 Objectives of Reactive Systems Analysis

Broadly speaking, reactive systems analysis is intended to verify that the system specification satisfies some properties and/or to acquire meaningful system behaviour. One of the most successful approaches targeting this issue is the Model Checking technique [Cla99]. Model Checking is a formal and automated application of computational logic with high relevance in concurrent and distributed systems verification. As shown in Figure 7, it consists of three main processes: modeling system behaviour, modeling the system requirements specification and the verification process.

Different formalisms have been proposed to model system behaviour, each tailored for specific domains. This is because each domain might involve different issues, like concurrency, distribution, object-orientation, etc. Amongst the most common formalisms, Labelled Transition Systems (LTS) are typically used. In general terms, a LTS is a set of states together with a transition set modeling how a system can change its state. In addition, a labelling function is used to relate states and transitions with observations. This activity has received tremendous attention in the modelling community and currently there are solutions that allow expressing LTSs by means of UML state charts, collaboration diagrams, sequence diagrams, class diagrams, etc [Bal04].

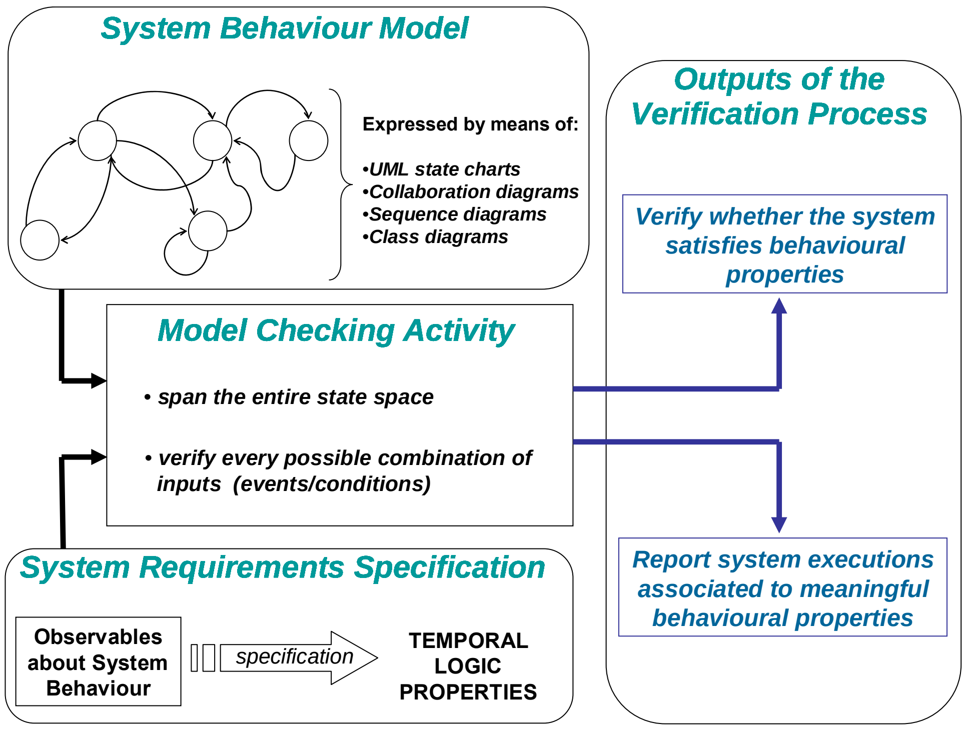

System Behaviour Model
Expressed by means of:
•UML state charts
•Collaboration diagrams
•Sequence diagrams
•Class diagrams
Outputs of the Verification Process
Verify whether the system satisfies behavioural properties
Model Checking Activity
• span the entire state space
• verify every possible combination of inputs (events/conditions)
Report system executions associated to meaningful behavioural properties
System Requirements Specification
Observables about System Behaviour
specification
TEMPORAL LOGIC PROPERTIES

Figure 7. General outline of Model Checking techniques

The second process of Model Checking corresponds to the modeling of requirements specifications. At this stage, system observables like events, value of variables or the processes responsible of the transitions are crucial when one wants to specify the requirements specifications. In this sense, a fundamental dimension is time and how observables are time-related. In many cases, an explicit treatment of real time is not required and it just suffices to have a mechanism that allows one to express the ordering of events in time. This is precisely the aim of Temporal Logics [Man92]. The requirements of a system then should be acknowledged as the means to specify the ongoing behaviour of an event/state-based system for which temporal logics provide an expressive and natural language for specifying this behaviour.

The third process of Model Checking is the verification of the properties. It attempts to span the entire state space and verify every possible combination of inputs (events and conditions). The success of the Model Checking technique relies in two main abilities; on one hand, its ability to verify that a system satisfies some Behavioural Properties (BP); and on the other hand its ability to find and report system executions associated to meaningful Behavioural Properties. The system executions are reported as traces that show the conditions, events/states and the managed objects involved in such conditions and events/states. This in fact is the ability that has made Model Checking so successful for reactive system analysis. Due to the relevance of the specification of Behavioural Properties in our analysis, the following section clarifies the nature of finite-state properties for Model Checking usage.

2.3.2 Specialisation of Behavioural Properties

In terms of finite-state system verification, a specification pattern is a generalised description of a commonly occurring requirement on state/event sequences in a system execution [Dwy98]. In order to use these patterns in Model Checking tools, they should be expressed as formal specifications. In this sub-Section we provide some background on the nature of patterns to specify Behavioural Properties (BP) and describe an approach to represent these into formal logic specifications suitable to use with Model Checking techniques.

2.3.2.1 Patterns for Behavioural Properties specification

Figure 8 shows a classification of design patterns for the specification of Behavioural Properties [Dwy98]. This classification takes into account two major groups: Occurrence and Order.

Occurrence patterns are used to identify behaviours in which a specific state/event takes place. Occurrence patterns are in turn classified in *Existence, Absence, Universality* and *Bounded Existence*. Existence patterns deal with situations in which the most important is to specify that a state/event occurs. Absence patterns instead deal with situations in which it is necessary to specify that a state/event does not occur. Universality patterns are meant to specify situations of occurrence of states/events throughout a scope. Finally, Bounded Existence patterns are used to specify the situations of occurrence of states/events for a number of *k* times within a scope.

Order patterns deal with prescribed behavioural arrangement of states/events in time sequence. These are classified in *Response* and *Precedence* patterns; Response patterns are used to represent constraints in the order of states/events. Precedence patterns are concerned with the specification of a given state/event *P* to be always preceded by a state/event *Q* within a scope.

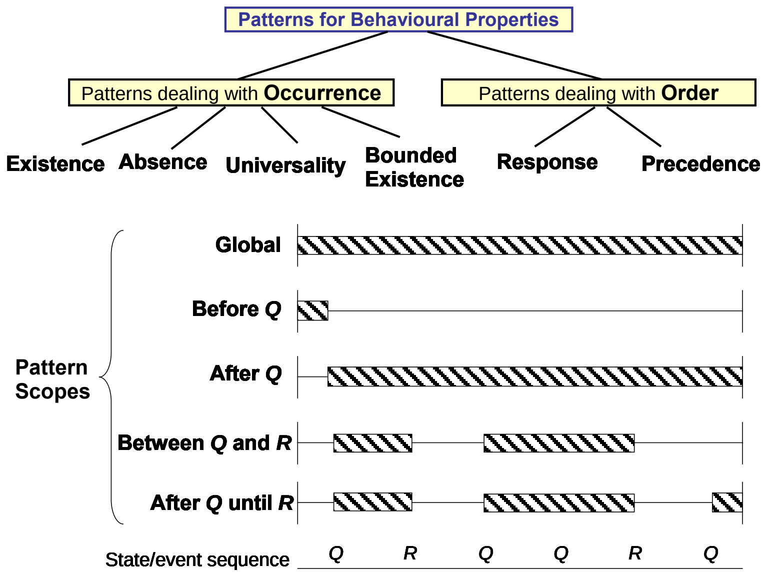


Figure 8. Behavioural pattern classification and scopes

The patterns for Behavioural Properties (BP) have a scope – the extent of system execution over which the pattern must hold. The lower part of Fig. 8 shows a graphical representation of the five possible scopes: *Global*, *Before*, *After*, *Between* and *After-until.* The scope Global prescribes that a pattern holds for the entire system execution; A Before scope is used to indicate that a pattern holds throughout the execution of the system up to a given state/event (state/event Q in Figure 8). The scope After is used in situations where the pattern must hold after a given state/event (state/event Q in Figure 8) and throughout the execution of the system. The Between scope helps to define behavioural situations in which a pattern must hold at any part of the execution of the system from one given state/event (state/event Q in Figure 8) to another state/event (state/event R in Figure 8). Finally the scope After-until is like Between with the difference that in the former the designated part of the execution continues even if the second state/event does not occur (state/event R in Figure 8).

The combination of patterns and scopes provides the sufficient support to express any kind of behaviour description. Broadly speaking, we can say that any behavioural system requirement can be mapped to their respective pattern/scope. For example, a combination of the Existence pattern with the Global scope would be used to specify the "occurrence of an event/state during the entire execution of the system". In other words, this combination would represent that a state/event is kept throughout all the lifecycle of the system. The other way around, any behaviour description can be mapped to their corresponding pattern/scope; for example two concrete examples would be the following requirements R_1 and R_2:

R_1 - "state/event B is reached After state/event A"
R_2 - "state/event B is not reached After state/event A" (R_2 is the opposite of R_1)

The requirement R_1 corresponds to an *Existence*-based property pattern since the statement demands the condition "state/event B is reached" to occur. Additionally, R_1 clearly prescribes the After scope. The requirement R_2 instead corresponds to an *Absence*-based pattern since the statement demands the non-occurrence of a condition "state/event A is not reached". This requirement also prescribes the After scope.

2.3.2.2 Formal specification of Behavioural Properties

The requirements for reactive systems and their representation into different temporal logics have been the subject of research for some time. This has been influenced by the high relevance of representing behavioural requirements with formal specifications suitable for use with reactive systems analysis like Model Checking, theorem proving, etc. The research community has identified several potential combinations of pattern/scopes of Behavioural Properties and their representation into different logics. Current approaches have been proposed to classify this information in databases and hence to relate formal representations with behavioural requirements in a systematic manner [Dwy98].

Consider the selection of Behavioural Properties shown in Table 2. Due to the relevance of our study we concentrate on Linear Temporal Logic (LTL) representations. As in previous sections we use the classical temporal operators: $\Diamond$ eventually in the future, $\Box$ always in the future, and the classical logic connectors $\wedge$ and, $\vee$ or, $\neg$ not, $\rightarrow$ logical implication, $\leftrightarrow$ equivalence, and so forth. A more detailed description of patterns for the specification of Behavioural Properties and practical database systems can be found in [Dwy] and [Dwy98].

Id	Pattern	Scope	LTL representation
BP1	Absence	Globally	$\Box(\neg P)$
BP2	Absence	After Q	$\Box(Q\rightarrow\Box(\neg P))$
BP3	Universality	After Q	$\Box(Q\rightarrow\Box(P))$
BP4	Response	Globally	$\Box(P\rightarrow \Diamond S)$
BP5	Response	After Q	$\Box(Q\rightarrow\Box(P\rightarrow\Diamond S))$

Table 2 Selection of behavioural pattern/scope database entries

- The LTL representation of BP1 "$\Box(\neg P)$" is as follows: *"property P is always absent"*. BP1 is used to express situations in which a property is globally absent
- The LTL representation of BP2 "$\Box(Q\rightarrow\Box(\neg P))$" is as follows: *"It always holds that after the occurrence of Q, the property P never holds"*. BP2 is used to express situations in which a property P is always absent after a property Q.
- The LTL representation of BP3 "$\Box(Q\rightarrow\Box(P))$" is as follows: *"It always holds that on the occurrence of Q, the property P will always occur"*. BP3 is used to express situations in which a property P always holds after a property Q.
- The LTL representation of BP4 "$\Box(P\rightarrow \Diamond S)$" is as follows: *"It always holds that on the occurrence of Q, the property P eventually occurs"*. BP4 is used to express situations in which the occurrence of a property P always causes the eventual occurrence of property S.
- The LTL representation of BP5 "$\Box(Q\rightarrow\Box(P\rightarrow\Diamond S))$" is as follows: *"The occurrence of Q always implies that the occurrence of P always causes the eventual occurrence of S"*. BP5 is used to express situations in which a property S responds to P after the occurrence of a property Q.

We have described a reduced number of behavioural properties (BP). The above is illustrative given that we can express an unlimited number of combinations of Pattern and Scopes. Moreover the most relevant of the above analysis is the possibility to express behavioural properties in LTL representations following a pre-established identification of Patterns and their application Scopes. For example, consider the requirements Rq_1 and Rq_2:
Rq_1 - "state/event B is always reached After state/event A"
Rq_2 - "state/event B is never reached After state/event A"

- For Rq_1 we could consider that the state/event B always holds after the occurrence of state/event A, and then instantiate BP3 to represent Rq_1 in LTL notations. In this view, the LTL representation of Rq_1 is as follows: $\Box$(state/event A$\rightarrow\Box$(state/eventB))

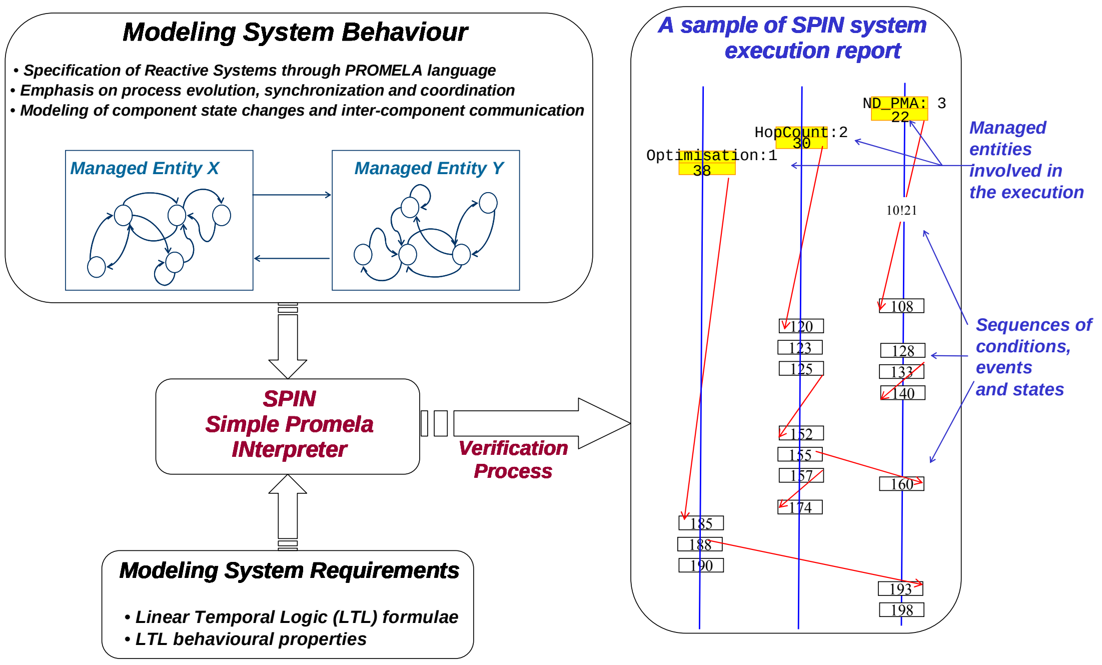
- For Rq_2 we could consider that state/event B is always absent after the occurrence of event/state A, and then instantiate BP2 to represent Rq_2 in LTL notations. In this view, Rq2 is represented as follows in LTL: $\Box$(state/event A$\rightarrow\Box$($\neg$state/event B)).

Following this approach we may use combinations of patterns/scopes to formulate requirements that specify "particular" behavioural aspects of system executions. Moreover, this approach enables to represent those requirements into formal specifications suitable for use with automated verification tools applying Model Checking techniques.

2.3.3 Automated Support for Reactive Systems Analysis

One of the main advantages of analysing reactive systems using Model Checking techniques is that there are several off-the-shelf tools available [Cla99]. Tools like SPIN (Simple PROMELA Interpreter) [Hol04] and NuSMV [Cim99] have enabled systematic analysis for different applications such as the verification of large-scale software specifications [Cha98], use of Model Checking as planning [Cim03], generation of test-cases through Model Checking [Amm99] and others. Due to the high relevance of linear temporal analysis in this work, we provide the general outline of SPIN [Hol04], a tool that provides support for linear temporal analysis.

As tool applying Linear Temporal Model Checking techniques, SPIN concerns with the three main activities: Modeling System Behaviour, Modeling System Requirements and the Verification Process. The verification process can consist of two options; (1) verify that a system satisfies some behavioural properties; and (2) find/report system executions associated to meaningful behavioural properties. The Figure 9 shows a graphical representation of these three activities for which a brief rationale is given hereafter.



Modeling System Behaviour
• Specification of Reactive Systems through PROMELA language
• Emphasis on process evolution, synchronization and coordination
• Modeling of component state changes and inter-component communication
Managed Entity X
Managed Entity Y
SPIN Simple Promela INterpreter
Verification Process
Modeling System Requirements
• Linear Temporal Logic (LTL) formulae
• LTL behavioural properties
A sample of SPIN system execution report
ND_PMA: 3
22
HopCount:2
30
Optimisation:1
38
10!21
108
120
123
125
128
133
140
152
155
157
160
174
185
188
190
193
198
Managed entities involved in the execution
Sequences of conditions, events and states

Figure 9. General Outline of SPIN support

2.3.3.1 Modeling System Behaviour: an Overview of PROMELA

For this fist activity SPIN uses PROMELA as specification language [Hol04]. PROMELA stands for Process Meta-Language and is not considered as an implementation language like Java or C but as a system description language. The emphasis on the language is on the modeling of process evolution, synchronisation and coordination, and not on computation. For example, it would be relatively hard to model the computation of, say, a square root but relatively easy to model the behaviour of clients and servers in distributed systems. This is deliberately designed to encourage the user to abstract from the purely computational aspects of a design, and to focus on the specification of processes evolution and interaction at system level. A PROMELA model represents an abstraction of design that contains only those aspects of a system which are relevant to the features to be modelled. For example, PROMELA emphasises on modeling how a managed entity changes from one state to other, or how a managed entity receives or sends information to other managed entity. In conclusion PROMELA has been designed to model processes that would be implemented by managed entities and hence to model the behaviour of such managed entities in collaborative environments. For this, the language includes a rich set of primitives for process modelling and inter-process communication to exchange information between processes.

2.3.3.1.1 An illustrative collaborative scenario

Consider two collaborative entities, Managed Object 1 (MO1) and Managed Object 2 (MO2) exhibiting the behaviour illustrated in Figure 10. In this basic example, the Managed Object 2 issues three events (event_1, event_2, and event_3) which in turn have some effect in the state transitions of Managed Object 1. Note for example that when MO2 transitions from state "MO2_State1" to state "MO2_State4" (occurrence of event_1 in MO2), MO1 is obligated to transition from state "MO1_state1" to state "MO1_State4". Similar effects are caused when MO2 issues the events event_2 and event_3: the MO transitions from MO1_State2 to MO1_State4 and from MO1_State3 to MO1_State4 respectively.

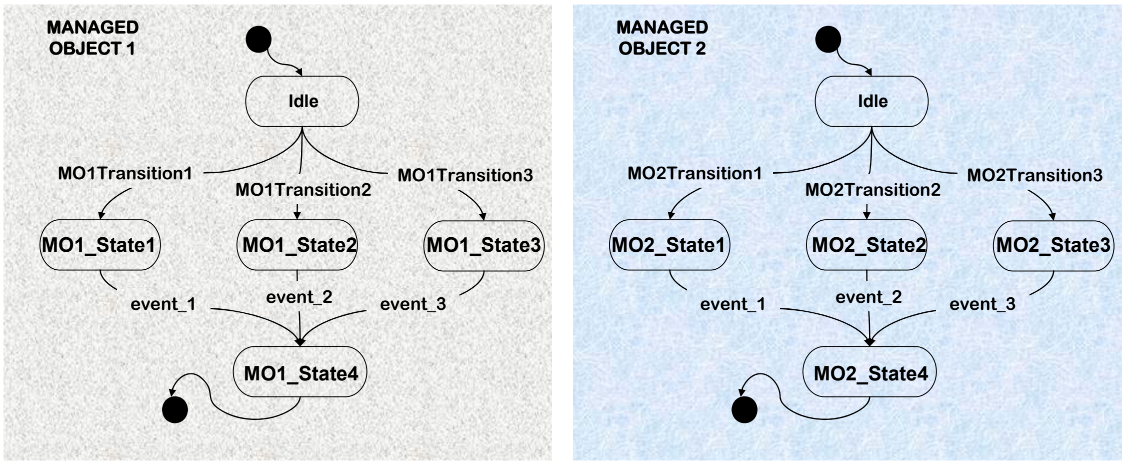


Figure 10. Example of collaborative behaviour

2.3.3.1.2 The Process element in PROMELA

PROMELA derives many of its notational conventions from the C programming language. In PROMELA, processes are used to define the body of the managed entities' behaviour. They are identified by `proctype` instantiations. There must be at least one `proctype` declaration in a model. Since we are modelling the behaviour of two managed objects, the specification of our example integrates two `proctype` instantiations: `proctype ManagedObject2` and `proctype ManagedObject1`. The Figure 11 shows the body of the `proctype ManagedObject1` declaration of our example.

Process declarations are followed by typical declarations of variables. Besides the typical declarations like `byte`, `bool`, `int`, etc, PROMELA introduces the concept of communication channel. The purpose of communications channels is to handle relevant information that causes state transitions in a state machine and amongst state machines. Communication channels are identified with the keyword `chan`. In our model example we have used two types of communications channels: A channel for internal communications declared as `chan internal_queue` and channels for communications amongst processes declared as `chan ack_in` and `chan ack_out`.

Since PROMELA emphasises on modeling how a managed entity changes states from one state to other, the actual state of every managed object are modeled with values of variables. In this basic example we have considered the variable `state0`. Note for instance that this variable takes different values that represent the evolution of the managed entities (MO1 and MO2). In this basic model, state transitions are modelled in two generic groups: *Transitions without relevant information* and *Transitions with relevant information*.

The *Transitions without relevant information* are characterised by the absence of information linked to state transitions. In our basic model these correspond to the transitions from Initial state to Idle and the transitions to the Final state of MO1 and MO2. Even in these circumstances the value of the `state0` variable changes as illustrated by the pointer "Transitions without relevant information" in Figure 11.

On the other hand, the *Transitions with relevant information* take into account the events/actions that cause the state transition like MO1Transition1 and event_1 in our running example. These transitions should consider the information of the channels. The pointer "Transitions with relevant information" in Figure 11 shows the six relevant transitions of MO1. Note the assignations of values to the `state0` variable in the six shaded regions. Particularly relevant is the modelling of the conditions that make these transitions to occur. In our example, this issue has been modelled by classical "if" clauses that consider the current state of the managed entity and the information of the channels.

For example, the transition from MO1_State1 to MO1_State 4 (see Fig. 10) occurs if the actual value of the variable `state0` is `MO1_State1` and the MO1 has received through its channels, the `send_event_1` notification (received from the MO2). At this point of the evolution, the value of the variable `state0` is set to `MO1_State4` by the sentence `state0 = MO1_State4`. After this the channels are updated as to make it official that the system has transitioned to MO1_State4. The above is modeled in the content of Figure 11 as follows:

```
:: state0 == MO1_State1 && current_event == send_event_1 && true … ->
   state0 = empty;
   state0 = MO1_State4;
   completed[cmplMO1_State4] = true;
   internal_queue!completion_MO1_State4;
   goto main
```

Other parts of the body of `proctype` declarations in PROMELA include the initialisation of the process evolution and the channel management declarations. These are also shown in Figure 11. A complete description of the PROMELA modelling language can be found at [Hol04]. The popularity of SPIN (Simple PROMELA Interpreter) has enabled the development of other tools intended for analysis of reactive systems. With this regard we can find translators of UML models into PROMELA code [Val04] which makes it much friendly and intuitive the use of PROMELA. The complete PROMELA representation of the example presented here is provided in the first part of Appendix A.

Process declaration

```
proctype ManagedObject1(chan event_queue; byte this; byte initialiser_MO20) {
```

Variables declaration

```
  chan ack_out;
  chan internal_queue = [completion_queue_size] of {byte};

  byte state0;
  byte state0_transition;
  bool completed[1];
  byte current_event;
  byte MO2;

  xr event_queue;
  atomic {
```

Initialization of references to collaborative entities e.g. MO2 and the variable that keeps record of MO1 behaviour

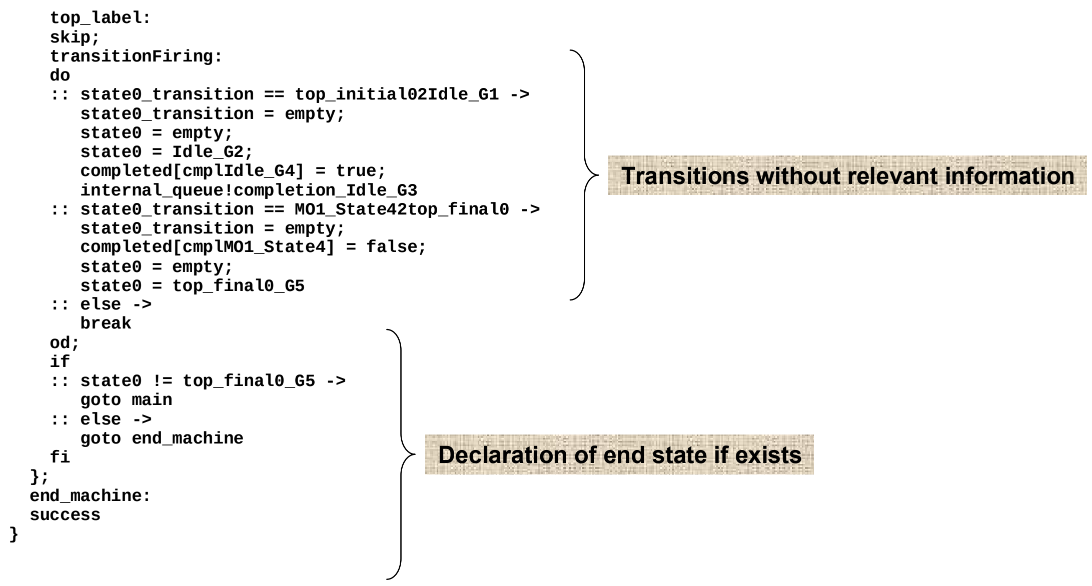
```
    MO2 = initialiser_MO20;
    state0 = top_initial0_G0;
    state0_transition = top_initial02Idle_G1;
    goto transitionFiring
  };
  main:
```

Channel management

```
  current_event = empty;
  if
  :: internal_queue?[current_event] ->
     internal_queue?current_event
  :: else ->
     if
     :: event_queue?[current_event, ack_out] ->
        event_queue?current_event,ack_out
     :: else ->
        event_queue?current_event,ack_out
     fi
  fi;
  atomic {
```

Transitions with relevant information

```
    if
    :: state0 == Idle_G2 && current_event == completion_Idle_G3
              && completed[cmplIdle_G4] == true && true && true ->
       completed[cmplIdle_G4] = false;
       state0 = empty;
       state0 = MO1_State2;
       goto main
    :: state0 == Idle_G2 && current_event == completion_Idle_G3
              && completed[cmplIdle_G4] == true && true && true ->
       completed[cmplIdle_G4] = false;
       state0 = empty;
       state0 = MO1_State1;
       goto main
    :: state0 == Idle_G2 && current_event == completion_Idle_G3
              && completed[cmplIdle_G4] == true && true && true ->
       completed[cmplIdle_G4] = false;
       state0 = empty;
       state0 = MO1_State3;
       goto main
    :: state0 == MO1_State1 && current_event == send_event_1 && true && true ->
       state0 = empty;
       state0 = MO1_State4;
       completed[cmplMO1_State4] = true;
       internal_queue!completion_MO1_State4;
       goto main
    :: state0 == MO1_State2 && current_event == send_event_2 && true && true ->
       state0 = empty;
       state0 = MO1_State4;
       completed[cmplMO1_State4] = true;
       internal_queue!completion_MO1_State4;
       goto main
    :: state0 == MO1_State3 && current_event == send_event_3 && true && true ->
       state0 = empty;
       state0 = MO1_State4;
       completed[cmplMO1_State4] = true;
       internal_queue!completion_MO1_State4;
       goto main
    :: state0 == MO1_State4 && current_event == completion_MO1_State4
              && completed[cmplMO1_State4] == true && true && true ->
       state0_transition = MO1_State42top_final0;
       goto top_label
    :: else
    fi;
```



```
      top_label:
      skip;
      transitionFiring:
      do
      :: state0_transition == top_initial02Idle_G1 ->
         state0_transition = empty;
         state0 = empty;
         state0 = Idle_G2;
         completed[cmplIdle_G4] = true;
         internal_queue!completion_Idle_G3
      :: state0_transition == MO1_State42top_final0 ->
         state0_transition = empty;
         completed[cmplMO1_State4] = false;
         state0 = empty;
         state0 = top_final0_G5
      :: else ->
         break
      od;
      if
      :: state0 != top_final0_G5 ->
         goto main
      :: else ->
         goto end_machine
      fi
    };
    end_machine:
    success
  }
```

Transitions without relevant information
Declaration of end state if exists

Figure 11. Fraction of PROMELA specification

2.3.3.2 Modeling System Requirements

For this second activity of the Model Checking technique, SPIN interprets Linear Temporal Logic (LTL) formulae. As we mentioned earlier, LTL formulae provide the means to specify any behavioural aspect of system executions. Consequently, SPIN can be seen as an interpreter of system behaviour.

2.3.3.3 Verification Process

This is the third activity of the Model Checking technique. In SPIN the Verification Process may consist of two options; (1) verify that a system satisfies some behavioural properties; and (2) find/report system executions associated to meaningful behavioural properties. Due to the relevance of the second option in our work, the remaining of this sub-Section focuses on the SPIN capabilities to find/report system executions

SPIN works producing system execution traces. These traces are sequences of inputs that indicate the conditions, events and states during system execution. In addition, SPIN reports how the managed entities collaborate during the execution trace. Given that the reports are associated to the provided LTL properties, the nature or the interpretation of the reports depends on the nature of those properties. LTL properties can be cast as either as a positive (desired) or a negative (undesired) property of the system model [Hol04].

- For "positive" properties, the reports provided by SPIN can be of two different types:
    1. Statements that the positive property is certainly satisfied.
    2. System executions that represent the behaviour that the system should exhibit so that the corresponding "negated" (undesired) property is satisfied.
- For "negative" properties, the reports provided by SPIN can be also of two different types:

1. Statements that the negative or undesired property never occurs during the system execution.
2. System executions that represent the behaviour that the system should exhibit so that the corresponding “positive” system property is satisfied.

Therefore note that there is the possibility to obtain execution traces that make a requirement property R not to hold, and the possibility to obtain execution traces that make a requirement property R to hold. From the above description we could summarise that if we provide R, SPIN would generate a system trace report (or all the execution reports if desired) in which R’ is satisfied. On the other hand, if we provide R’, SPIN would generate a system trace report (or all the execution reports if desired) in which R is satisfied. As a consequence of this, the resulting execution trace depends only on the type of requirement that the user provides

For instance, consider from our running example, Managed Object 1 (MO1) and its states MO1_State1 and MO1_State4 as we showed earlier in Figure 10. Assume the following positive/desired requirement R and its corresponding negative/undesired requirements R’:
R: “MO1_State4 is reached After MO1_State1”.
R’: “MO1_State4 is not reached After MO1_State1”

With the above requirements if we were to provide R as input to SPIN, we would obtain a system trace execution that would make R not to hold. In other words, we would obtain a system trace execution (or all the execution reports if desired) in which MO1_State4 is not reached After MO1_State1. Moreover, if we were to provide R’ as input to SPIN, we would obtain a system trace execution that would make the requirement R to hold. In other words, we would obtain a system trace in which MO1_State4 is reached After MO1_State1.

The central part of Figure 12 shows the visual representation of the system trace execution obtained when providing R’ to SPIN in our example. This represents the system trace executions that both, MO1 and MO2 should exhibit so that MO1_State4 is reached after MO1_State1. The left and right parts of Figure 12 show the interpretation of such execution with respect to the behaviour of MO1 and MO2, initially shown in Figure 10. Note for instance that the left part of Figure 12 shows the restricted behaviour of MO1 in which MO1_State4 is certainly reached after MO1_State1.

In addition, SPIN includes the provisioning of the details of the system execution expressed in terms of the PROMELA specification. The Figure 13 shows part of the execution report of our example. These reports show the actual lines of the PROMELA specification that are executed the by the corresponding Managed Objects. The complete execution report of this example is presented in the second part of Appendix A.

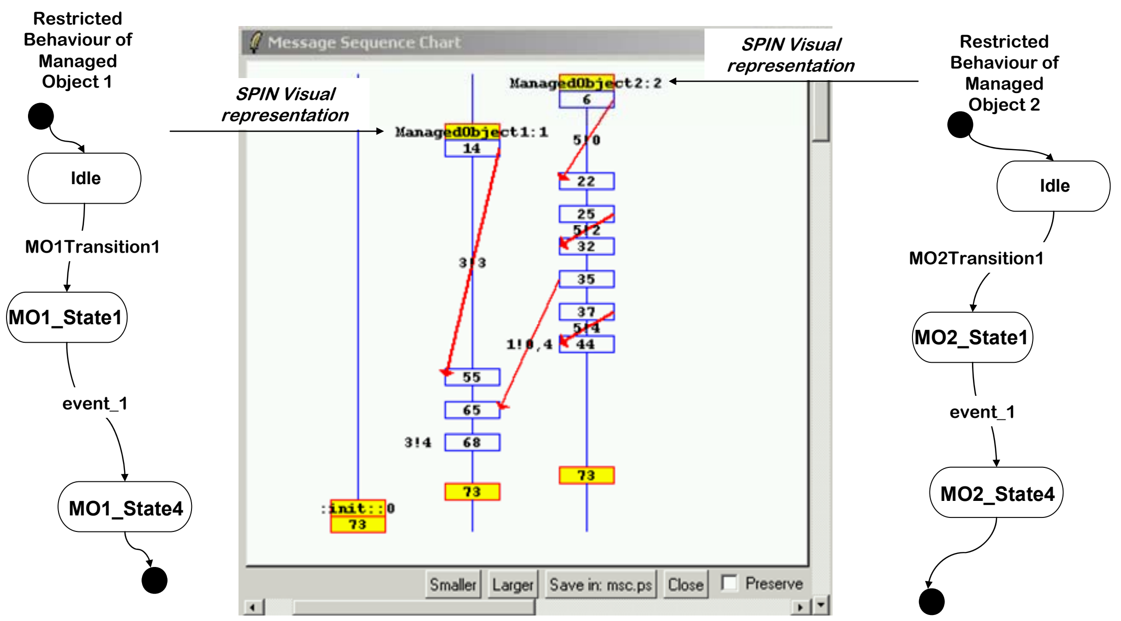


Figure 12. Visual representation of execution report

Sequence of execution for Managed Object 2

Sequence of execution for Managed Object 1

```
 4:  proc  2 (ManagedObject2) line  61 "pan_in" (state 1) [MO1 = initialiser_MO10]     <merge 83 now @2>
 4:  proc  2 (ManagedObject2) line  62 "pan_in" (state 2) [state0 = 1]      <merge 83 now @3>
 4:  proc  2 (ManagedObject2) line  63 "pan_in" (state 3) [state0_transition = 1] <merge 83 now @83>
 5:  proc  2 (ManagedObject2) line 135 "pan_in" (state 70)[((state0_transition==1))]    <merge 0 now @71>
 5:  proc  2 (ManagedObject2) line 136 "pan_in" (state 71)[state0_transition = 0] <merge 75 now @72>
 5:  proc  2 (ManagedObject2) line 137 "pan_in" (state 72)[state0 = 0]      <merge 75 now @73>
 5:  proc  2 (ManagedObject2) line 138 "pan_in" (state 73)[state0 = 2]      <merge 75 now @74>
 5:  proc  2 (ManagedObject2) line 139 "pan_in" (state 74)[completed = 1]   <merge 75 now @75>
 6:  proc  2 (ManagedObject2) line 140 "pan_in" (state -) [values: 5!0]
 6:  proc  2 (ManagedObject2) line 140 "pan_in" (state 75)[internal_queue!0]
 7:  proc  2 (ManagedObject2) line 146 "pan_in" (state 81)[else]
 8:  proc  2 (ManagedObject2) line 150 "pan_in" (state 86)[((state0!=7))]   <merge 0 now @6>
10:  proc  2 (ManagedObject2) line  67 "pan_in" (state 6) [current_event = 0]
12:  proc  1 (ManagedObject1) line 172 "pan_in" (state 1) [MO2 = initialiser_MO20]     <merge 71 now @2>
12:  proc  1 (ManagedObject1) line 173 "pan_in" (state 2) [state0 = 1]      <merge 71 now @3>
12:  proc  1 (ManagedObject1) line 174 "pan_in" (state 3) [state0_transition = 1] <merge 71 now @71>
13:  proc  1 (ManagedObject1) line 234 "pan_in" (state 58)[((state0_transition==1))]    <merge 0 now @59>
13:  proc  1 (ManagedObject1) line 235 "pan_in" (state 59)[state0_transition = 0] <merge 63 now @60>
13:  proc  1 (ManagedObject1) line 236 "pan_in" (state 60)[state0 = 0]      <merge 63 now @61>
13:  proc  1 (ManagedObject1) line 237 "pan_in" (state 61)[state0 = 2]      <merge 63 now @62>
13:  proc  1 (ManagedObject1) line 238 "pan_in" (state 62)[completed = 1]   <merge 63 now @63>
14:  proc  1 (ManagedObject1) line 239 "pan_in" (state -) [values: 3!3]
14:  proc  1 (ManagedObject1) line 239 "pan_in" (state 63)[internal_queue!3]
15:  proc  1 (ManagedObject1) line 245 "pan_in" (state 69)[else]
16:  proc  1 (ManagedObject1) line 249 "pan_in" (state 74)[((state0!=7))]   <merge 0 now @6>
18:  proc  1 (ManagedObject1) line 178 "pan_in" (state 6) [current_event = 0]
```

Figure 13. Fragment of detailed execution report

In conclusion, the Model Checking technique provides the means to acquire the necessary system trace executions in accordance with specific LTL requirements properties characterising requirements of behavioural aspects of a reactive system. In addition, SPIN allows to achieve this in a fully automatic manner. The execution reports show the managed entities behaviour as a collaborative sequence of conditions, states and actions that the involved entities should exhibit as to commit with the LTL characterisation. For a detailed description of SPIN and its reference manual, please refer to [Hol04].

2.4 Conclusions

In this Chapter we have provided background material on two technical issues that concern with our policy refinement analysis; (1) the principles to formalise system requirements, concretely we have described the principles, the scope, activities, foundations and the practicality of Goal-oriented Requirements Engineering techniques; and (2) a general description on the utilisation of reactive systems analysis techniques, more specific we have provided the general outline of Model Checking techniques.

The Goal-oriented Requirements Engineering techniques provide support to establish formal representations, at different levels of abstraction, of the requirements that an envisioned system should fulfill. Concretely, we have outlined the KAOS goal elaboration method. This method provides the means to carry out goal-based reasoning activities grounded in Linear Temporal Logic (LTL). KAOS proposes to use goal refinement patterns as a standard method to identify and make decisions on the achievements of the target systems. The system requirements are formalised into goal-graph structures that may represent either the different strategies with which the envisioned system may fulfill specific requirements, or the necessary requirements that a system may include to fulfill the high-level administrative needs.

While KAOS provides support to document and elaborate goal graph structures through the Requirements Elaboration process, it does not provide support to relate managed objects' behaviour to goal fulfillment finding. In this sense, the goal graph structures elaborated with KAOS should be acknowledged as a potential source of information to overcome these limitations. With this regard, we have provided the principles of the Responsibility Assignment and Operationalisation activities. While the former is meant to identify the agent objects and their responsibilities as to guarantee the system requirements, the latter is meant to identify the operations and their domain pre- and post-conditions of such agent objects. These two activities are completely dependant on the application domain and should be supported by the information formalised with the KAOS goal graph structures.

Another technical issue that we have outlined in this Chapter concerns with the analysis of reactive systems, more concretely we have provided the principles of verification techniques through Model Checking. The key for the success of the latter technique relies on the accuracy to asses two major aspects for analysis: (1) the verification that the specification of a system fulfills specific properties; and (2) the acquisition of meaningful system behaviour as a response to the input of behavioural properties. With this regard, a key issue is the specification of behavioural properties and their representation with formal notations suitable to use with automated Model Checking engines.

We have outlined the principles of specification of behavioural properties for LTL and the corresponding linear temporal Model Checking process, effectively assessed with the SPIN searching engine. We have demonstrated the feasibility to express behavioural properties in LTL and the capability of the SPIN searching engine to acquire and report meaningful system behaviour committing to such behavioural specifications. The system behaviour is reported as a collaborative sequence of events, conditions and actions that specific managed entities may exhibit in runtime. It is worth noticing that this is achieved by a fully LTL-based state exploration process.

Chapter 3 A Policy Refinement Framework

3.1 Introduction

Policy-based management has been regarded as a potential approach to allow the management tasks in complex environments in network and service management. Despite having been introduced in different application domains through various research projects, several standardisation efforts and substantial interest from industry, policy-based management is still not widespread used. The number of policies necessary to control medium- to large-scale systems may be in the order of thousands and these policies are likely conflicting. In addition to policy conflict analysis techniques, a key issue is the need to derive policies from high-level administrative goals or from higher level policies.

Policy refinement is understood as the process aimed to derive lower-level policies from higher-level ones so that the former are better suited for use in different execution scenarios. Although policy refinement has been recognised as crucial for the success of policy-based management, it has been severely dismissed due probably to its inherent complexity. In this sense, a holistic approach to the policy refinement process still remains unclear.

The main motivation of this chapter is to provide a framework that considers the different aspects involved in the policy refinement process. One of these aspects is the need to represent the aim of policies at different levels of abstraction. For this, we consider the aim of policies as goals [Lam99], [Ban04] and hence we use goal refinement methodologies as the means to ground of the policy refinement process. In this way, the proposed framework is built upon goal-oriented requirements engineering methodologies. In other words, this framework is intended to specialise the GORE (Goal-Oriented Requirements Engineering) into a policy authoring environment.

In addition to laying down the aforementioned formal concepts into a policy refinement framework, another aspect that we address in this chapter is the identification of the processes and activities involved in the refinement paradigm. This is, we clarify the nature of the different tasks that the administrative parties should carry out during the life cycle of the policy-based management system.

The goal elaboration process in our framework relies on the KAOS approach [Dar96], [Dar98], [Lam99]. As presented in Chapter 2, KAOS provides the theoretical support to formalise the requirements of a target system into goal graph structures. Nevertheless, it doesn't provide support to relate systems' behaviour to goal fulfillment. Another aspect that we address in this chapter is the need to produce enforceable/deployable policies from High-level Goals in a systematic manner. This is, we formalise the necessary mechanisms that assess the Responsibility Assignment and Operationalisation activities of the KAOS GORE approach in our policy authoring environment.

We support Responsibility Assignment and Operationalisation activities by means of Linear Temporal Logic and Model Checking verification techniques [Rub05]. These techniques are applied in the policy refinement mechanisms as the means to provide a formal procedure to obtain runtime system trace executions aimed at fulfilling lower-level goals, and consequently the fulfillment of High-level Goals.

Besides the framework addressing the holistic implications of the policy refinement process, this Chapter also provides the overall implementation of a solution for generating policies following this one framework [Rub06a], [Rub06b]. This is a step ahead towards the materialisation of the policy refinement paradigm based on the framework proposed.

After this Introduction, this Chapter is sectioned in three parts: Section 3.2 provides the rationale of the policy refinement framework; Section 3.3 describes a prototype solution that entails the former; and Section 3.4 provides the conclusions on our proposal.

3.2 Rationale of the Policy Refinement Framework

In policy authoring contexts, there is a need to formalise the aim of policies. Having adopted the view of considering the aim of policies as goals [Lam99], [Ban04], the general idea behind the policy framework is to specialise the GORE concepts of Requirements Elaboration, Responsibility Assignment and Operionalisation into a policy authoring environment. In this sense, the final outcome of the process would be a set of deployable and enforceable policies committing with high-level administrative goals. The following is the overall rationale of the elements and activities involved in the policy refinement process.

3.2.1 Actors and roles

In a policy authoring environment, policies are refined before and/or during the operation of the system by an administrative party. Moreover, in order to refine policies in a systematic manner, previous supporting activities should be carried out during the design and implementation of the management system. In this sense, we consider two administrative parties; an Administrator Developer and an Administrator Consultant. On the one hand, the Administrator Developer carries out the supporting activities during the design and implementation, and on the other hand, the Administrator Consultant carries out the refinement-aware activities during the start-up and operation of the policy system.

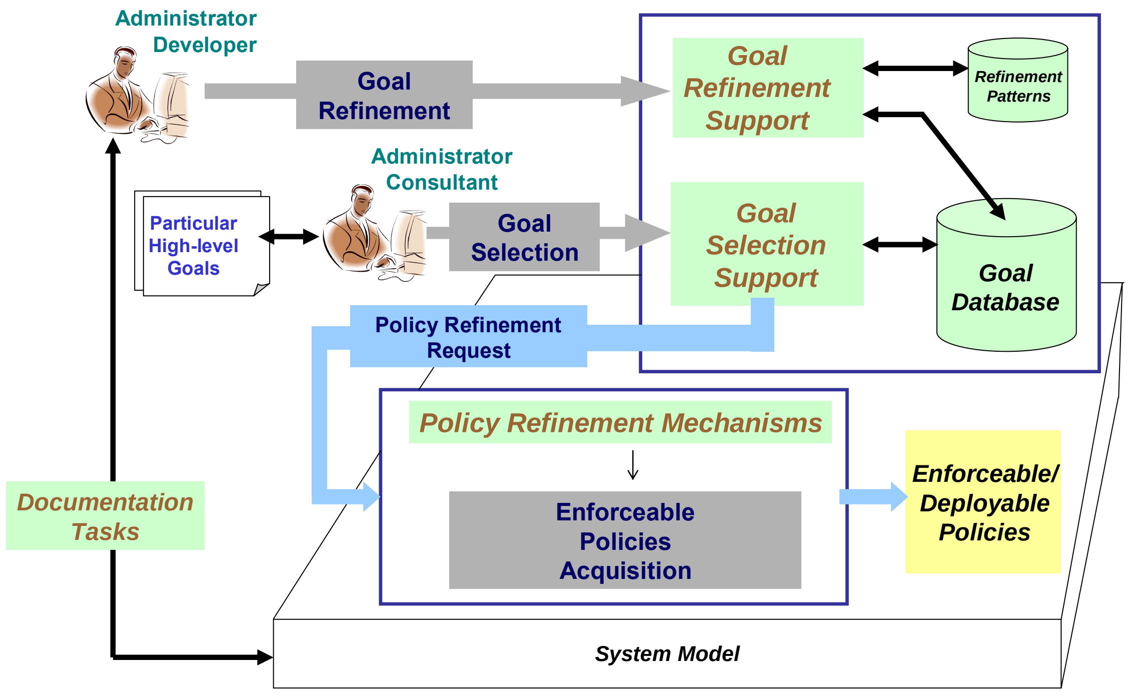


Figure 14. Overall Policy Refinement Framework

The framework shown in Figure 14 provides support for both, Administrator Developer and Administrator Consultant. In administrative terms, the framework considers the following activities:

- *Goal Refinement*. This activity is intended to elaborate goal graph structures of the High-level Goals that the system can handle. These goal-graphs represent both, the requirements and the different options with which the High-level Goals could be achieved. This activity is application-dependent and is carried out by the Administrator Developer, and consequently this activity is carried out during the design and implementation of the system.
- *Goal Selection*. This activity is meant to define the Administrator Consultant's "particular" goals[1] for the functional policy-based system. This activity is achieved by selecting, from the different options represented in the goal-graph structures, the goal strategies that better reflect the Administrator Consultant's criteria. We should point out that Goal Selection is carried out at the beginning of the operation of the system operation and at service operation time.
- *Enforceable Policies Acquisition*. This activity is meant to produce a set of lowest-level policies that would fulfill the "particular" High-level Goals resulting from the Goal Selection activity. The acquired policies should include meaningful elements of policies like subjects, targets, events and actions. It is worth mentioning that these policy fields should commit with the managed system capabilities i.e. these should

[1] During this work the term "particular" goals is used to denote the goals defined by the Administrator Consultants which are in turn reproductions of the goals defined by the Administrator Developer.

produce enforceable policies. This activity uses the System Model. The later includes the detailed behaviour of the managed elements through finite state machines, class and collaboration models of the management system, and their object distribution within the managed system. In this sense, the acquired policies should commit with the actual object distribution of the managed system so that not only the policies can be enforced but also for further deployment of these policies onto the managed system. The System Model is provided by the Administrator Developer at design/implementation time.

3.2.2 Goal Refinement Support

Figure 15 shows the general outline of the Goal Refinement Support function whose general objective is to allow the Administrator Developer to elaborate the goal graph structures that the policy driven system can handle. The challenge in this activity is to make use of KAOS [Dar96], [Dar98], an application independent refinement approach, in order to build an application-dependent refinement graph.

The Administrator Developer uses the KAOS application-independent refinement patterns combined with the managed system features (i.e. the managed system capabilities) to drive the goal refinement process. Once the High-level Goals have been defined, these are further refined through the instantiation of refinement patterns that enable the developer to identify lower-level goals. After the goal graph structures have been elaborated, these are stored in a goal database for eventual use and/or maintenance. This activity should be accomplished during the design and development phase of the system.

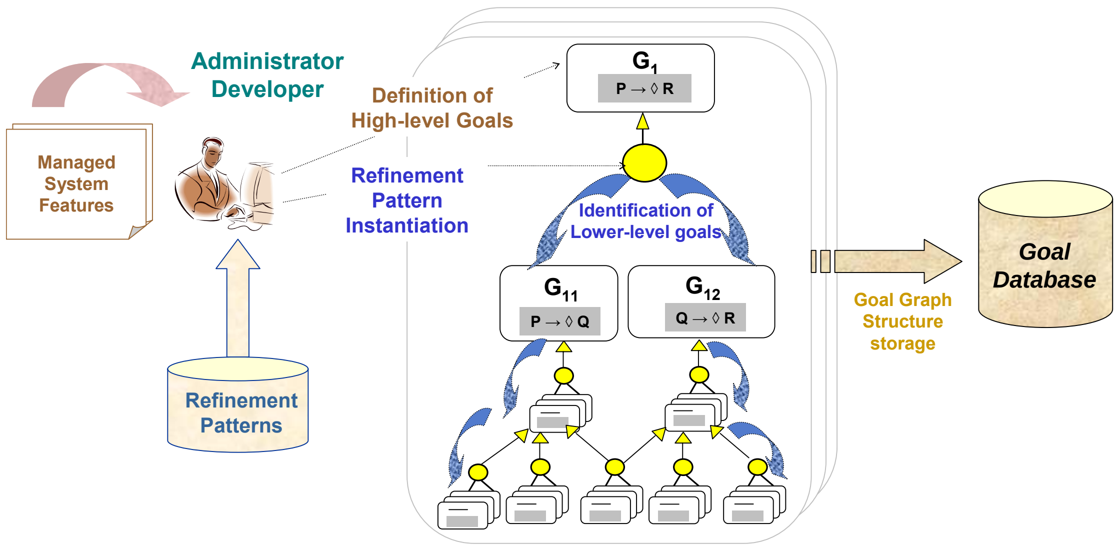

Administrator Developer
Managed System Features
Definition of High-level Goals
Refinement Pattern Instantiation
Refinement Patterns
G1
P → ◊ R
Identification of Lower-level goals
G11
P → ◊ Q
G12
Q → ◊ R
Goal Graph Structure storage
Goal Database

Figure 15. Goal graph structure elaboration outline

In order to understand better the Goal Refinement Support function, consider for instance a policy-driven Network Dimensioning (ND) system for which the Administrator Developer wants to elaborate the goal graph structure within this framework.

3.2.2.1 *Identification of Managed System Features*

Consider that the developer has identified the following features of the system:

- *General purpose*: The ND sub-system is in charge of assigning network resources according to given traffic predictions in the network. This is our policy-driven element or managed system
- *Input*: It receives a traffic estimation matrix that defines the minimum and maximum values for traffic estimations.
- *Output*: It produces configuration sets that may eventually be propagated to enforce the traffic allocation calculations.
- *Capabilities*: It basically integrates policy-based dimensioning mechanisms that influence the allocation of resources. These mechanisms include functions affecting the following parameters: limit of hop-counts, overall utilisation of the network, hop-count estimation for delay and packet loss estimations, explicit allocations, distribution/reduction of extra link capacity.

3.2.2.2 *Defining the nature of High-level Goals*

Consider the case where the administrator must ensure a given system behaviour; this is, the way it will assign network resources according to specific traffic distributions. The above could be interpreted as a High-level Goal, identified as **ConfigurationDirectives-Set** and formalised in Linear Temporal Logic as follows:

G1 ConfigurationDirectivesSet: ND_Request→◊DirectConfig&Propagated

In the context of the goal refinement process, the semantics of administrator's goal G1 is "when a network dimensioning request is received, the configuration directives should be generated and eventually stored and propagated to the underlying components". Other High-level Goals could be defined for the ND system for which the intersection of all would become the different targets of interest that the Administrator Developer has envisaged as controllable for the ND system.

3.2.2.3 *Goal refinement assisted by refinement patterns*

Once the High-level Goals like G1 have been identified, they are decomposed into offspring goals. The KAOS method prescribes to tackle this issue by asking HOW questions. In the context of our ND example this is equivalent to figure out how the system can configure dimensioning directives. The ultimate objective is to achieve some behaviour in the envisioned system. For this purpose, different propositional patterns would be useful. For example, consider the selection of KAOS patterns to refine the application-independent *Achieve* goal P →◊Q shown in Table 3.

ID	Formal Representation of sub-goals			Description
RP3	P → ◊R	R → ◊Q		Milestone-driven refinement
RP4	P ∧ P1→◊Q1	P ∧ P2→◊Q2	□(P1 ∨ P2) Q1 ∨ Q2 →Q	Case-driven refinement
RP5	P ∧ ¬ R→◊R	P ∧ R→ ◊Q	P→□P	Conditional milestone-driven
RP6	¬ R→ ◊R	P ∧ R→ ◊Q	P→□P	Inconditional milestone-driven

Table 3 Different patterns to refine the *Achieve* goal P →◊Q

The use of one pattern instead of another depends indeed on the capabilities of the system and the criteria of the developer. Following on with our sample, suppose P stands for "ND_Request" (reception of a network dimensioning request) and Q stands for "DirecConfig&Propagated" (ND directives generated and propagated). Suppose also that R stands for "TM_reception" (reception of the traffic matrix). The patterns shown in Table 3 are brought to this specific application domain as follows:

- The "Milestone-driven refinement" pattern, identified as RP3 proposes that *"on the occurrence of a state P, an intermediate state satisfying R must first be reached from which a goal state Q must eventually be reached"*. In our ND example this pattern proposes that on the occurrence of "ND_Request" state, a state prescribing "TM_reception" may first be reached from which the goal state "DirecConfig&Propagated" must be reached. In other words, two goals would be necessary to fulfill the original **G1 ConfigurationDirectivesSet** goal.
- The "Case-driven refinement" pattern, identified as RP4 proposes that *"on the occurrence of P, an alternative sate satisfying either P1 or P2 will eventually satisfy the goal states Q1 or Q2 respectively which in turn suffices to satisfy the state Q"*. In our ND example this pattern proposes that on the occurrence of an "ND_Request" state, it is possible to take two alternative intermediate goal states P1 or P2 which in turn should eventually satisfy the goal states Q1 or Q2 respectively. Also, either Q1 or Q2 would yield to the "DirecConfig&Propagated" goal state. In other words, the administrator should identify two alternatives to satisfy the **G1 ConfigurationDirectivesSet** goal.
- The "Conditional milestone-driven" pattern, identified as RP5 proposes that *"on the occurrence of P and under the absence of the state R, it is a requirement that state R and state P both hold so that the goal state Q eventually holds". "In addition, state P should hold "always" during the execution of the system.* In our ND example, RP5 proposes that on "TM_reception", it should hold that both "TM_Reception" and "ND_Request" to be present at the same time as for the goal state "DirecConfig&Propagated" to eventually hold. Also, this pattern proposes that the state "ND_Request" to be present "always" until the goal state "DirecConfig&Propagated" is reached.
- The "Inconditional milestone-driven" pattern, identified as RP6 proposes that "*on the absence of a state satisfying R, the latter must hold when P occurs so that the goal state Q can be eventually reached"*. In our ND example, RP6 proposes that on the absence of "TM_Reception", the latter must eventually hold so that on the occurrence of "ND_Request" both hold and consequently, the goal state "DirecConfig&Propagated" can be eventually reached.

Continuing in the ND example, the developer evaluates how the above four alternatives can match the capabilities and the operation of the ND system to decide which refinement pattern is better suited to use to decompose the **G1 ConfigurationDirectivesSet** goal.

For instance, RP5 is discarded because the reception of a configuration request (ND_Request) is a discrete event that cannot happen continuously until the traffic matrix (TM_reception) is received nor until the directives are configured and propagated (DirecConfig&Propagated). RP6 is also discarded because the reception of a traffic matrix (TM_reception) is also a discrete event that cannot hold until the "ND_Request" state is reached.

The two remaining options are then a milestone-driven tactic (refinement pattern RP3) and a refinement by cases (refinement pattern RP4). The decision of which pattern to apply to refine the original **G1 ConfigurationDirectivesSet** goal is taken by the Administrator Developer following the propositions of the two alternatives (RP3 and RP4) and the design/capabilities of the policy system. With regard to the latter, let us assume that the dimensioning of the network can be achieved in two ways: (1) considering the allocation of minimum demand or (2); considering the allocation of minimum demand plus the usage of the remaining physical network resources. These two options are exclusive one from the other and both can be used to configure the directives in the ND system. For this reason, the milestone-driven pattern RP3 is discarded as it may imply that both options are necessary to fulfill G1. In other words, bringing RP3 into this application at this stage may imply that the ND system should be configured considering the allocation of minimum demand and also, considering again the allocation of minimum demand, plus the usage of the remaining physical resources.

Finally, bringing the case-driven refinement pattern RP4 into our Network Dimensioning example formalises two different alternatives of how to refine the original **G1 ConfigurationDirectivesSet** goal. More concretely, the developer brings the case-driven RP4 pattern to refine G1 as he considers that the High-level Goal **G1** could be fulfilled in two ways which in turn define the goals **G2** and **G3** respectively: **G2** opts to dimension the network considering the allocation of minimum demand; **G3** opts for network dimensioning considering the allocation of minimum demand plus the usage of the remaining physical network resources. These two refinements are graphically represented in the upper part of Figure 16 and are formally expressed as follows:

G1 ConfigurationDirectivesSet: ND_Request → ◊ DirectConfig&Propagated
G2 MinDemandStrategy: ND_Request /\ minDemandStrategy → ◊ DirectConfig&Propagated
G3 MinWExtraCapStrategy: ND_Request /\ minWExtraCapStrategy→ ◊DirectConfig&Propagated

G2 should be interpreted as follows; "when a network configuration request is received, configuration directives for the minimum demand will be eventually generated, stored and propagated". Similarly, G3 should be interpreted as follows; "when a network configuration request is received, configuration directives for the minimum demand plus the usage of the remaining physical resources will be eventually stored and propagated".

We must acknowledge that the Goal Refinement process is aimed at elaborating goal graph structures of High-level Goals that the system can handle. These represent both, the requirements and the different options with which the High-level Goals could be achieved. As we will describe latter, one of the above two options to generate the configuration directives would be selected by means of the Goal Selection process during the start up and/or the operation of the system.

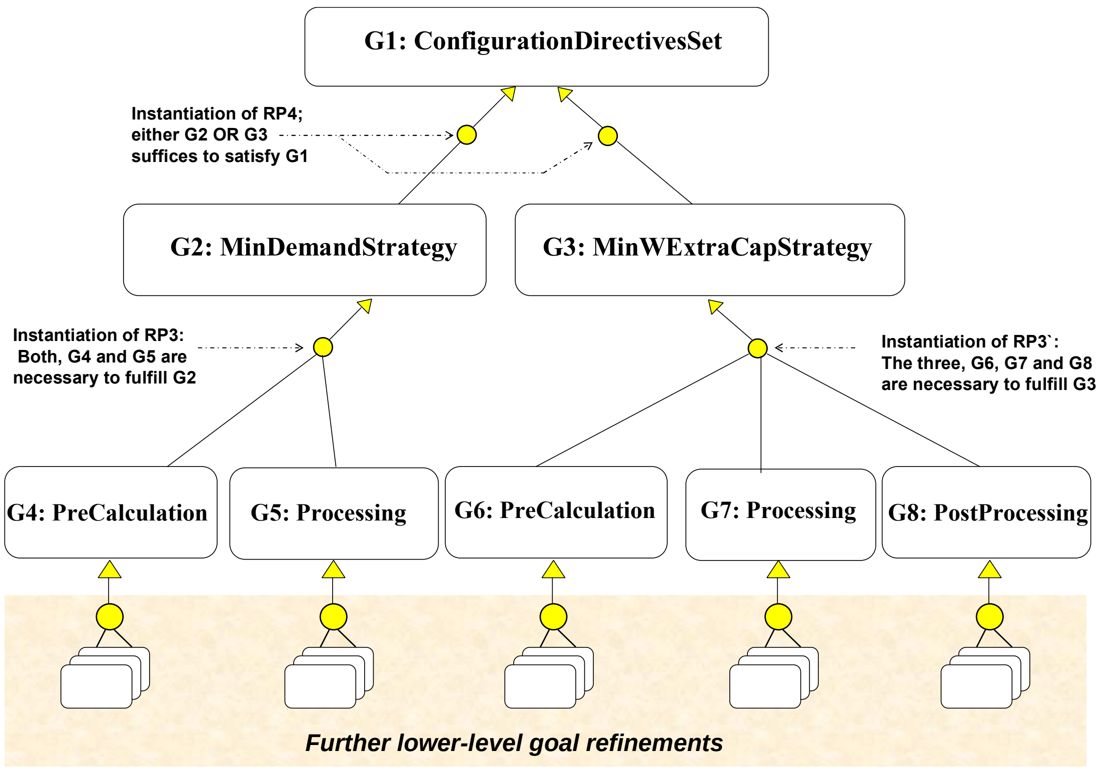

G1: ConfigurationDirectivesSet
Instantiation of RP4; either G2 OR G3 suffices to satisfy G1
G2: MinDemandStrategy
G3: MinWExtraCapStrategy
Instantiation of RP3: Both, G4 and G5 are necessary to fulfill G2
Instantiation of RP3`: The three, G6, G7 and G8 are necessary to fulfill G3
G4: PreCalculation
G5: Processing
G6: PreCalculation
G7: Processing
G8: PostProcessing
Further lower-level goal refinements

Figure 16. Initial goal graph structure elaboration process

The patterns shown in Table 3 can be extended to make them suitable for use in particular application domains. For instance, RP3 and RP4 could be extended to deal with multiple milestone-driven and multiple case-driven situations as shown in Table 4.

RP	Formal Representation of sub-goals	Description
RP3'	$P \rightarrow \Diamond R$ $R \rightarrow \Diamond S$ $S \rightarrow \Diamond Q$	Multiple milestone-driven refinement
RP4'	$P \wedge P1 \rightarrow \Diamond Q1$, $P \wedge P2 \rightarrow \Diamond Q2$, $P \wedge P3 \rightarrow \Diamond Q3$, $\Box(P1 \vee P2 \vee P3)$, $Q1 \vee Q2 \vee Q3 \rightarrow Q$	Multiple case-driven refinement

Table 4 Extension of the refinement patterns RP3 and RP4

Going further in the refinement process of our example, **G2 MinDemandStrategy** and **G3 MinWExtraCapStrategy** should be further refined. For **G2 MinDemandStrategy**, the developer considers that this requirement must be accomplished in two steps: (1) a pre-calculation of resources; (2) a processing step that should take into account the pre-calculations. Consequently, the goal **G2 MinDemandStrategy** is refined into the sub-goals **G4 PreCalculation** and **G5 Processing** by bringing the pattern RP3 into the context of the refinement of G2 (see graphical representation in Figure 16). This goal sub-tree formalises the requirement of achieving network dimensioning configurations in two steps; a pre-calculation AND the processing step properly said.

Similarly, for the goal **G3 MinWExtraCapStrategy**, the developer considers that in addition to calculating the allocation of the minimum demand, **G3** requires a post-processing step to allocate the remaining capacity in the core links of the network and to reduce the capacity when the links are over-provisioned. Consequently the goal **G3 MinWExtraCapStrategy** is refined into the sub-goals **G6 PreCalculation, G7 Processing** and **G8 PostProcessing**, by the bringing the extended pattern RP3' into the context of the refinement of G3 (see graphical representation in Figure 16). The sub-tree defined by **G3** formalises the requirement of having a processing step AND an additional post-processing calculation step. The above mentioned processes yield the third level in the refinement graph shown in Figure 16. These third level goals are formally represented in KAOS as follows:

G4: ND_Request /\ minDemandStrategyReq /\ preCalculation → ◊ Processing
G5: Processing → ◊ DirectConfig&Propagated
G6: ND_Request /\ minWExtraCapStrategyReq /\ preCalculation → ◊ Processing
G7: Processing → ◊ PostProcessing
G8: PostProcessing → ◊ DirectConfig&Propagated

The semantics of the above abstractions is as follows:
G4.- "When a Network Dimensioning process is Requested (ND_Request) and the minimum demand strategy is required (minDemandStrategyReq), a pre-calculation process for dimensioning (preCalculation) will eventually result in a processing step (Processing)".
G5.- "A Processing step (Processing) will eventually result in directives being configured and propagated (DirectConfig&Propagated)".
G6.- "When a Network Dimensioning process is Requested (ND_Request) and the minimum demand with extra capacity strategy is required (minWExtraCapStrategyReq), a pre-calculation process for dimensioning (preCalculation) will eventually result in a processing step (Processing)".
G7.- "A Processing step (Processing) will eventually result in a post-processing step (PostProcessing)".
G8.- "A post-processing step (PostProcessing) will eventually result in directives being configured and propagated (DirectConfig&Propagated)".

At this stage of the goal refinement process, the developer determined the initial steps and alternatives by means of which the ND system would be able to generate network configuration primitives. At the same time, this intermediate steps have been formalised in terms of temporal logic, which will be important for subsequent stages. Nevertheless, the developer still needs to identify and formalise the requirements and options for each lower-lever refinement (**G4** to **G8** in Fig. 16). In other words, a similar procedure has to be carried out to refine each lower-level goal.

For example, the **PreCalculation** sub-goal (**G4** and **G6** in Figure 16) is refined into two alternative cases ; an alternative that considers estimations for delay/loss purposes (**G9 delayLossEstimated**), and another alternative that considers delay/loss estimations with explicit resources allocation (**G10 delayLossEstimationWithExplicitAllocation**). The result is a goal graph structure linked to the **PreCalculation** goal as shown in Figure 17. For each of these two lower-level goals (**G9** and **G10** in Figure 17), further refinements are necessary until the developer finds it feasible to relate the refinements with a specific state of the system. In fact, the KAOS methodology estates that a goal that is identifiable with a state of the system can be considered as a lowest-level goal. The later situation is exemplified hereafter with the refinement applied to the **G9 delayLossEstimated** sub-goal.

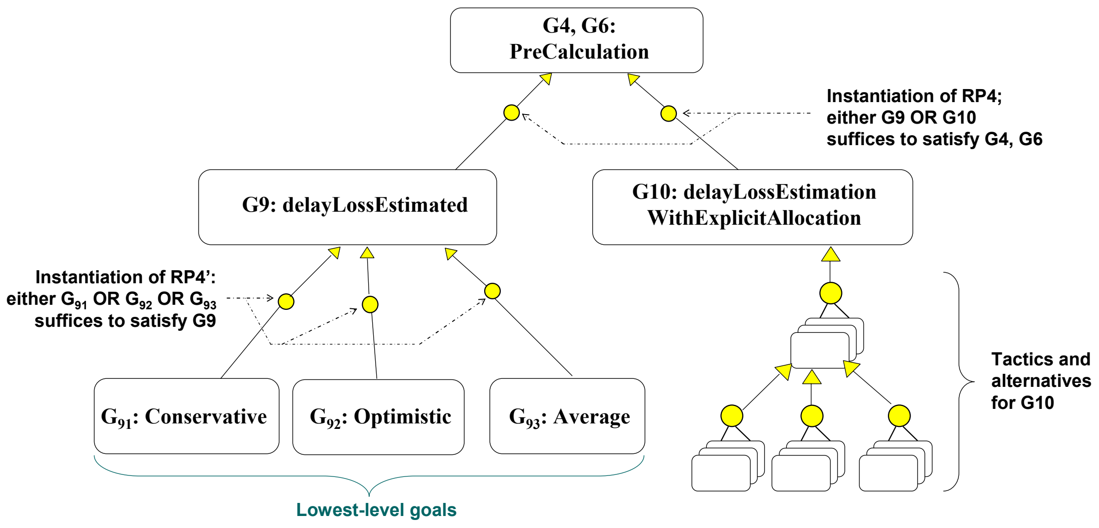

G4, G6: PreCalculation
Instantiation of RP4; either G9 OR G10 suffices to satisfy G4, G6
G9: delayLossEstimated
G10: delayLossEstimation WithExplicitAllocation
Instantiation of RP4': either G91 OR G92 OR G93 suffices to satisfy G9
G91: Conservative
G92: Optimistic
G93: Average
Tactics and alternatives for G10
Lowest-level goals

Figure 17. Goal graph sub-tree for the PreCalculation sub-goal

The sub-goal **G9** is further refined into three alternative cases by means of the extended refinement pattern RP4'. The result is a set of three refinements expressed as **G_{91} Conservative**, **G_{92} Optimistic** and **G_{93} Average**. **G_{91}** proposes to consider the maximum delay and the maximum packet-loss recorded in the network as the base to estimate the delay and packet loss during the dimensioning process; taking the maximum values for each link is considered to be a *conservative tactic* in the pre-calculation process. On the other hand, **G_{92}** considers the minimum values for delay and packet loss; this is considered an *optimistic tactic*. Finally **G_{93}** proposes a tactic that considers the *average values* of delay and packet losses in every link of the network, as the base to estimate the delay and packet loss during the calculation process. These three goal refinements (**G_{91}**, **G_{92}** and **G_{93}**) are specific enough to relate them to states of the system; these are considered as lowest-level goals given that there is not further tactic to refine them. The sub-goal **G9 delayLossEstimated** and its refinements **G_{91} Conservative**, **G_{92} Optimistic** and **G_{93} Average** are graphically represented in Figure 17 and are formally represented as follows:

G9: PreCalculated $\wedge$ delayLossEstimatedReq$\rightarrow \Diamond$ Processed
G_{91}: PreCalculated $\wedge$ delayLossEstimatedReq $\wedge$ conservativeReq $\rightarrow \Diamond$ Processed
G_{92}: PreCalculated $\wedge$ delayLossEstimatedReq $\wedge$ optimisticReq $\rightarrow \Diamond$ Processed
G_{93}: PreCalculated $\wedge$ delayLossEstimatedReq $\wedge$ averageReq $\rightarrow \Diamond$ Processed

The semantics of the above abstractions is as follows:
G9.- "When a pre-calculation step is requested (PreCalculated), a delay/loss estimation process (delayLossEstimatedReq) will eventually result in a processed state (Processed)".
G_{91}.- "When a pre-calculation step is requested (PreCalculated) and a delay/loss estimation process

is under request (delayLossEstimatedReq), a conservative selection of settings (conservativeReq) will eventually result in a processed state (Processed)”.
G_{92}.- “When a pre-calculation step is requested (PreCalculated) and a delay/loss estimation process is under request (delayLossEstimatedReq), an optimistic selection of settings (optimisticReq) will eventually result in a processed state (Processed)”.
G_{93}.- “When a pre-calculation step is requested (PreCalculated) and a delay/loss estimation process is under request (delayLossEstimatedReq), an average selection of settings (averageReq) will eventually result in a processed state (Processed)”.

The goal refinement process is finalised when the lowest level goals for all possible links in the graph are identified. Therefore, in our example, the developer should carry out a refinement procedure for the sub-goal **G10 delayLossEstimationWithExplicitAllocation** similar to the one carried out for the sub-goal **G9 delayLossEstimated**. Likewise, the developer should proceed with the remaining refinements of the highest-level goal **G1 ConfigurationDirectivesSet**

3.2.3 Goal Selection Support

Once system goals refinement is completed, the goal graph structures should be stored in databases for further use and maintenance. Having in mind that goal graph structures formalise potential requirements and options to fulfill High-level Goals, they will be used to select a concrete one among the different available alternatives.

Goal Selection Support consists of browsing through the previously elaborated goal graph structures stored in the goal database and selecting the strategies that better reflect the consultant administrative criteria.

Following with our scenario example, consider the case where the consultant wants to provide configuration directives only for the minimum demand estimations. Consider also the situation where the consultant wants to have better chances to fulfill the SLAs of real-time services. This is his particular administrative guideline to navigate through the graph that finally yields to the selection of the path reflected in Fig 17 with shadow boxes.

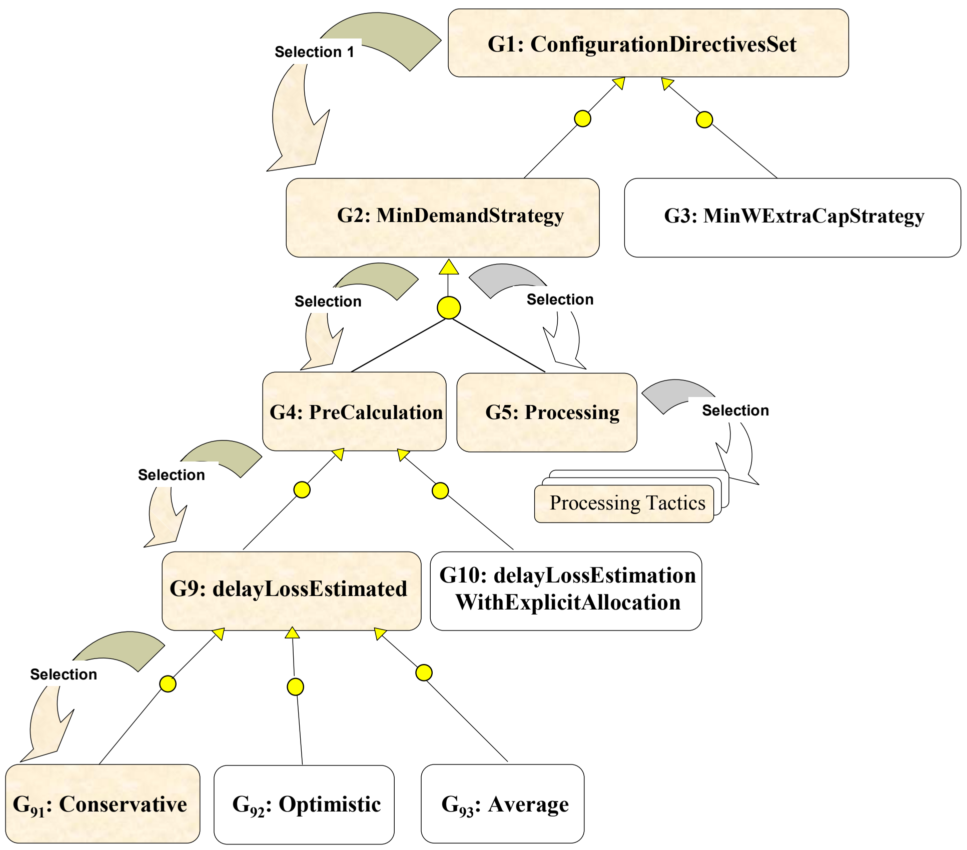


Figure 18. Sample of a Goal Selection action

3.2.4 Policy Refinement Mechanisms

The Policy Refinement Mechanisms supports the *Enforceable Policies Acquisition* activity of the policy refinement framework (see Figure 14). The target of these mechanisms is to determine policies fulfilling the goals resulting from the Goal Selection. With this aim, we make use of the logical foundations of the goal elaboration methodology, reactive systems analysis techniques and novel concepts developed on purpose to make the policy refinement activity a systematic process. In this sub-section we first provide a general outline of the policy refinement mechanisms and then we describe the involved sub-processes.

3.2.4.1 General Outline of the Policy Refinement Mechanisms

The principle of the policy refinement mechanisms is to abstract enforceable policies that can be deployed onto the managed system to commit with a given goals selection. The whole process can be decomposed into four sequential steps as follows: (1) *Establishment of Temporal Relationships*; (2) *Enforcement of System Behaviour*; (3) *Translation Process*; (4) *Encoding of Deployable policies*. These processes are graphically represented in Figure 19 and are briefly outlined hereafter.

1. *Establishment of Temporal Relationships*. This process is aimed at characterising the lowest-level goals as system specifications that are suitable for use with automated analysis techniques.
2. *Enforcement of System Behaviour*. This process is aimed at forcing the underlying managed system behaviour so that the specifications provided by the "Establishment of Temporal Relationships" process can be fulfilled.
3. *Translation Process*. This process is aimed at identifying a subset of transitions in the restricted system behaviour as event-condition-action policies.
4. *Encoding of Deployable policies*. This process produces the policies that should be deployed onto the actual managed system from the policy-controlled transitions provided in the previous step.

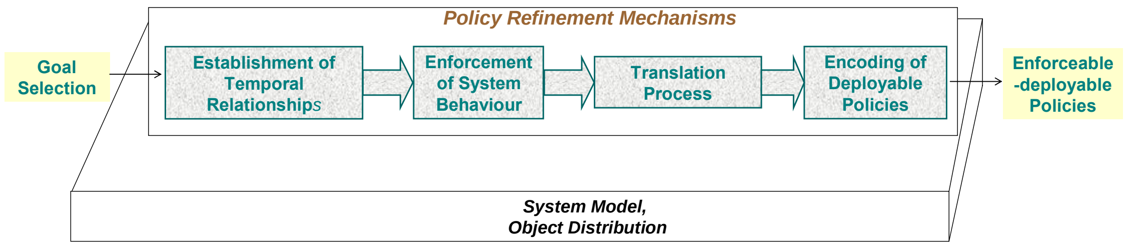

Policy Refinement Mechanisms
Goal Selection
Establishment of Temporal Relationships
Enforcement of System Behaviour
Translation Process
Encoding of Deployable Policies
Enforceable -deployable Policies
System Model, Object Distribution

Figure 19. Policy Refinement Mechanisms

3.2.4.2 Establishment of Temporal Relationships for Refined Goals

The main activities behind this process can be summarised in two: (1) Identification of temporal relationships between refined goals (lowest-level goals), and (2) representation of these temporal relationships by means of suitable formats for use with automated analysis techniques. This approach is similar to the principle of using temporal logics to express search control knowledge for planning techniques [Bac00]. The following is the rationale of these two activities.

For the identification of temporal relationships between refined goals, we use the fact that the selected goals are a sub-set of KAOS goal graph structures and as such, they are intrinsically time-related. Consider for example a parent goal G_1 refined into G_{11} and G_{12} according to the refinement pattern RP3 as shown in Figure 20. For this refinement, the KAOS methodology establishes the temporal prescriptions of the parent goal (tp_1) and of its corresponding refinements (tp_{11} and tp_{12}) as follows:

tp_1, formally expressed as $P \rightarrow \Diamond Q$, identifies that *"under the occurrence of a state P, the state Q must eventually be reached"*.
tp_{11}, formally expressed as $P \rightarrow \Diamond R$, identifies that *"under the occurrence of a state P, the state R must eventually be reached"*.
tp_{12}, formally expressed as $R \rightarrow \Diamond Q$, identifies that *"under the occurrence of a state R, the state Q must eventually be reached"*.

Central to our study are the Temporal Relationships (TR) between the goal refinements: in this particular example between G_{11} and G_{12}. In a strict temporal ordering of properties, the temporal prescriptions of these refinements (tp_{11} and tp_{12}) suggest that property P should hold before property R from which property Q should eventually hold. In other words, this ordering of properties implies that G_{12} should be fulfilled after G_{11}. The latter is in fact a temporal relationship between these two goal refinements. Figure 20 shows the Temporal Relationship TR_1 which formally states that *"on the occurrence of a state P, an intermediate state R must first be reached from which a goal state Q will eventually be reached"*.

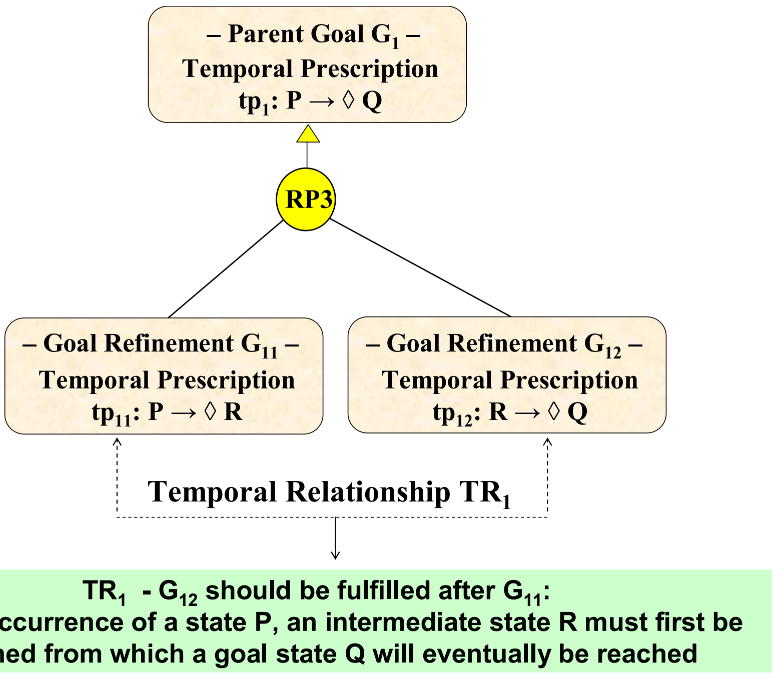

– Parent Goal G_1 –
Temporal Prescription
$tp_1: P \rightarrow \Diamond Q$
RP3
– Goal Refinement G_{11} –
Temporal Prescription
$tp_{11}: P \rightarrow \Diamond R$
– Goal Refinement G_{12} –
Temporal Prescription
$tp_{12}: R \rightarrow \Diamond Q$
Temporal Relationship TR_1
TR_1 - G_{12} should be fulfilled after G_{11}:
On the occurrence of a state P, an intermediate state R must first be reached from which a goal state Q will eventually be reached

Figure 20. Temporal Relationships of Goal Refinements

Having identified the temporal relationships between goal refinements, the second key aspect is the characterisation of these temporal relationships with formal representations that are suitable for use with automated analysis techniques. In other words, we need a formalism to express the ordering of events in time. To this aim we have followed the principles of Finite-state specification patterns [Dwy98]. It is worth noticing that these patterns are different from the patterns used to refine goals. The Finite-state specification patterns are used to specify behavioural properties of reactive systems.

Finite-state specification patterns are classified in two main groups [Dwy98]: Occurrence Patterns and Order Patterns. While Occurrence Patterns are used to identify behaviours in which a specific state/event takes place, Order Patterns deal with prescribed behavioural arrangement of states/events. A complete classification of these patterns is shown in the upper-left part of Figure 21.

Occurrence Patterns are classified in *Existence, Absence, Universality* and *Bounded Existence*. Existence patterns should be used in cases where the most important is to specify that a state/event occurs. Absence patterns should be used when it is necessary to specify that a state/event does not occur. Universality patterns are meant to specify that states/events occur always throughout a given scope. Bounded Existence patterns should be used in situations where it is important to specify the number of times a state/event occurs within a scope.

Order Patterns are classified in *Response* and *Precedence*. Response patterns are used to represent constraints in the order of states/events. Precedence patterns are concerned with the specification that a given state/event *P* must be always preceded by a state/event *Q* within a scope.

In this context, a scope represents a restriction on the time a given property must hold. For example, with regard to an Existence Pattern, a scope would be used to define whether the occurrence of a state/event holds either globally throughout the execution of the system, or before, or after or between other situation(s). The literature [Dwy98] has identified five possible scopes: *Global*, *Before*, *After*, *Between* and *After-until.* A graphical representation of these scopes is shown in the upper-right part of Figure 21.

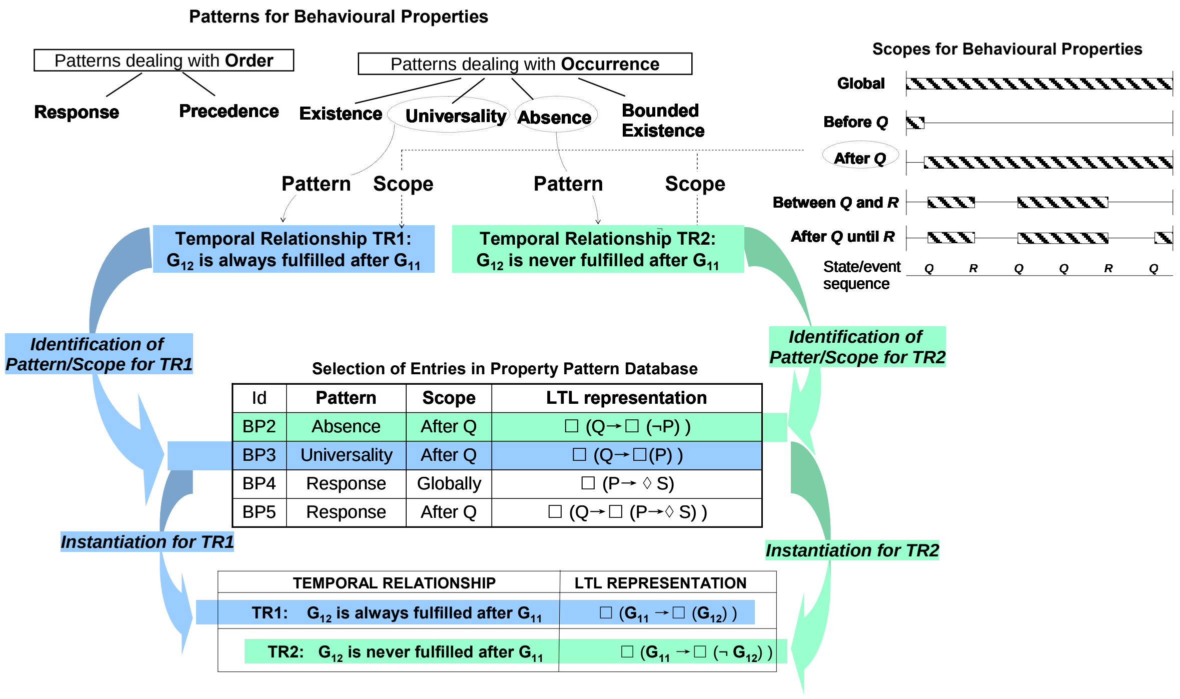
A Global scope prescribes that a pattern holds for the entire system execution; A Before scope is used to indicate that a pattern holds throughout the execution of the system up to a given state/event. The After scope is used in situations where the pattern must hold after a given state/event and throughout the execution of the system. The Between scope helps define behavioural situations in which a pattern must hold at any part of the execution of the system from a given state/event to another state/event. Finally the scope After-until is like Between with the difference that in the former the designated part of the execution continues even if the second state/event does not occur.

The study of the above patterns, scopes and their representation in different logics has been the subject of research for some time [Dwy98]. Thousands of combinations of pattern/scopes have been identified in the literature and practical approaches have been proposed to elaborate databases that classify combinations of patterns/scopes [Dwy]. The classification of pattern/scopes with their corresponding logical representations enables to find the formal representation of practically any requirement in a systematic manner. The following presents how these concepts have been laid down in our approach.

Consider G_{11} and G_{12} from our previous Goal Selection example and two temporal relationships as follows:

- TR1: "G_{12} is always fulfilled after G_{11}"
- TR2: "G_{12} is never fulfilled after G_{11}"

The Figure 21 shows how the process that formalises the process of specifying TR1 and TR2 making use of the pattern/scope approach described above. By means of behavioural patterns/scopes we could express such temporal relationships as formal specifications of Linear Temporal Logic (LTL) in a systematic manner [Dwy]. As in previous sections we use the classical temporal operators: $\Diamond$ eventually in the future, $\Box$ always in the future, and the classical logic connectors $\wedge$ and, $\vee$ or, $\neg$ not, $\rightarrow$ logical implication, $\leftrightarrow$ equivalence, and so forth.



Patterns for Behavioural Properties
Patterns dealing with Order
Response
Precedence
Patterns dealing with Occurrence
Existence
Universality
Absence
Bounded Existence
Pattern
Scope
Pattern
Scope
Temporal Relationship TR1: G12 is always fulfilled after G11
Temporal Relationship TR2: G12 is never fulfilled after G11
Scopes for Behavioural Properties
Global
Before Q
After Q
Between Q and R
After Q until R
State/event sequence
Q R Q Q R Q
Identification of Pattern/Scope for TR1
Identification of Patter/Scope for TR2
Selection of Entries in Property Pattern Database
Instantiation for TR1
Instantiation for TR2

Id	Pattern	Scope	LTL representation
BP2	Absence	After Q	□ (Q→□ (¬P))
BP3	Universality	After Q	□ (Q→□(P))
BP4	Response	Globally	□ (P→ ◊ S)
BP5	Response	After Q	□ (Q→□ (P→◊ S))

TEMPORAL RELATIONSHIP	LTL REPRESENTATION
TR1: G_{12} is always fulfilled after G_{11}	$\Box (G_{11} \rightarrow \Box (G_{12}))$
TR2: G_{12} is never fulfilled after G_{11}	$\Box (G_{11} \rightarrow \Box (\neg G_{12}))$

Figure 21. Establishment of Temporal Relationships rationale

The key issue here is then to define which combination of pattern/scope to retrieve from the database of pattern/scopes. Pattern/scopes databases contain entries of pattern/scopes combinations as shown in the *Selection of Entries in Property Pattern Database* in the central part of Figure 21.

- For TR1 we could consider that G_{12} always holds after the fulfillment of G_{11}. This temporal relationship fits into the prescription of a Universally/After Pattern/Scope combination and then we instantiate BP3 to represent TR1 in LTL notations. The LTL representation of TR1 is as follows: $\Box(G_{11} \rightarrow \Box(G_{12}))$
- For TR2 we could consider that G_{12} is always absent after the fulfillment of G_{11}. This temporal relationship fits into the prescription of a Absence/After Pattern/Scope combination and then we instantiate BP2 to represent TR2 in LTL notations. TR2 is represented as follows in LTL: $\Box(G_{11} \rightarrow \Box(\neg\ G_{12}))$.

So far we have described the process to identify temporal relationships between refined goals and the characterisation of these relationships with LTL representations with an illustrative example of two selected goals (G_{11} and G_{11}). Moreover the target of the *Establishment of Temporal Relationships* activity up to the lowest-level goals for which the same principle is applied as described hereafter.

For example, consider the composite Goal Selection shown in Figure 22. Given that G_{11} and G_{12} are not lowest-level goals, the fulfilment of the highest-level goal G_1 should consider not only the temporal relationship TR_1 but also the temporal relationship of the lowest-level goals, TR_{11} and TR_{12}. Consider that the temporal relationship TR_1

prescribes that "G_{12} should be fulfilled after G_{11}"; that TR_{11} prescribes that "G_{112} should be fulfilled after G_{111}"; and that TR_{12} prescribes that "G_{122} should be fulfilled after G_{121}". Under these circumstances, the temporal relationship that characterises the fulfillment of G_1 may prescribe that the lowest-level goals should be fulfilled in the following ordering: G_{111}, G_{112}, G_{121} and G_{122} for which a similar approach may be followed to characterise this temporal relationship with formal LTL representations.

In our approach, by using the KAOS methodology, we establish a hierarchy amongst goals, formalised through goal graph structures. This concept has been crucial to consider the fulfillment of High-level Goals as the combination of the fulfillment of lowest-level goals. Given that lowest-level goals are connected or linked through temporal relationships, we propose viewing the fulfillment of High-level Goals by sequences of lowest-level goals. This KAOS-based approach is different to traditional plan-based techniques in which a goal is identified by mere state predicates [Lam01] or other traditional approaches in which goals have been acknowledged as a set of desirable final states [Bac98]. Our approach takes advantage of temporal relationships to carry out automated analysis techniques.

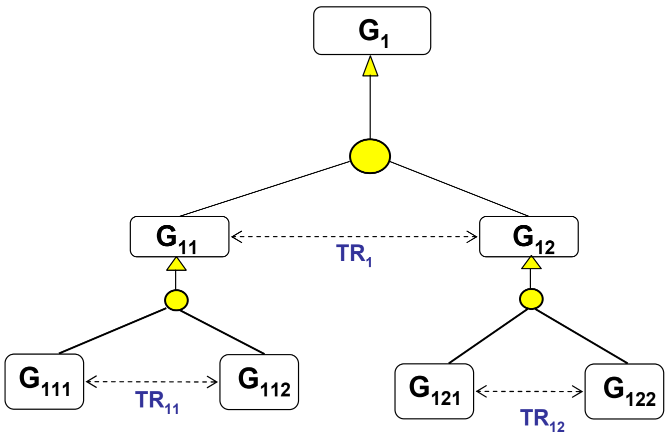


Figure 22. Temporal Relationships for composite Goal Selections

3.2.4.3 Enforcement of System Behaviour

This process is aimed at determining the necessary system behaviour that the underlying managed entities should exhibit as to accomplish with the specifications provided by the "Establishment of Temporal Relationships" process.

This sub-process relies on automated mechanisms grounded in AI planning techniques, particularly through Model Checking [Giu99], [Cim03]. More concretely, as our "Establishment of Temporal Relationships" technique is by means of LTL, we use the concept of Planning as Model Checking with LTL in the context of our refinement framework [Rub05].

In general terms, plans are understood as sequences of actions that lead one system from an initial state to a target state [Giu99]. In the Planning as Model Checking with LTL technique, a goal can be expressed as an LTL formula and LTL Model Checking is used to determine plans that indicate how such system should behave for satisfying the goal [Kab97], [Bac98], [Thi06]. The Figure 23 shows a graphical representation of how we have laid down this concept in our policy refinement framework for which we provide an explanation thereafter.

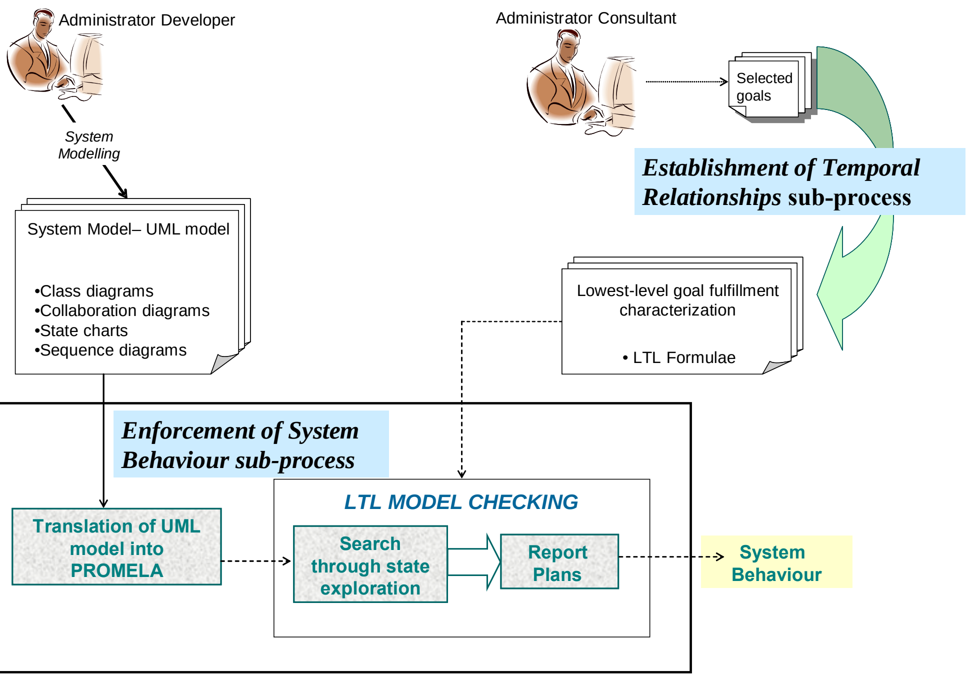


Figure 23. The Planning as Model Checking approach in our framework

As a Model Checking-based approach, the Enforcement of System Behaviour process has two inputs; namely the System Model and the established temporal relationships amongst lowest-level goals, i.e. the output of the process *Establishment of Temporal Relationships*. Regarding the System Model, we consider that the Administrator Developer uses for this purpose standard UML notations such as class diagrams, collaboration diagrams, state charts and sequence diagrams. Given that the main process relies on LTL-based state exploration via Model Checking, this process includes a translation mechanism of the UML models into PROMELA, the language interpreted by the SPIN Model Checking engine, as shown in Figure 23.

The SPIN searching engine explores the state space finding the necessary transitions the system must take to reach the sequence of states as prescribed by the LTL formulae (see *Search through state exploration* Fig. 23). The SPIN searching engine then provides a plan of transitions to be executed by the model entities. The process ends verifying that the plan includes the states prescribed by the lowest-level goals and that these states are reported in the ordering prescribed by the LTL formulae (see *Report*

Plans in Fig. 23). Figure 24 shows a graphical representation of a plan obtained in the *Enforcement of System Behaviour* sub-process` which in turn corresponds to a SPIN system trace report.

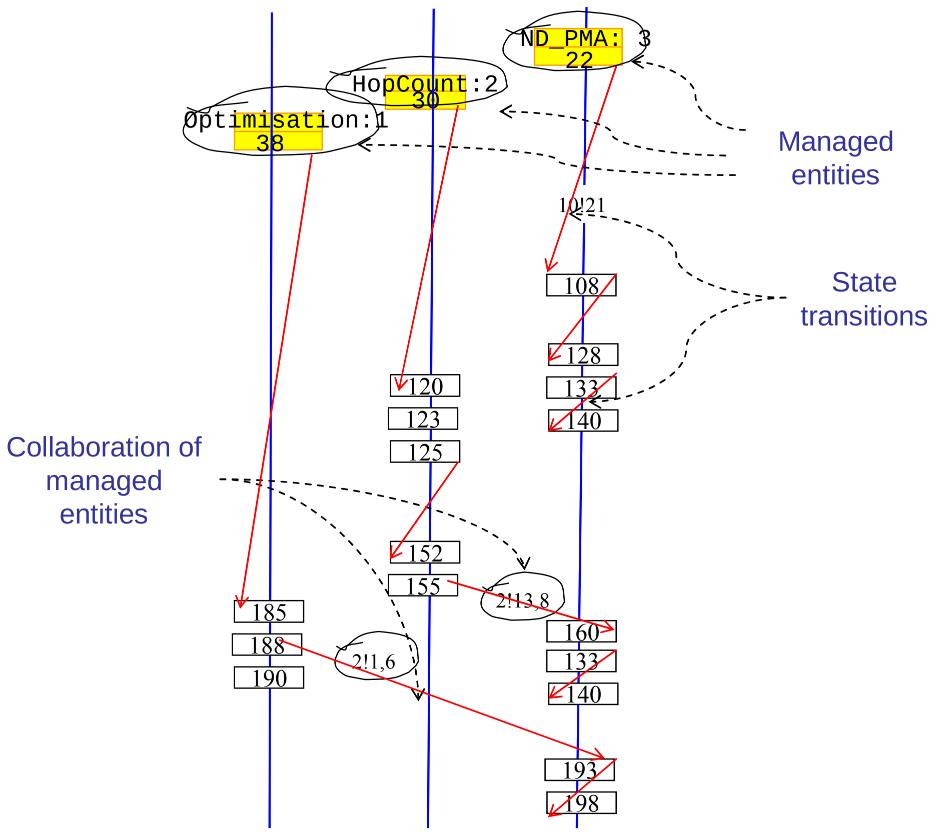


Figure 24. An output of the Enforcement of System Behaviour process

The system trace executions provided by the SPIN searching engine allow to identify the managed entities in charge of executing the plan of actions (see *Managed entities pointers* in Figure 24). In addition, the traces allow to identify the state transitions, at every point of the execution, that the corresponding Managed entities should take during the plan (see *State transition pointers* in Fig 24). In addition, in the context of our refinement framework, plans satisfying the established temporal relationships of lowest-level goals may require the action of several model entities and, more important, their collaboration. The plans reported by the SPIN searching engine allow identifying how the model entities collaborate during state transitions. This situation occurs for example when an entity transitions from one state to another as a result of the reception of a notification from another entity as illustrated by the pointer *Collaboration of model entities* in Figure 24.

The plans provided by the *Enforcement of System Behaviour* process indicate the behaviour that the managed entities should exhibit in runtime to fulfill the High-level Goals i.e. the goal selection. Since the plans include every transition at every point of the execution, the Model Checking reports do not identify the state transitions that should be controlled by policies from those inherent of the system functioning. As a consequence, it is impossible to find meaningful policy information from the execution reports produced by SPIN. Therefore, further analysis is necessary to acquire meaningful policy information from the reported trace execution. The following section introduces the concept of Translation Process [Rub06a], [Rub06b] that we have developed on purpose to abstract policy information from system trace executions.

3.2.4.4 Translation Process

This process is intended to identify the policies that would be necessary to reproduce the system trace execution reported by the *Enforcement of System Behaviour* process. This process relies on the principle that policies control system state transitions [Str04] and that the reported plans above mentioned include, among others, the policy-controlled state transitions. A Translation Process [Rub06a], [Rub06b] is used to abstract the policy-related information from policy-controlled transitions. In this sense, it is mandatory that the developer provides the state transitions that are controlled by policies during the design/development of the system.

The first step towards the application of Translation Process is the identification of *transition plans*. A *transition plan* (TP) is a sub-section of a system trace execution that includes a policy-controlled state transition and that is characterised as follows:
Be

- PSi a pre-condition in a managed entity S
- PQi a pre-condition in a managed entity Q
- $T_{Si,Si+1}$ a state transition in the managed entity *S*
- $T_{Qi,Qi+1}$ a policy-controlled state transition in the managed entity *Q* as a result of transition $T_{Si,Si+1}$

Then, the transition plan TP is represented as $TP=[PS_i,T_{Si,Si+1} => PQ_i,T_{Qi,Qi+1}]$ and its semantics is: "on the occurrence of PSi in the managed object S, preceding the transition $T_{Si,Si+1}$, the managed object Q must enforce the transition $T_{Qi,Qi+1}$". A graphical representation of a transition plan TP is illustrated in the left part of Figure 25.

The Translation Process materialise the above prescription with Event-Condition-Action (ECA) policies. For this, we have used the Ponder obligation policy structure [Dam01]. Obligation policies are event-triggered Condition-Action rules that define the activities subjects must perform on objects in a target domain.

In the context of our refinement approach we consider that a policy-controlled transition is the result of an action enforcement within the target domain. Hence, transitions are interpreted as actions executed on a managed object. A graphical representation of the Translation Process is shown in the right part of Figure 25. This mapping is done following the Ponder prescription that Obligation policies are event triggered and that define the actions that subjects must perform on objects of the target domain.

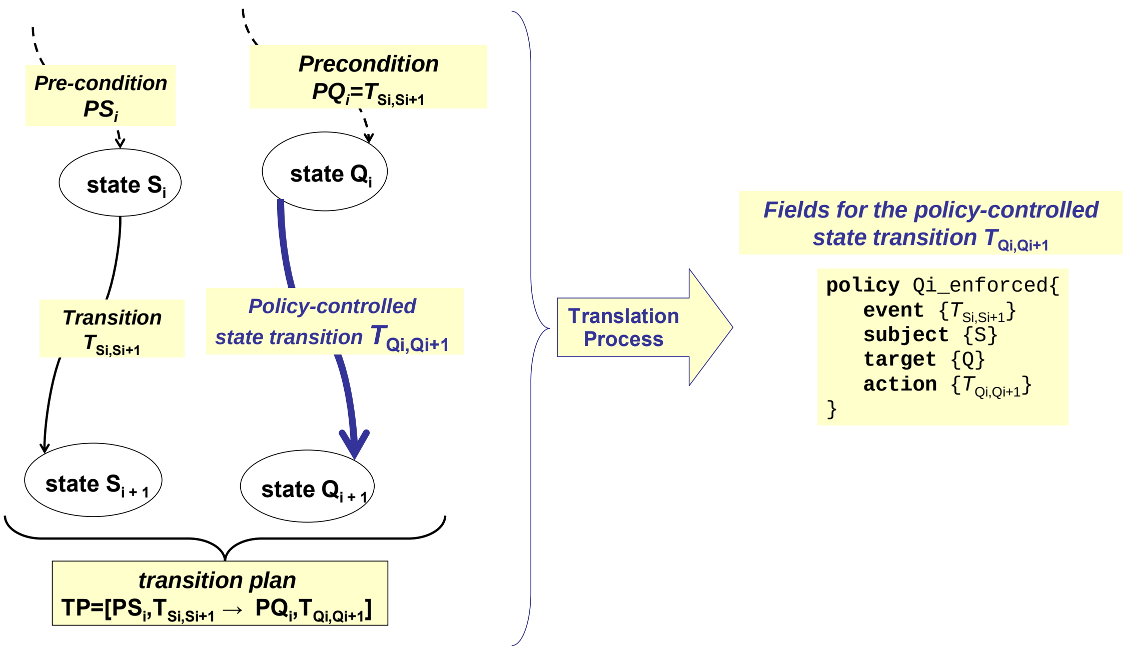

Pre-condition PS_i
Precondition $PQ_i=T_{Si,Si+1}$
state S_i
state Q_i
Transition $T_{Si,Si+1}$
Policy-controlled state transition $T_{Qi,Qi+1}$
state S_{i+1}
state Q_{i+1}
transition plan TP=[PS_i,$T_{Si,Si+1}$ → PQ_i,$T_{Qi,Qi+1}$]
Translation Process
Fields for the policy-controlled state transition $T_{Qi,Qi+1}$
policy Qi_enforced{
event {$T_{Si,Si+1}$}
subject {S}
target {Q}
action {$T_{Qi,Qi+1}$}
}

Figure 25. Representation of the Translation Process

Typical outputs of the *Enforcement of System Behaviour* process may include more than one policy enforceable transition (e.g. transition $T_{Qi,Qi+1}$ in Fig. 25) and consequently multiple transition plans. The transition plans are mapped to their corresponding policy fields by applying the Translation Process. Consequently, the result of the *Translation Process* sub-process is a set of policy fields (e.g. `policy Q`$_{i\_enforced}$ in Fig. 25), one for each policy-controlled transition.

3.2.4.5 Encoding of Deployable Policies

In order to deploy the abstracted policies onto the managed system, subjects and targets identified above must be matched to the actual object distribution of the system. This is the aim of this final refinement process.

The policy refinement framework considers that the Administrator Developer should document the Object Distribution during system design and development as part of the System Model. The *Encoding of Deployable Policies* process makes use of this information to encode deployable policies from the policy fields provided by the *Translation Process*. Figure 26 shows a graphical representation of final step within the refinement process scenario. This process is achieved in a fully automated manner once the object distribution documentation is available during the operation of the system.

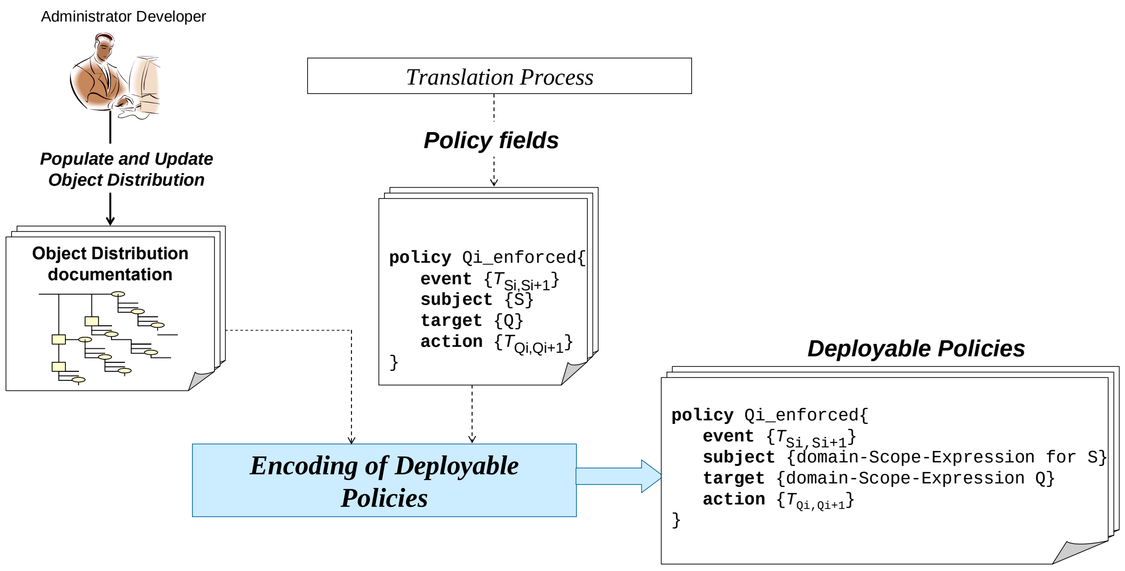


Figure 26. Encode Deployable Policies sub-process within framework

3.3 A Functional Prototype for Policy Refinement

Whereas in Section 3.2 we have described a policy refinement framework detailing its different constituent functional components, this section deals with how to provide an implementable solution [Rub06a] [Rub06b] based on the former.

3.3.1 General outline of the prototype

3.3.1.1 Components of the solution

From a functional point of view, we distinguish three main functions in the refinement process; (1) *Goal Refinement Support*; (2) *Goal Selection*; and (3) *Policy Refinement Mechanisms*. Other supporting functions help in documenting the System Model. Concerning the epoch in the execution of these functions, we differentiate between functions that are carried out during the development/design of the system, and functions that are carried out during the start up/operation of the system.

Our solution has been designed and implemented as a multi-component architecture, in the framework of the DSC platform [Mee00]. A total of seven DSC components have been identified as depicted in Figure 27. The Objectiver Package and the Goal Manager implement the *Goal Refinement Support* and the *Goal Selection Support* functions, whereas the Requirements Manager, the Search Manager and the Policy encoder do the *Policy Refinement Mechanisms*. The Behaviour Manager and the Inventory Manager are supporting components that handle information related to the System Model.

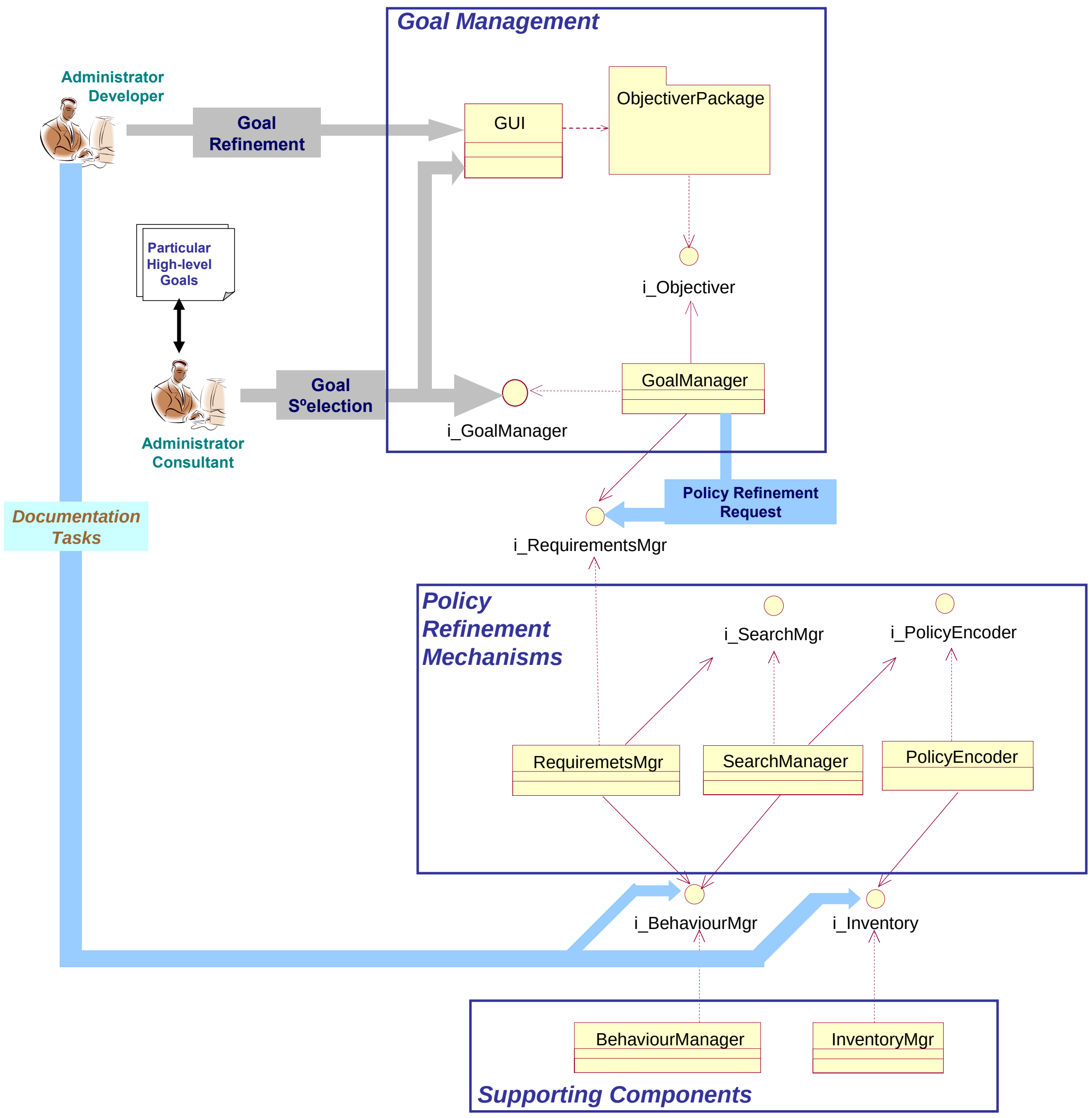


Figure 27. Class Diagram of our Solution

The Figure 28 shows a sequence diagram involving the above mentioned components in a typical refinement scenario that will be referenced in the subsequent component description.

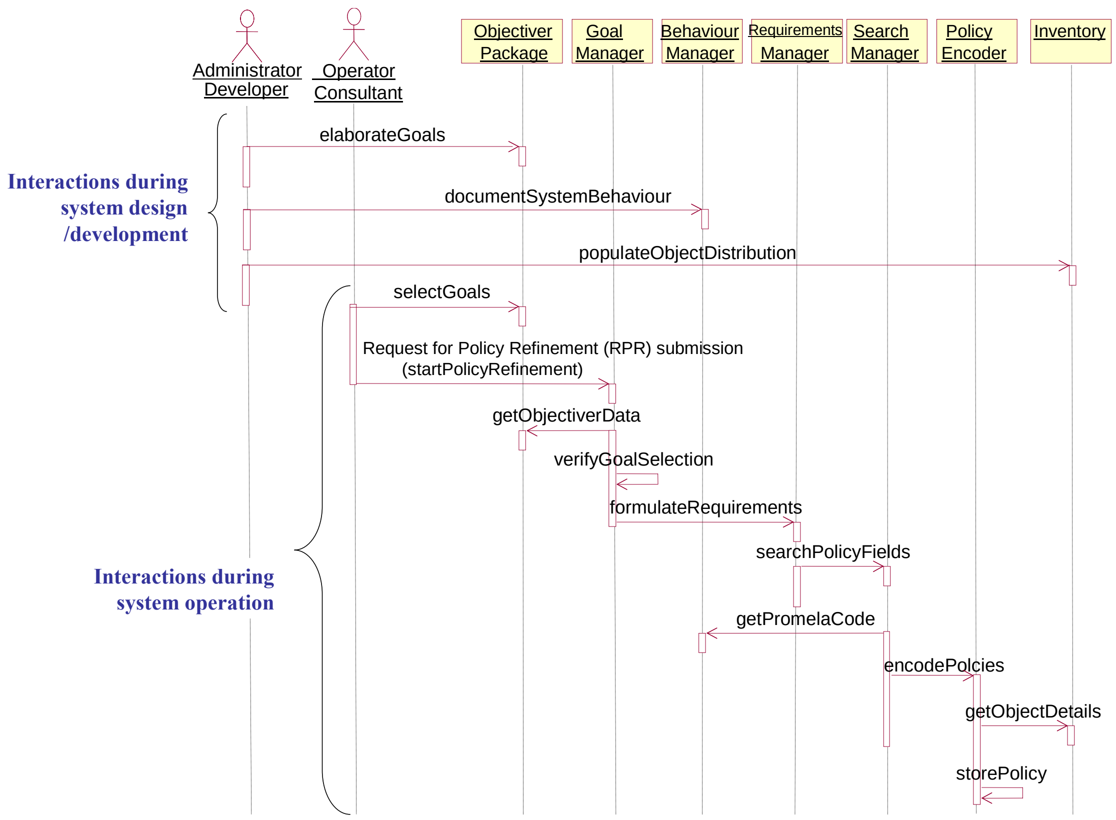


Figure 28. Sequence Diagram of our Solution

3.3.1.2 Goal Management components

The Goal Management components are the **Objectiver Package** and the **Goal Manager**. Figure 29 shows these two components in context.

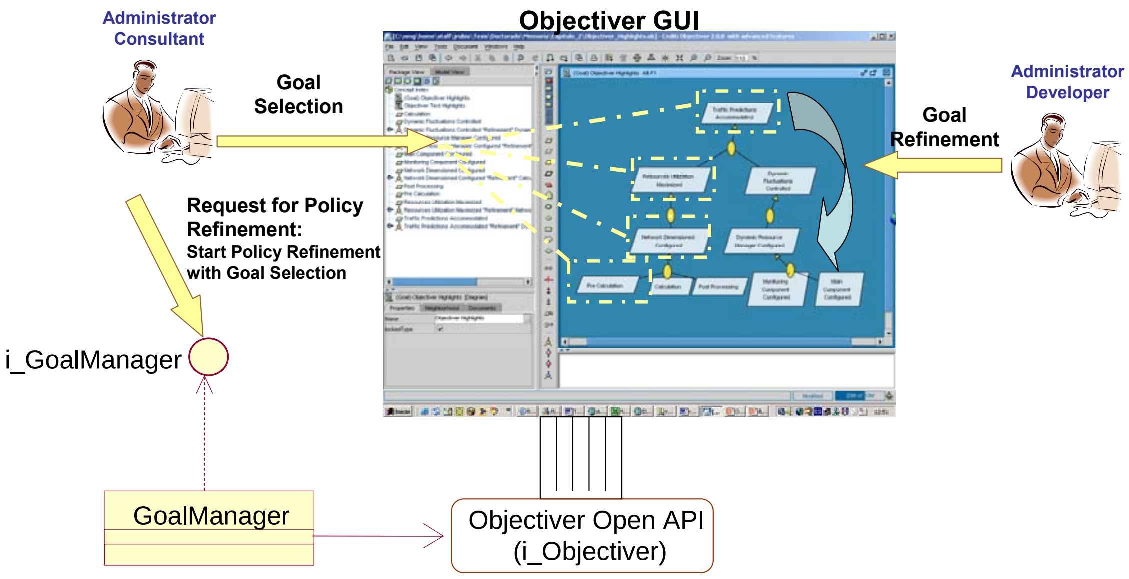


Figure 29. Goal Management Components of the prototype

The **Objectiver package** consists of the Objectiver toolkit [Obj]. The Objectiver toolkit provides a GUI from which the Administrator Developer is enabled to carry out the goal refinement process during the development/design of the system. In this sense, the Objectiver package is used to specialise the *Goal Refinement Support* function of the refinement process (elaborateGoals interaction in Fig. 28).

During the operation of the system, the Objectiver GUI is used to define the administrative guidelines that the Consultant wants to fulfill with the policies. For this, we take advantage of the Objectiver explorer facility as the means to browse through the goal libraries and to select the strategies that better reflect a given administrative view. In this sense, the Objectiver package is used to carry out the *Goal Selection Support* function of the refinement process (selectGoals interaction in Fig. 28).

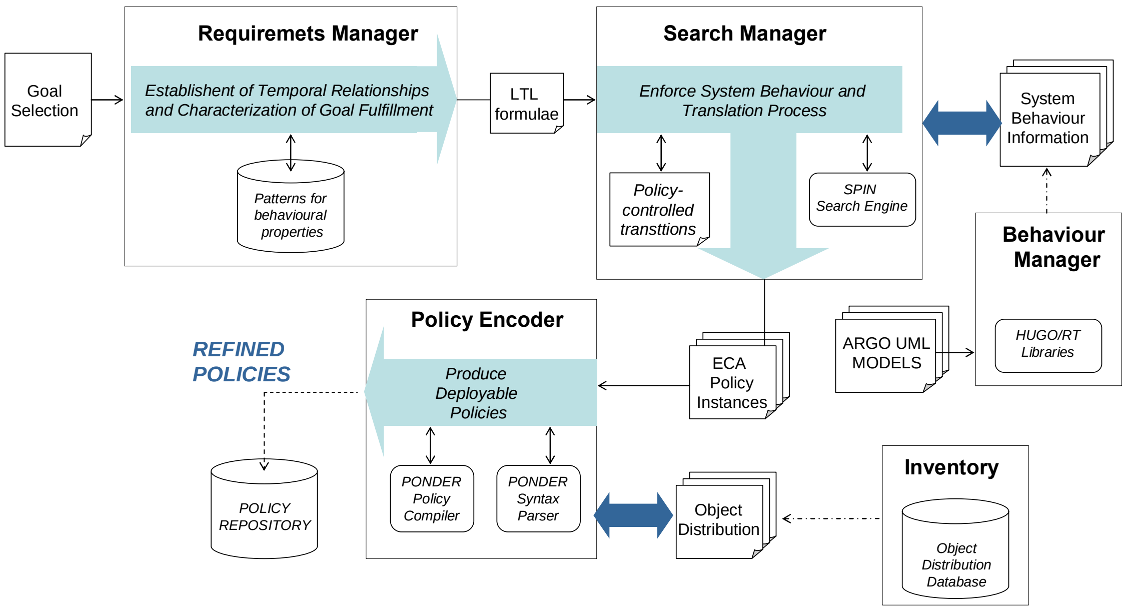
Once the Consultant has selected the goals that better reflect his/her administrative criteria, the **Goal Manager** receives the order to start the policy refinement process from the Goal Selection (startPolicyRefinement interaction in Fig 28). This order will be called "Request for Policy Refinement (RPR) submission" in the remaining of the Thesis. It is worth mentioning that for large-scale refinements, the prototype can handle multiple RPR submissions at a time.

When the Goal Manager receives a RPR, it queries the relevant information affecting the Goal Selection from the goal database in the Objectiver package (getObjectiverData interaction in Fig 28). The aim is to make sure that the Goal Selection is logically complete in the sense that the selected goals entail the higher-level goal fulfillment (verifyGoalSelection interaction in Fig 28). In order to carry out this analysis the Goal Manager retrieves information like goal names, temporal prescription of goals, refinement patterns used to decompose each parent goal, etc.

Worthy to mention is the Open API that has enabled the use of Objectiver as a server within our prototype. In fact, the Goal Manager implements the Objectiver Open API to perform queries to the Objectiver toolkit.

3.3.1.3 Components specialising the Policy Refinement Mechanisms

The *Policy Refinement Mechanisms* are specialised by the Requirements Manager, the Search Manager and Policy encoder components. The target of these components is to produce deployable/enforceable policies in a fully automated manner from the Goal Selection produced by the Goal Manager. Figure 30 shows the layout of these components in the context of our policy refinement framework. The figure also shows the supporting components that make possible the automation of the Policy Refinement Mechanisms, namely the Behaviour Manager and the Inventory components.



Goal Selection
Requiremets Manager
Establishent of Temporal Relationships and Characterization of Goal Fulfillment
Patterns for behavioural properties
LTL formulae
Search Manager
Enforce System Behaviour and Translation Process
Policy-controlled transttions
SPIN Search Engine
System Behaviour Information
Behaviour Manager
HUGO/RT Libraries
ARGO UML MODELS
ECA Policy Instances
Policy Encoder
Produce Deployable Policies
PONDER Policy Compiler
PONDER Syntax Parser
REFINED POLICIES
POLICY REPOSITORY
Object Distribution
Inventory
Object Distribution Database

Figure 30. Components specialising the Policy Refinement Mechanisms

- The **Requirements Manager** implements the *Establishment of Temporal Relationships* process. It receives the Goal Selection provided by the Goal Manager (formulateRequirements interaction in Fig 28). This component is aimed at producing the LTL formulae that characterise goal fulfillment. Generally speaking, this component implements algorithms that first abstract the temporal relationships between lowest-level goals and then characterises those relationships by means of LTL formal representations. Internally it implements a database of patterns for finite-state verification that allows producing LTL formulae from specific requirements in a fully automatic manner.
- The **Search Manager** is in charge of producing policy fields from the LTL formulae (searchPolicyFields interaction in Fig 28). It implements the *Enforcement of System Behaviour* and *Application of Translation Process* mechanisms of our framework. For this purpose it integrates a SPIN [Hol04] search engine to acquire the system trace executions that commit with the LTL goal characterisation. This component uses the PROMELA code (getPromelaCode interaction in Fig. 28), generated by the support component Behaviour Manager (described below). In order to produce Event-Condition-Action (ECA) policies automatically, this component implements an algorithm that applies Translation Process to the system trace executions. For this, the Search Manager handles the policy-controlled transitions documented through the Behaviour Manager.
- The **Policy encoder** implements the *Encoding of Deployable Policies* process of the overall policy refinement process (encodePolicies interaction in Fig. 28). It follows the syntax of the Ponder specification language [Dam02] to encode ECA deployable policies committing to the actual object distribution (getObjectDetails interaction in Figure 28). This component includes a Ponder syntax parser and an adapted Ponder compiler to automate the compilation process.

- The **Behaviour Manager** is a supporting component that handles the System Model with regard to the behaviour of the managed objects. It translates UML specifications into PROMELA code (the language used by SPIN) and manages this information during the refinement process, i.e. it povides information related to the system specification (getPromelaCode interaction of Fig. 28). This component provides the means to document the behaviour of the managed objects of the System Model with UML representations as active classes, state machines, collaborations and interactions (documentSystemBehaviour interaction in Fig. 28). The Behaviour Manager implements the libraries provided by HUGO/RT [Bal04]. HUGO/RT is a UML model translator that allows translating UML models into code for SPIN and other off-the-shelf tools.
- The **Inventory Manager** is also a supporting component that handles the object distribution of the actual managed objects of the System Model. A well-defined structure of this information is necessary to automate the policy refinement mechanisms. The Inventory implements a database for the Administrator Developer to document this information during the design of the system (populateObjectDistribution in Fig. 28). During system operation the latter information is made available to other components like for instance to the Policy Encoder.

Our policy refinement prototype has been implemented as a distributed environment [Rub06a], [Rub06b] making use of the Distributed Software Component (DSC) development framework [Mee00]. The DSC SDK is a development environment for building CORBA (Common Object Request Broker Architecture)-based, distributed component applications. It includes useful tools for developing and testing distributed applications. A brief description of the capabilities of the DSC toolkit can be found in the Appendix B. Also, a detailed description of our prototype can be found in Appendix C.

3.4 Conclusions

This Chapter has provided two main contributions to the policy refinement area. (1) we have provided a holistic approach to policy refinement grounded in requirements engineering and planning by means of Model Checking principles [Rub05], and (2) we have materialised the concept of the proposed approach with a novel and practical solution that makes it possible to address the policy refinement problem [Rub06a],[Rub06b]. The peculiarities of the assessments of these two contributions and some future directions are provided hereafter.

The crucial aspects of our approach are the following three key issues:

- The representation of the aims of policies at different levels of abstraction to ground the policy refinement process.
- The definition of mechanisms that make possible the production of enforceable/deployable policies from high-level abstract requirements.
- The identification of the administrative tasks that should be carried out throughout the life cycle of a policy-based system.

For the first key issue we have borrowed the idea of considering the achievements of policies as goals [Lam99], [Ban04] and therefore we can make use of Requirements Engineering techniques [Dar96] to formalise the representation of the aims of policies at different levels of abstraction. Moreover, we have adopted the KAOS goal-oriented methodology [Dar96] and made use of its technical foundations to ground the analysis techniques on which our refinement approach relies.

The second key aspect of the refinement framework overcomes the limitations of the adopted goal elaboration method such as the lack of support to relate goal fulfillment to system's behaviour and consequently with the abstraction of policies that make possible such a goal fulfillment. For this purpose we have taken advantage of the logical foundations of the KAOS methodology, which is grounded in Linear Temporal Logic, combined with the use of LTL-based state exploration through Model Checking. So far, these logical foundations have not been exploited in policy refinement contexts and consequently their potential to systematise the policy refinement process have remained unexploited or unconsidered. Specifically, the contributions can be stated as follows:

- We have introduced the use of specification patterns for behavioural properties as the means to represent the fulfillment of High-level Goals, namely establishing temporal relationships amongst lowest-level goals. This has allowed us to represent goal fulfillment with formal notations. Given that our analysis technique relies on the Planning as Model Checking approach, we have used Linear Temporal Logic notations to specify the Establishment of Temporal Relationships. It must be realised that without formal relationships between goals at different levels of abstraction, it would be practically impossible to systematise the policy refinement process.
- We have introduced the planning as Model Checking approach as the means to acquire the behaviour that a managed system should exhibit as to fulfill High-level Goals. In addition, we have provided the methods that should be considered when adopting Model Checking searching engines, as the means to systematise the abstraction of policy-aware information from runtime system execution traces. We have

presented the description and the considerations to produce enforceable/deployable policies from abstract requirements making use of the above techniques.

With regard to the third key aspect of the refinement framework, we have clarified the nature of the activities that the administrative parties should carry during the life cycle of the policy system. It is imperative that, during the design/development of the system, the administrative parties should document the System Model and assess the Goal Refinement process. In this approach, there is no means to delegate these two activities on automated processes and hence human intervention is mandatory. During the start up/operation of the system, through Goal Selection, the framework provides the means to abstract automatically policies aligned to high-level administrative guidelines.

Work done so far is open ended. We outline some of the most relevant directions that could be explored with regard to the framework presented in this Chapter.

Our policy refinement process has been limited to consider exclusively Achieve goals, thus leading to relate the concept of achievable goals with obligation policies. Future work could be directed to explore the usefulness and implications of other goals supported by the KAOS methodology like Cease, Maintain and Avoid goals. We envisage that the consideration of these types of goals may enable to carry out analysis for application domains in which authorisation, access control and security issues have special relevance.

A key concept that has enabled us to systematise the policy refinement process is the Translation Process [Rub06b] concept by means of which we derive policies from system state transitions. So far we have considered unconditional executions of actions, namely we have considered that all states and transitions are permitted in a policy-based system. Future work could be directed to consider non permitted states likely dictated by Maintain/Avoid goals. In this sense, our Translation Process could be extended to consider the generation of actions based for example on guard conditions. On the other hand, so far we have considered only situations where the Model Checking approach reports single system behaviour traces. Nevertheless, considering multiple traces would allow choosing between different transition plans, possibly for different conditions and hence to encode policies for different sets of conditions.

The policy refinement framework described in this Chapter provides the means to carry out analysis to bridge the gap between High-level Goal fulfillment and the acquisition of enforceable/deployable policies. Moreover we are still far away from proposing a generic solution that bridges the gap between SLA fulfillment and the formulation of High-level Goals. This is a critical and challenging issue that may possibly imply to adapt other mechanisms to relate service management, system performance, goal specialisation and feedback mechanisms for particular application domains.

Chapter 4 A Systematic Approach to Goal Refinement

4.1 Introduction

In the previous chapter we have presented the decomposition of system goals at different levels of abstraction. As goals can be seen as the aims to be fulfilled by policies, the above implies a mechanism to find enforceable policies that finally should accomplish the system goals. In addition we have also proposed the steps of a systematic refinement process.

The main motivation of this Chapter is to provide a self-contained example of how the goal refinement process can be assessed in management environments. Our claim is that the framework and the techniques we use are application-domain independent. Therefore, the above mentioned example is illustrating a systematic approach to goal refinement. The key concept consists of driving the goal refinement process by the composition hierarchy of the managed system.

This Chapter is divided in four parts. After this Introduction, Section 4.2 provides the principles to define a hierarchical composition of the managed system; Section 4.3 describes how to assess goal refinement based on the hierarchical composition of the system. Finally Section 4.4 provides the conclusions on this systematic approach.

4.2 Identifying the System Composition Hierarchy

We consider a management system as providing a service (or even a set of services) and that the management system relies on system components supporting this service. Then we propose to drive the goal refinement process making use of the managed system composition. The idea behind the use of this service decomposition is to allow the establishment of a parallelism between the component system functions and the goals coming out of a refinement cycle process. In other words, it seems reasonable to decompose a goal in sub-goals according to the system composition architecture. To better describe this approach we will provide a running example that is formulated based on the principles developed in the context of the IST project WINMAN – WDM and IP Network Management [Kar05]. A High-level architecture of the WINMAN system is shown in Figure 31 for which a brief description is provided thereafter.

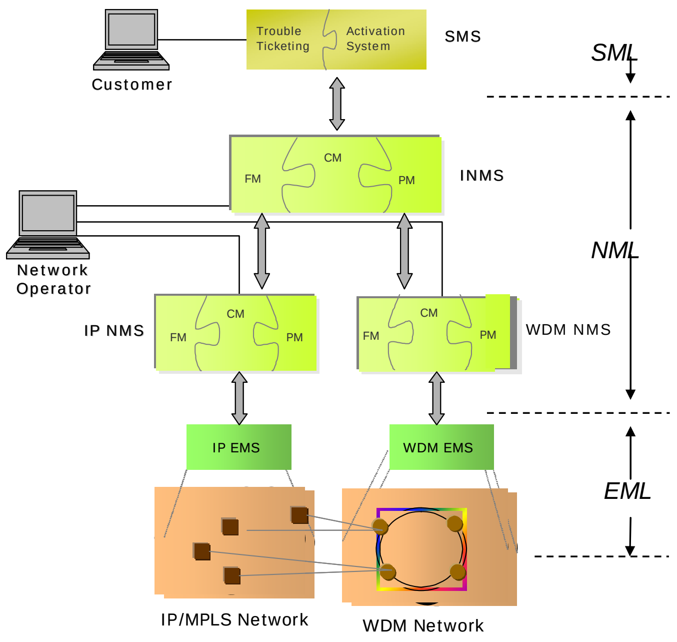


Figure 31. WINMAN high-level architecture

WINMAN is a management-plane-based solution for the provisioning of IP over WDM connectivity services with guaranteed quality of service. The WINMAN architecture adopts a multilayer model spanning different management functional areas. The service management layer (SML) supports the visibility of only service-related managed objects (e.g., service-related connection termination points). In the network management layer (NML) the system view consist of connections with characteristic QoS parameters. Finally, in the element management layer (EML), equipment-related information (e.g., ports, cards) is represented. WINMAN focuses on the network management layer (NML), covering three management functional areas: configuration management (CM), fault management (FM), and performance management (PM). The NML is further subdivided in two sub-layers, one being the integrated or inter-technology network management sub-layer, with the corresponding network management system (INMS), and the other being the technology-dependent sub-layers, comprising the IP network management system (IP NMS) and WDM network management system (WDM NMS). Each of these NMSs consists of the specialization of a generic network management system, performing configuration, fault and performance management tasks accordingly.

Figure 32 shows a generic system composition hierarchy and an instantiation for the WINMAN management system. The following are the proposed guidelines to define the composition hierarchy shown in the aforementioned figure:

1.- Identification of the service provided by the managed system. Considering that any management system provides a service, the top level of the system composition hierarchy will be represented by the service that such system provides. In the left part of Figure 32 we identify this highest-level of the system composition hierarchy as “Service to be provided”. For example, an IP over WDM connectivity service is the basic service provisioned by the WINMAN system. The highest-level composition of the WINMAN system will then be defined as “IP over WDM Connectivity Service”.

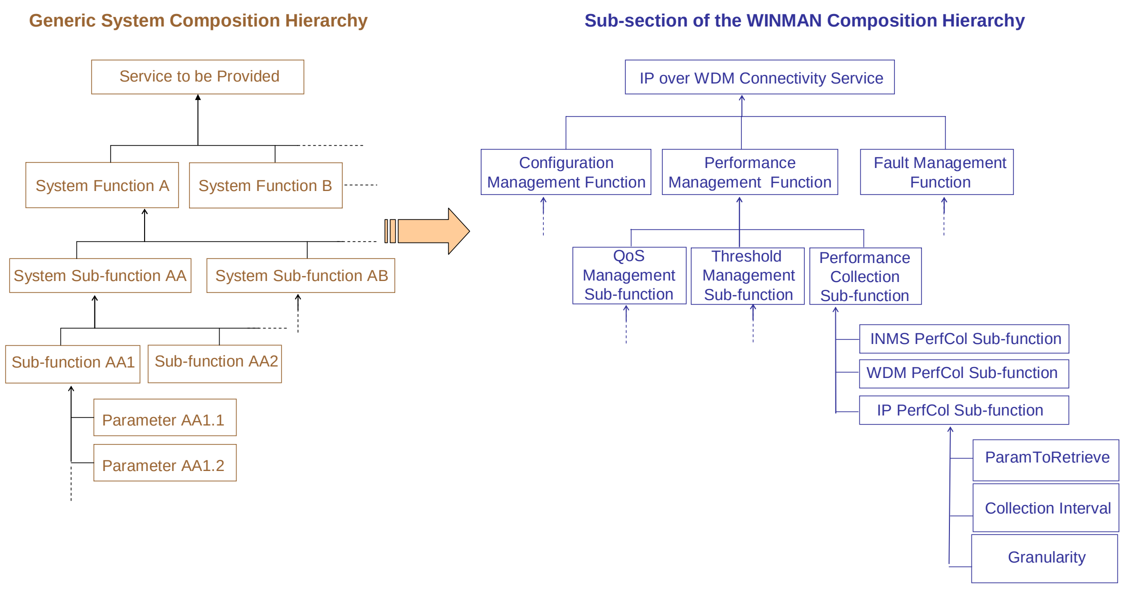

Generic System Composition Hierarchy
Service to be Provided
System Function A
System Function B
System Sub-function AA
System Sub-function AB
Sub-function AA1
Sub-function AA2
Parameter AA1.1
Parameter AA1.2
Sub-section of the WINMAN Composition Hierarchy
IP over WDM Connectivity Service
Configuration Management Function
Performance Management Function
Fault Management Function
QoS Management Sub-function
Threshold Management Sub-function
Performance Collection Sub-function
INMS PerfCol Sub-function
WDM PerfCol Sub-function
IP PerfCol Sub-function
ParamToRetrieve
Collection Interval
Granularity

Figure 32. Generic system composition hierarchy and a WINMAN sub-section

2.- Identification of the system functions. Linked to the highest-level of the system composition mentioned above, there would be as many lower-level system functions supporting the "Service to be Provided". In other words, the next level of the system composition hierarchy would be defined by the system functions on which the service provisioning relies. Each function of the system is represented as "System Function" as we show in the left-part of Figure 32. In our example, the WINMAN system implements functions of Configuration, Performance and Fault Management. Consequently, the aforementioned functions would define the "Configuration Management Function", "Performance Management Function" and "Fault Management Function" which are in turn linked to the "IP over WDM Connectivity Service" identified above.

3.- Identification of the system sub-functions. Each function should be linked to the sub-functions that integrate the corresponding system function as we show in Figure 32. The "System Function A" in our composition hierarchy is in turn integrated by "System Sub-function AA", System Sub-function AB, etc. For instance, in the WINMAN example, we show that the Performance Management function is sub-divided in three sub-functions; QoS Managent, Performance Collection, and Threshold Management because has been decided that way by the WINMAN system developer. These three modules will then define the "QoS Management Sub-function", "Performance Collection Sub-function", and "Threshold Management Sub-function" respectively.

This third step should be carried out repetitively in the cases where a System Sub-function is integrated by other sub-functions. The later sub-functions will build up the lower level composition hierarchy of the System Sub-functions identified above, e.g. the "Sub-function AA1" and "Sub-function AA2" represent two sub-functions that compose the "System Sub-function AA" as we show in Figure 32. In our example, WINMAN has been designed to integrate sub-functions well defined by technological domain, namely IP domain, WDM domain, and an Inter-technological Network Management (INMS) domain. Each of these sub-functions will define one level below the corresponding WINMAN System Sub-functions. A more concrete example is the "Performance Collection Sub-function" that is carried out by the "INMS PerfCol Sub-function", "WDM PerfCol Sub-function", and "IP PerfCol Sub-function" as we show in the right-part of Figure 32.

5.- Identify the parameters that influence each Sub-function in a every System Module. This identification step will define the lowest-level of the system composition hierarchy. These parameters will represent the parameters that have some impact on the execution of the corresponding sub-functions. In our system composition hierarchy for example, "Parameter AA1.1" is a parameter that influences the execution of "Sub-function AA1" which in turn is part of the "System Sub-function AA" as we show in the left part of Figure 32. In our WINMAN example, consider the case of the Performance Collection Module in the IP sub-function of the WINMAN approach. Here, the data to collect from the IP network is controlled by three parameters; namely "parameters to retrieve", "time intervals of the collection", and "granularity of the collection". In this case, the "IP PerfCol Sub-function" would yield three lower-level parameters in the system composition hierarchy, namely the "ParamToRetrieve", "Collection Interval", and "Granularity" as we show in the right part of Figure 32.

4.3 Addressing the Goal Refinement Process

We describe the general guidelines to carry out Goal Refinement driven by the hierarchical composition of the system in the context of our policy refinement framework.

4.3.1 Defining High-level Goals

In addition to defining the composition hierarchy of the managed system, another aspect to be addressed is the definition of the High-level Goals that such system can handle. Generally speaking, it is a matter of realistic judgment to decide which High-level Goals should be defined in the target system [Mof93]. Several methods have been proposed to specify High-level Goals and to provide indicators to assess IT performance related to them [Bar06]. Although this is an application-oriented issue, it is imperative to express and represent High-level Goals in a feasible manner.

It is the Administrator Developer who should define the High-level Goals that the system can handle. A key issue during the definition of High-level Goals is the association of High-level Goals with the system functionality. Namely, it is imperative that High-level goals should be aligned with the scope of the target system. For this reason, it is justifiable that the Administrator Developer defines the High-level Goals during the design of the system. In addition, High-level Goals should be consistent with the functionalities designed/implemented by the target system.

Following on with the WINMAN approach, it is comprehensible that the Administrator Developer defines goals that allow him control the way that connectivity services are provisioned with the network managed with WINMAN. High-level Goals will be set in correspondence to the main WINMAN system function (connectivity service provisioning) as well as with the three main sub functions, namely configuration management, fault management and performance management. It is worthy to mention in this example that it would be impossible to think of High-level Goals to commit with billing or pricing of the connectivity services due to its lack of functionalities to handle those High-level Goals.

From the above, three basic High-level Goals for the WINMAN system could be defined as “Connectivity Configuration Set”, “Fault Handling Set”, “Service Performance Set”. While the “Connectivity Configuration Set” High-level Goal sets the ground to control the way that connections are configured in the network, the “Fault Handling Set” High-level Goal is defined to control the way WINMAN should react when faults occur in the underlying network. Finally, the “Service Performance Set” High-level Goal targets the assurance of QoS metrics in the connectivity services provisioned with WINMAN.

4.3.2 Assessing the Goal Refinement Process

Coming back to what we stated before, Goal refinement is an activity designated to elaborate the goal graph structures to which the system must be adhered. These goal-graphs represent both, the requirements and the different options through which they could be achieved. This activity is application-dependent and should be carried out by the Administrator Developer at system design.

4.3.2.1 Starting the refinement process

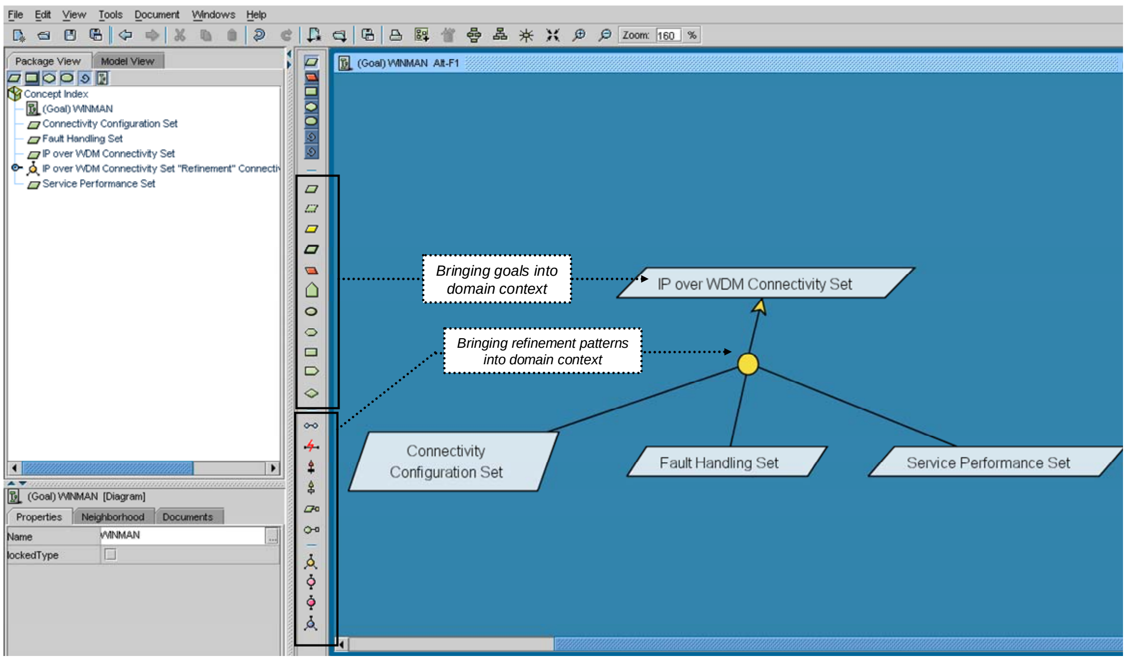
The goal refinement process starts with the definition of the root goal for the goal graph structure. The root goal should be defined taking into account the basic functionality of the target system. As we described earlier, the basic functionality of the system can be controlled with the High-level Goals that the developer has defined. We propose to refine a root goal with the High-level Goals of the system. For this initial step, refinement patterns should be brought into the context of the application domain.

Another way is to skip this initial process and directly start refining each High-level Goal independently and then consider as many goal graph structures as High-level Goals. Nevertheless, we will exemplify the policy refinement process considering root goals. For this, let us consider the WINMAN system introduced in advance.

The root goal could be formalised as “IP over WDM Connectivity Set” since WINMAN is aimed at provisioning IP connectivity services making use of underlying managed sub-systems. This process is graphically illustrated in Figure 33 in which we make use of our prototype to exemplify this initial step of the goal refinement process.

Having defined the root goal “IP over WDM Connectivity Set”, the next step is to bring a refinement pattern into the WINMAN context to refine it. Here, the nature of the system requires that the three functionalities, Configuration, Performance and Fault, should be set to provision WINMAN services. In turn, each of these functionalities is achieved by the corresponding High-level Goals “Connectivity Configuration Set”, "Fault Handling Set", and "Service Performance Set". In order to achieve this, the Administrator Developer brings a multiple-milestone refinement pattern into this particular context as shown in Figure 33 (see note "*Bringing refinement patterns into domain context*").

Consequently, the root goal "IP over WDM Connectivity Set" is refined into the goals "Connectivity Configuration Set", "Fault Handling Set", and "Service Performance Set" altogether with a multiple-milestone refinement pattern. The outcome of this initial process is a goal-graph structure that consists of three sub-trees defined by the High-level Goals of the system as illustrated in the Objectiver GUI of our prototype (Fig. 33).


File Edit View Tools Document Windows Help
Zoom: 160 %
Package View
Model View
Concept Index
(Goal) WINMAN
Connectivity Configuration Set
Fault Handling Set
IP over WDM Connectivity Set
IP over WDM Connectivity Set "Refinement" Connectiv
Service Performance Set
(Goal) WINMAN [Diagram]
Properties
Neighborhood
Documents
(Goal) WINMAN Alt-F1
Bringing goals into domain context
IP over WDM Connectivity Set
Bringing refinement patterns into domain context
Connectivity Configuration Set
Fault Handling Set
Service Performance Set

Name WINMAN
lockedType

Figure 33. Starting the Goal Refinement Process

4.3.2.2 Goal refinement driven by the system composition hierarchy

In this second phase of Goal Refinement, the system composition hierarchy is used to drive the refinement of the High-level Goals that define the trees of the initial goal graph structure. Same as the initial phase, refinement patterns are brought into the context of each application domain to formalise the refinements at each level of the refinement. This process is also exemplified making use of our WINMAN example.

For the "Service Performance Set" High-level Goal, the Administrator Developer should look at the composition hierarchy in that particular node. In this particular example this composition has been defined as "Performance Management Function" which in turn will be used to define the achieve goal "Performance Management Function Set". Consequently, the "Service Performance Set" High-level Goal is refined into the goal "Performance Management Function Set" as we show in Figure 34. The same principle applies to the Configuration and Fault management functions which would yield to the refinements “Configuration Management Function Set” and “Fault Management Function Set” as shown in Figure 34.

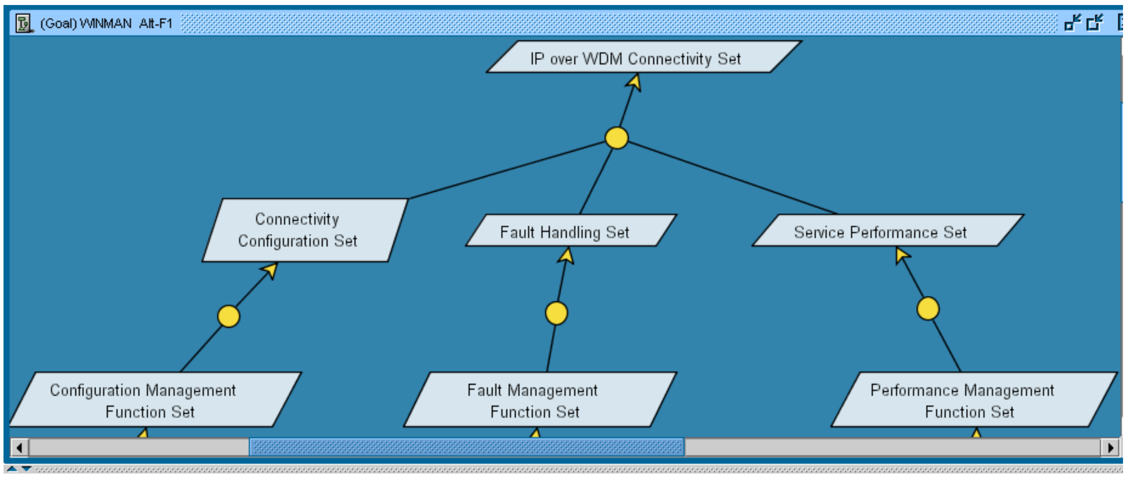

(Goal) WINMAN Alt-F1
IP over WDM Connectivity Set
Connectivity Configuration Set
Fault Handling Set
Service Performance Set
Configuration Management Function Set
Fault Management Function Set
Performance Management Function Set

Figure 34. Refining the High-level Goals of WINMAN

Following on with the sub-tree defined by the “Performance Management Function Set” achieve goal, according to the WINMAN composition hierarchy, the “Performance Management Function” is achieved by the “Performance Collection Sub-function” and other sub-functions. The “Performance Collection Sub-function” is composed by the “INMS PerfCol Sub-function”, “WDM PerfCol Sub-function” and “IP PerfCol Sub-function” (see composition hierarchy in Figure 32). Taking this into account, the goal refinement process should consider that the “Performance Management Function Set” achieve goal would be refined into the achieve goals “Performance Collection Set” as shown in Figure 35, and other goals defined by the rest sub-functions in charge of achieving the Performance Management Function. The later goals are all represented as “Other Requirements for Performance Management” in Figure 35. Going a step further, the “Performance Collection Set” goal would be refined into the “INMS PerfCol Set”, “WDM PerfCol Set”, and “IP PerfCol Set”. For the above refinements the Administrator Developer brings milestone refinement patterns into the context of the WINMAN approach. This level of the refinement process is graphically shown in Figure 35.

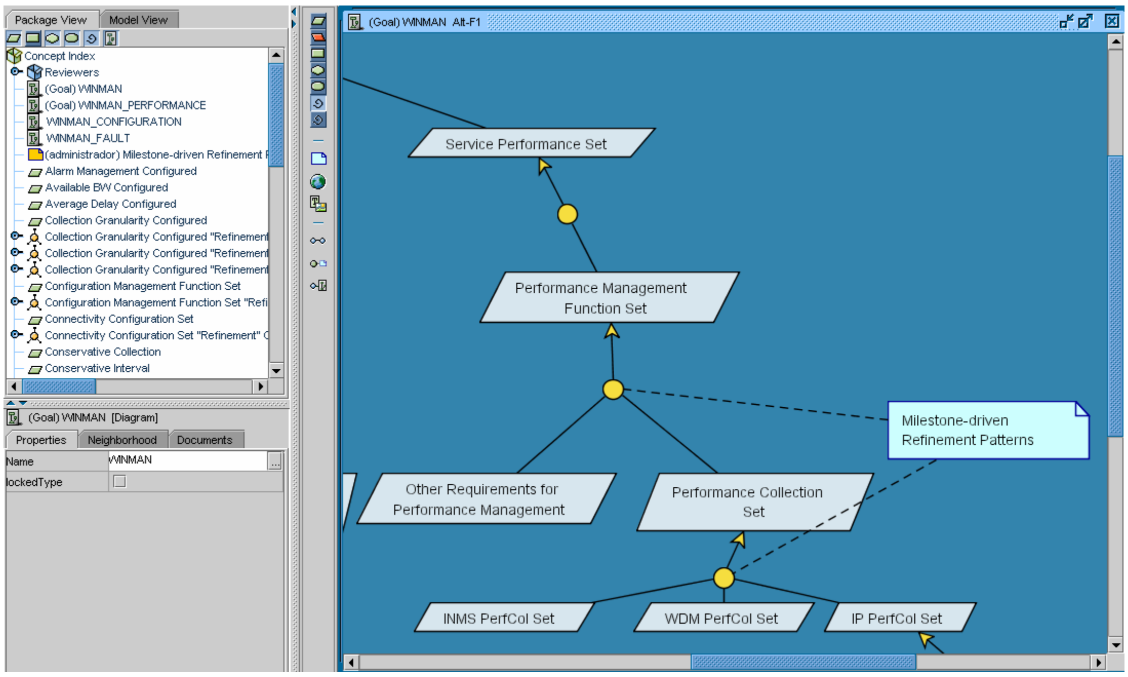

Package View
Model View
Concept Index
Reviewers
(Goal) WINMAN
(Goal) WINMAN_PERFORMANCE
WINMAN_CONFIGURATION
WINMAN_FAULT
(administrador) Milestone-driven Refinement P
Alarm Management Configured
Available BW Configured
Average Delay Configured
Collection Granularity Configured
Collection Granularity Configured "Refinement
Collection Granularity Configured "Refinement
Collection Granularity Configured "Refinement
Configuration Management Function Set
Configuration Management Function Set "Refi
Connectivity Configuration Set
Connectivity Configuration Set "Refinement" C
Conservative Collection
Conservative Interval
(Goal) WINMAN [Diagram]
Properties
Neighborhood
Documents
Name
WINMAN
lockedType
(Goal) WINMAN Alt-F1
Service Performance Set
Performance Management Function Set
Milestone-driven Refinement Patterns
Other Requirements for Performance Management
Performance Collection Set
INMS PerfCol Set
WDM PerfCol Set
IP PerfCol Set

Figure 35. Refining the goal Performance Management Function Set

Going a step further in the goal refinement process guided by the WINMAN composition hierarchy, the “IP PerfCol Set” goal is refined into the goals “Parameters To Retrieve Set”, “Interval Collection Set”, and “Collection Granularity Set”. The Administrator Developer brings milestone refinement patterns into this particular context as we show in Figure 36. Further refinements for these goals would represent the alternatives to fulfill the corresponding goal as we explain hereafter.

The alternatives to achieve the “Parameters To Retrieve Set” achieve goal are, amongst others, “Average Delay Set“, or “E2E Pkt Loss Set“, or “Available BW Set”, or “Explicit Params Set”. These goals are representative of the different alternatives for performance parameters that can be retrieved from the network with WINMAN. The lower part of Figure 36 shows this refinement level together with the resulting refinements for the “Interval Collection Set” goal and the “Collection Granularity Set” goal. In all these cases, the Administrator Developer brings “case-driven” patterns into the context of the WINMAN solution.

The goal refinement process ends when each sub-tree emerging from the root goal “IP over WDM Connectivity Set” is refined up to the lowest level similar as the ones described above. The goal graph structures should be available for further use.

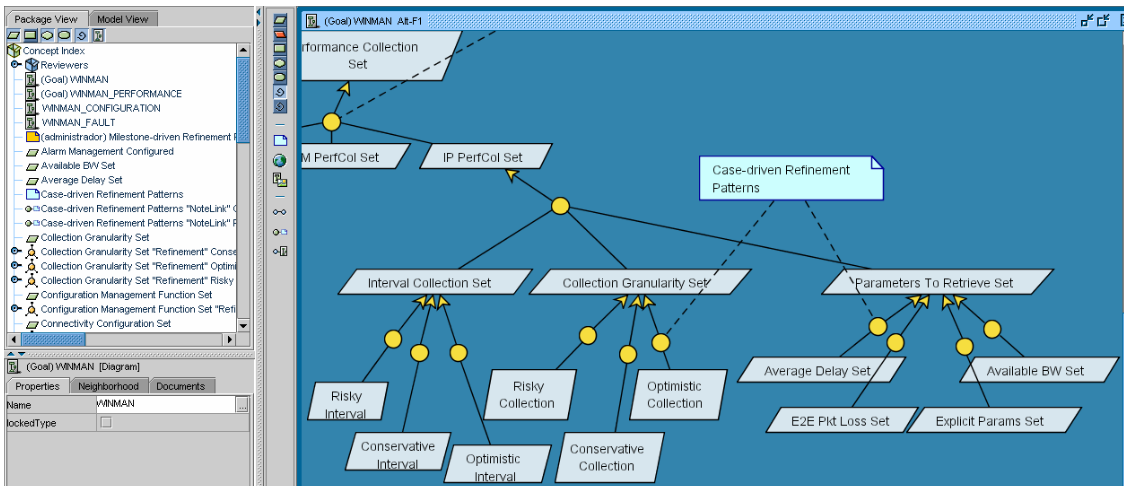

Package View
Model View
Concept Index
Reviewers
(Goal) WINMAN
(Goal) WINMAN_PERFORMANCE
WINMAN_CONFIGURATION
WINMAN_FAULT
(administrador) Milestone-driven Refinement
Alarm Management Configured
Available BW Set
Average Delay Set
Case-driven Refinement Patterns
Case-driven Refinement Patterns "NoteLink"
Case-driven Refinement Patterns "NoteLink"
Collection Granularity Set
Collection Granularity Set "Refinement" Conse
Collection Granularity Set "Refinement" Optimi
Collection Granularity Set "Refinement" Risky
Configuration Management Function Set
Configuration Management Function Set "Refi
Connectivity Configuration Set
(Goal) WINMAN [Diagram]
Properties
Neighborhood
Documents
Name
WINMAN
lockedType
(Goal) WINMAN Alt-F1
rformance Collection Set
M PerfCol Set
IP PerfCol Set
Case-driven Refinement Patterns
Interval Collection Set
Collection Granularity Set
Parameters To Retrieve Set
Risky Interval
Conservative Interval
Optimistic Interval
Risky Collection
Optimistic Collection
Conservative Collection
Average Delay Set
Available BW Set
E2E Pkt Loss Set
Explicit Params Set

Figure 36. Refining the IP PerfCol Set goal of WINMAN

4.4 Conclusions

This Chapter has presented how the high-level goals that a system designer establishes for a policy driven system and the system compositional hierarchy should be used to systematise the goal elaboration process. To the best of our knowledge, no other work has defined generic guidelines to take advantage of hierarchical relationships of policy systems in favour of the policy refinement problem in management contexts. As we described in Chapter 3 the outcome of the goal refinement is essential for the acquisition of enforceable policies from given High-level Goals. Therefore, the contribution of this Chapter is the methodology to address the goal refinement process in management system contexts [Rub06c]. To this aim we considered the following key issues: The definition of the management system hierarchical composition, the definition of high-level goals to which the target system must be adhered and finally, the assessment of the goal refinement process making use of the former.

The outcome of this methodological approach is a goal graph, which integrates goal refinement patterns, that mimics the management system composition hierarchy. In this sense, system compositional hierarchies are essential to formalise the goal refinement process. In addition, it would be very difficult, if not impossible, to achieve systematic goal refinement without specifying High-level Goals and their relationships with the levels of the system compositional hierarchy.

The main advantage of our approach is that it makes use of information and concepts that are necessary for the design and implementation phases of the managed system. In fact, our proposal organises and formalises information to address the policy refinement problem that anyway should be used for system design purposes. The incremental effort for a system designer shouldn’t be significant.

On the other hand, this methodology is clearly application-domain independent and we can claim it as an advantage as well. Nevertheless, it is a fact that applying it to specific refinement problem solving requires a deep knowledge of the target system and/or the application domain. This has been made clear in the above subsections, where each step of the process is driven by decisions that only the system designer can adopt. Anyhow, we consider that this is an affordable price to systematise the goal refinement process and consequently, the policy refinement problem.

Chapter 5 Application Scenario

5.1 Introduction

Policy refinement has been regarded as a crucial process for policy-based management although the research community has remained reticent to analyse its overall implications in complex environments. For instance, previous works presenting models, methodologies, or approaches addressing the refinement problem have not been yet evaluated in complex or real-life environments and as such, their feasibility to address realistic refinement scenarios is questioned.

The main motivation of this Chapter is to provide a complete view [Rub06c] of a refinement scenario in management solutions. For this purpose we deal with the critical nature of addressing the overall implications to refine enforceable policies from abstract requirements intended to manage Quality of Service provisioning.

In this Chapter we address policy refinement for intra-domain Quality of Service Management based on the principles developed in the context of the IST project *TEQUILA - Traffic Engineering for Quality of Service for the Internet at Large Scale* [Tri01]. We demonstrate the feasibility of our refinement framework and our systematic approach with a real and complete scenario applied to this domain. To the best of our knowledge, no other work in the literature has provided a complete refinement scenario applied to real-life management situations.

We initially describe the QoS Management approach on which our scenario relies; we describe the TEQUILA solution which provides an overall architecture for QoS support in IP Networks by bringing Service Management and Traffic Engineering functionalities together in a collaborative environment. For this management solution we apply our systematic approach that consists on the definition of a QoS-aware policy hierarchy and the definition of QoS-aware High-level Goals, both tailored to address policy refinement following the principles developed in our refinement framework. Following on, we provide the execution of our refinement scenario making use of our prototype implementation, and provide the result of its execution.

Due to the broad scope and the complexity of the policy refinement process, this Chapter is sectioned in eight parts: following this Introduction, Section 5.2 provides the rationale of the application domain on which our scenario relies; Section 5.3 describes a QoS-aware system composition hierarchy and Section 5.4 presents the goal refinement process for QoS Management. In Section 5.5 we assess goal selections tailored to QoS Management, making use of our prototype implementation. We provide a description of the automated acquisition of policies of our refinement scenario in Section 5.6. Section 5.7 describes the policies refined through the scenario and Section 5.8 provides a brief analysis of the enforcement of the policies refined in the scenario. Finally Section 5.9 concludes this chapter providing some future work in the policy refinement area.

5.2 Quality of Service Management Application Domain

In order to support the Quality of Service (QoS) guarantees for the forthcoming services, Next Generation IP Networks will make use of technologies such as Differentiated Services (DiffServ) and Multi-Protocol Label Switching (MPLS) for traffic engineering and network-wide resource management. In addition, the volume and type of the traffic injected by these services will need to be controlled as the means to both, prevent QoS degradation of active services, and verify that clients inject traffic in accordance with pre-agreed Service Level Agreements (SLAs). In order to address the challenge of QoS delivery in Next Generation IP Networks, the research community has envisaged the integration of Service Management and Traffic Engineering functions [Tri01].

In this Section, we present an approach for intra-domain QoS Management based on the principles developed in the context of the IST project TEQUILA - Traffic Engineering for Quality of Service for the Internet at Large Scale [Tri01]. To the best of our knowledge, this is the only approach that brings together Service Management and Traffic Engineering functionalities to provide an overall architecture for QoS support in IP Networks. We initially present the generic concept of this approach, and we then lay down the use of policies as a means of extending the programmability for both, Service Management and Traffic Engineering functions.

5.2.1 Bidirectional Approach for QoS Management

A simplified representation of the *TEQUILA* approach is depicted in Figure 37, which demonstrates the integration of Service Management and Traffic Engineering functions to achieve Quality of Service provisioning.

The Service Management part has two objectives: the maximisation of traffic entering the network, and the commitment of the service provider's QoS guarantees. As the traffic entering the network is a function of the number of subscribed contracts and active services, admission control mechanisms are defined for service subscriptions and invocation requests. QoS commitment is addressed by enforcing preventive and corrective actions as a means to police misbehaving users, and also to resolve potential cases of network congestion.

The Traffic Engineering functionality is concerned with the management of physical network resources. An off-line dimensioning process is responsible for mapping the predicted traffic demand to the physical network resources. In addition, real-time operations are implemented as the means to first, balance the load amongst the established Label Switched Paths (LSPs) in the network, and second, to ensure that link capacities are appropriately distributed among the different Per-Hop-Behaviours (PHBs) sharing each link. These real-time operations react dynamically to statistical traffic fluctuations.

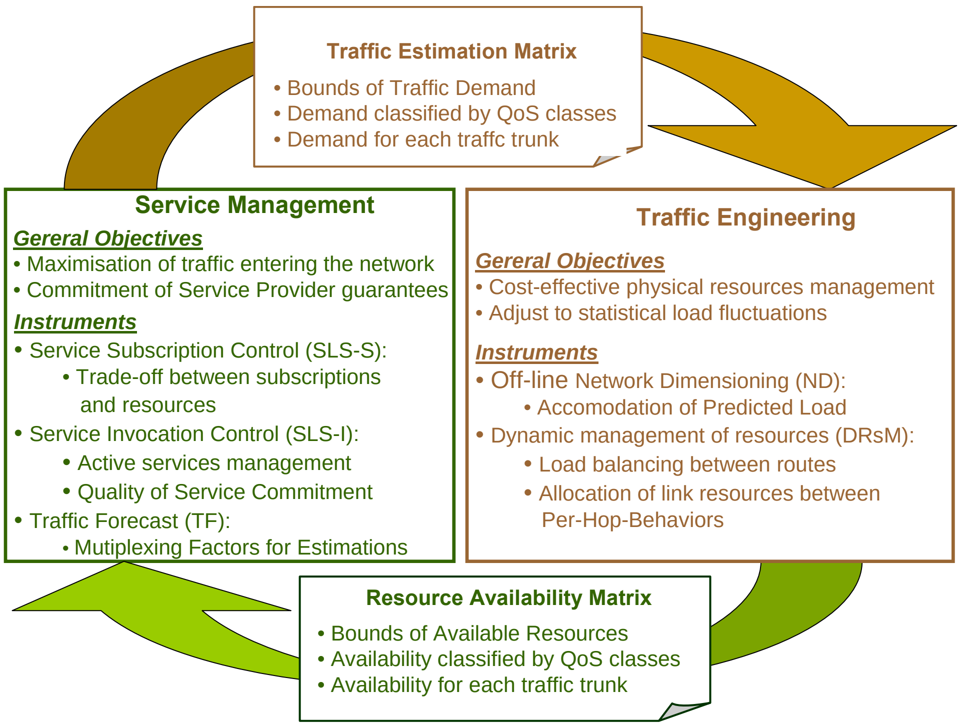

Traffic Estimation Matrix
• Bounds of Traffic Demand
• Demand classified by QoS classes
• Demand for each traffc trunk
Service Management
Gereral Objectives
• Maximisation of traffic entering the network
• Commitment of Service Provider guarantees
Instruments
• Service Subscription Control (SLS-S):
• Trade-off between subscriptions and resources
• Service Invocation Control (SLS-I):
• Active services management
• Quality of Service Commitment
• Traffic Forecast (TF):
• Mutiplexing Factors for Estimations
Traffic Engineering
Gereral Objectives
• Cost-effective physical resources management
• Adjust to statistical load fluctuations
Instruments
• Off-line Network Dimensioning (ND):
• Accomodation of Predicted Load
• Dynamic management of resources (DRsM):
• Load balancing between routes
• Allocation of link resources between Per-Hop-Behaviors
Resource Availability Matrix
• Bounds of Available Resources
• Availability classified by QoS classes
• Availability for each traffic trunk

Figure 37. Bidirectional approach for intra-domain QoS management

In general terms, the Service Management function abstracts and classifies the traffic demand into QoS classes, i.e. traffic streams sharing the same edge-to-edge QoS requirements (packet loss, delay, and throughput), and generates a *Traffic Estimation Matrix* (TEM). The latter provides the bounded demand (i.e. minima and maxima), for each QoS class, on a per-Traffic Trunk fashion. In the sequel, a traffic trunk (TT) will be considered as an ingress/egress node pair.

The TEM is in turn passed to the Traffic Engineering functions which accommodate the traffic estimation demands into the physical network resources. This process results in a *Resource Availability Matrix* (RAM) that contains the available resources again, for each QoS-class, and for each traffic trunk of the network. Finally, the RAM is passed to the Service Management function, where it is used for subscription and invocation control. The exchange of the aforementioned matrices between the two sub-systems of the architecture, takes place in long-term periods which are known as Resource Provisioning Cycles (RPCs).

5.2.2 Service Management Functionality

In this section we describe the policy-influenced processes of the Service Management functionality of TEQUILA. The Figure 38 depicts the sub-functions that integrate it, namely Service Subscription, Service Invocation and Traffic Forecast. A brief description of these components is described hereafter. A detailed description of the Policy-based Service Management functionality can be found in [Myk03].

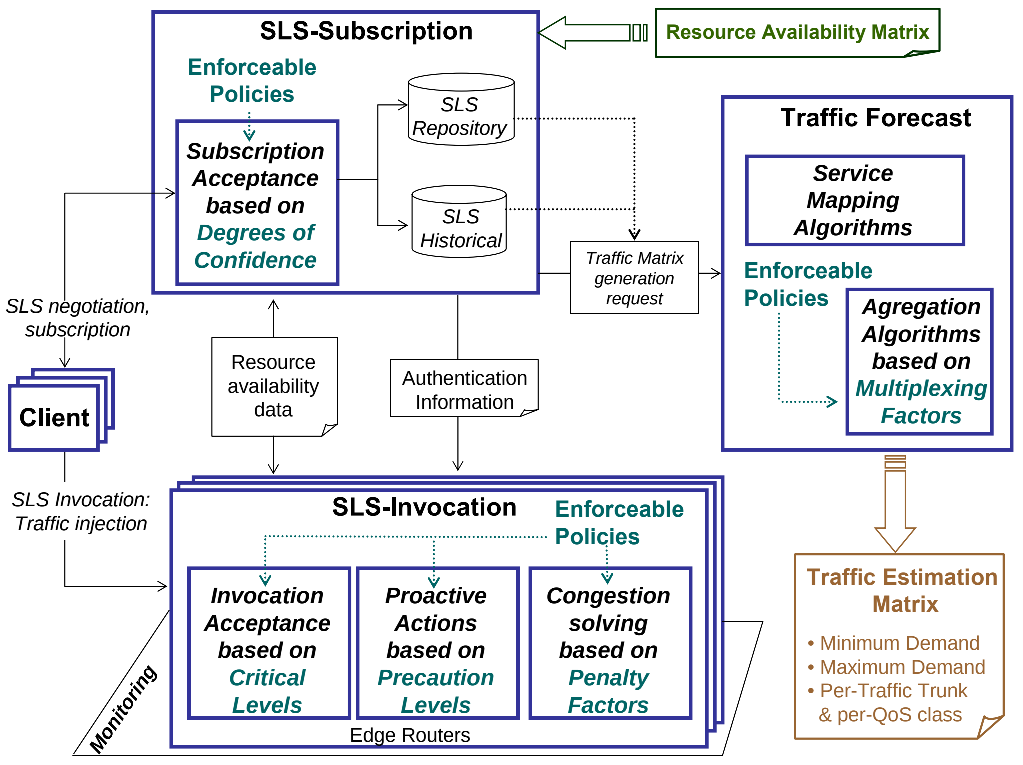

SLS-Subscription
Enforceable Policies
Subscription Acceptance based on Degrees of Confidence
SLS Repository
SLS Historical
Resource Availability Matrix
Traffic Forecast
Service Mapping Algorithms
Enforceable Policies
Agregation Algorithms based on Multiplexing Factors
Traffic Matrix generation request
SLS negotiation, subscription
Client
Resource availability data
Authentication Information
SLS Invocation: Traffic injection
SLS-Invocation
Enforceable Policies
Invocation Acceptance based on Critical Levels
Proactive Actions based on Precaution Levels
Congestion solving based on Penalty Factors
Monitoring
Edge Routers
Traffic Estimation Matrix
• Minimum Demand
• Maximum Demand
• Per-Traffic Trunk & per-QoS class

Figure 38. Service management functionality of TEQUILA

The Service Subscription (SLS-S) sub-function is carried out by a centralized component that manages Service Level Specifications (SLSs) subscription requests. It maintains databases for subscribed services and service-related historical data. It accepts/negotiates subscriptions based on policies that define some degrees of confidence with which the subscribed services would enjoy their contractual traffic rates. In this sense, the SLS-S addresses the trade-off between the number of subscriptions and the confidence for ensuring QoS.

The Service Invocation (SLS-I) sub-function is carried out by distributed components located at the edges of the network. It controls the number and type of active services and consequently, the volume of injected traffic. SLS-I addresses the trade-off between maximizing the number of admitted invocations and preventing QoS degradation by overloading the network. For this, it integrates three policy-based mechanisms: First, SLS-I performs admission control based on policy-based levels that define when the likelihood to overwhelm the network is considered critical. With this regard, service invocations are accepted when the actual measured level is below this critical level. Second, the SLS-I components prevent QoS degradation by carrying out proactive actions based on policy-based precaution levels. The enforcement of these proactive actions may either result in service rate allocation re-adjustments, and/or admission control re-adjustments for new invocations. Third, when congestion occurs in the network, SLS-I applies different penalties defined by policies. Again, the result of these penalties may affect either service rate re-allocation and/or admission control re-adjustments.

The Traffic Forecast (TF) sub-function is dedicated to translate the SLS customer-oriented information into resource-oriented data. It deduces the traffic demand based on subscribed SLSs, and generates the Traffic Estimation Matrix (TEM). For this, TF applies service mapping algorithms to adapt the service-oriented information into the appropriate resource-oriented format. In addition, it uses policy-defined multiplexing factors to derive the minimum and maximum demand for each QoS class and traffic trunk. These factors specify the criteria by which a service is considered to enjoy *almost satisfied* and *fully satisfied* rates, which are eventually used to define the minimum and maximum values for the traffic demand.

5.2.3 Traffic Engineering Functionality

In this section we describe the policy-influenced processes of the Traffic Engineering functionality of TEQUILA. The Figure 39 depicts the sub-functions that integrate this functionality, namely Network Dimensioning (ND), Dynamic Resource Management (DRsM) and Dynamic Route Management (DRtM). A brief description of these sub-functions is provided hereafter. A detailed description of the Traffic Engineering functionality can be found in [Tri03] and [Fle02].

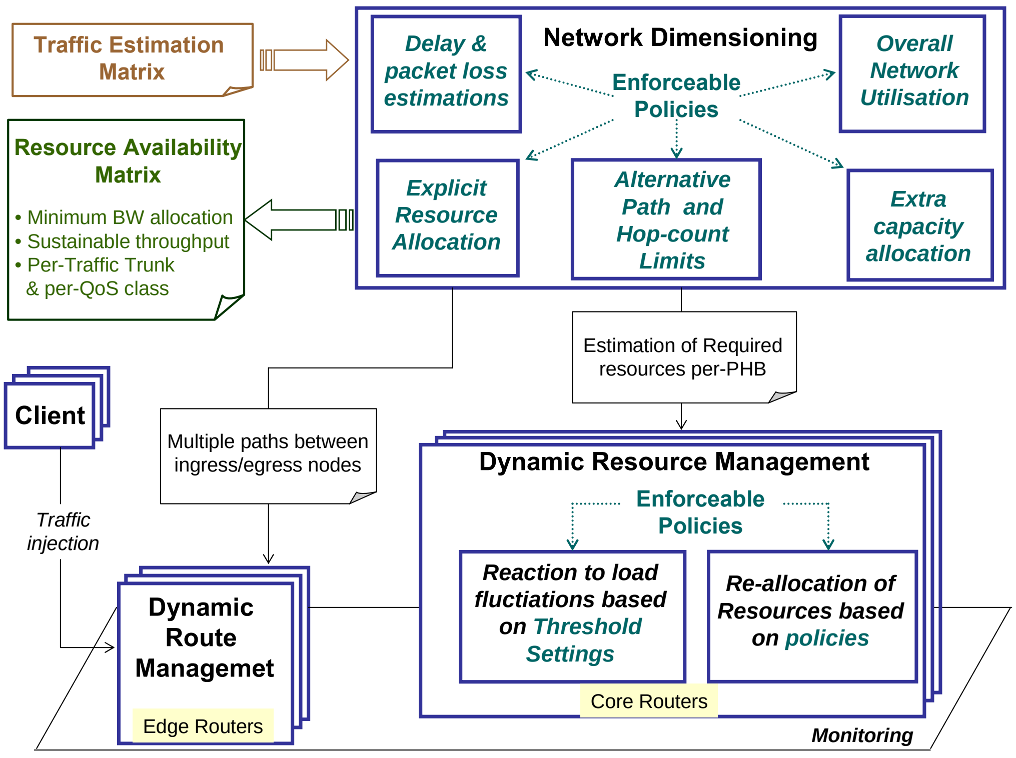

Traffic Estimation Matrix
Resource Availability Matrix
• Minimum BW allocation
• Sustainable throughput
• Per-Traffic Trunk & per-QoS class
Network Dimensioning
Delay & packet loss estimations
Enforceable Policies
Overall Network Utilisation
Explicit Resource Allocation
Alternative Path and Hop-count Limits
Extra capacity allocation
Estimation of Required resources per-PHB
Client
Traffic injection
Multiple paths between ingress/egress nodes
Dynamic Resource Management
Enforceable Policies
Reaction to load fluctiations based on Threshold Settings
Re-allocation of Resources based on policies
Dynamic Route Managemet
Edge Routers
Core Routers
Monitoring

Figure 39. Traffic engineering functionality of TEQUILA

The Network Dimensioning sub-function is in charge of accommodating the traffic estimations into the physical network resources. To accomplish this, ND uses the Traffic Estimation Matrix generated by the Service Management function. The purpose of the Network Dimensioning sub-function is twofold: First, to produce the Resource Availability Matrix (RAM) which contains the calculated available resources for the forthcoming Resource Provisioning Cycle, and second, to provide the lower-level TE components (DRtM and DRsM) with resource management guidelines.

With respect to the Resource Availability Matrix (RAM), ND calculates on a per-QoS-class fashion, the minimum “always” available allocation of resources and the sustainable throughput (BW available above the minimum available resources).

Regarding the resource management guidelines provided to the TE sub-functions, Dynamic Resource Management (DRsM) and Dynamic Route Management (DRtM), these are calculated based on historical data and customer subscriptions. For the DRtM and DRsM sub-functions, these guidelines represent “nominal” values within which they will work. For instance, the DRsM receives from the ND different estimates of the required resources for each Per-Hop-Behaviour (PHB).The DRtM instead receives the different LSP paths for the multiple traffic trunks of the network. The DRsM sub-function is assessed through the core routers of the underlying network. The DRtM sub-function instead is assessed by ingress/egress nodes.

In the Network Dimensioning sub-function, policies are used to extend the programmability in the following aspects.

- *Delay and packet loss estimation requirements*. Policies define the criteria of how to consider hop-count with respect to delay and packet loss constraints.
- *Explicit resource allocation*. Policies define explicit LSP routes and/or explicit BW allocations during the dimensioning process.
- *Alternative path and Hop-count bounds*. Policies define the limits on the number of alternative paths for the traffic trunks and the limit on the number of hops for the LSPs.
- *Network utilisation*. Policies are used to provide guidelines on how to proceed with the distribution of resources with respect to the overall network utilisation.
- *Extra and over provisioned capacity distribution*. Once the minimum demands have been allocated, there may be remaining capacity in some or all network links. Policies are used to define the criteria of how the remaining capacity may be distributed amongst the different PHBs. In addition, these policies define how to reduce the capacity allocation amongst the different PHBs under over-provisioning states.

5.3 A System Composition Hierarchy for QoS Management

In this sub-Section we follow the guidelines of our systematic approach to define a system composition hierarchy for QoS Management. We exploit the hierarchical relationships of the TEQUILA approach and consider both, Service Management and Traffic Engineering. These two system functions are in turn in charge of controlling the QoS Provisioning Management service delivered by the TEQUILA approach. The Figure 40 depicts the system composition hierarchy of this application domain for which a general description is given hereafter.

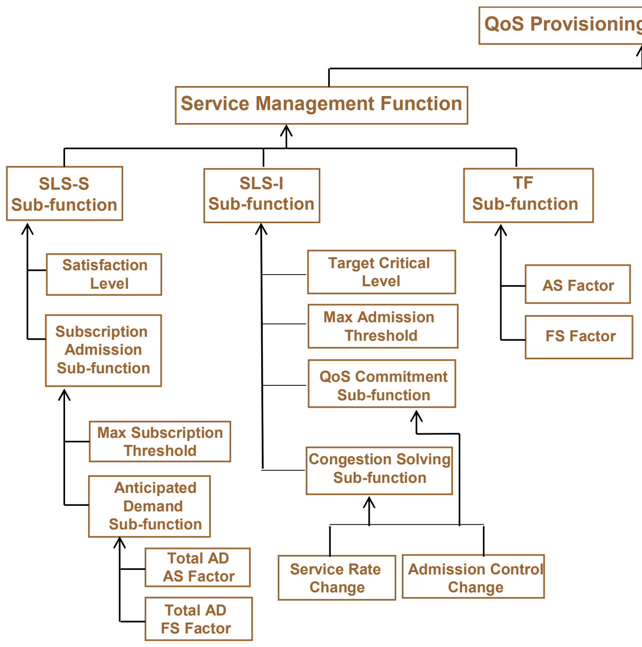


Figure 40. System composition hierarchy for QoS Management

5.3.1 Service Management Function

In the TEQUILA architecture, the Service Management Function is integrated by Service Subscription (SLS-S) Sub-function, Service Invocation (SLS-I) Sub-function, and Traffic Forecasting (TF) sub-function. These sub-functions define the next level in the composition hierarchy of the Service Management Function as we show in Figure 40.

As we described earlier, the SLS-S Sub-function is in charge of executing the service subscription functions of TEQUILA. This sub-function is influenced by a parameter used to define degrees of confidence for service fulfilment, namely the "Satisfaction Level", and a sub-function dedicated to control the subscription's admissions defined as "Subscription Admission Sub-function" of the composition hierarchy shown in Fig. 40. The "Subscription Admission Sub-function" is in turn influenced by a parameter that defines an admission control for subscriptions identified as "Max Subscription Threshold", and a sub-function that defines the total anticipated demand strategy to follow for subscription admissions' control identified as "Anticipated Demand Sub-function". The latter sub-function is influenced by two parameters that are used to calculate the total anticipated demand with regard to the satisfaction factors almost satisfied (AS) and fully satisfied (FS). These two parameters are identified as “Total AD AS Factor” and “Total AD FS Factor” in the composition hierarchy shown in Fig. 40.

Coming back to what we described earlier, the SLS-I Sub-function is in charge of executing the service invocation functions of TEQUILA. The composition hierarchy for this sub-function is defined by a parameter that is used to define the likelihood of overwhelming the network identified as “Target Critical Level”, and another parameter that defines the admission functionalities of the SLS-I Sub-function identified as “Max Admission Threshold”. In addition, the SLS-I Sub-function is integrated by a sub-function that achieves the programmability of configuring the proactive actions when statistical fluctuations of the traffic load occur. This is identified as “QoS Commitment Sub-function” in the composition hierarchy shown in Fig. 40. In addition, the SLS-I Sub-function is integrated by a sub-function that deals with proactive actions commited to resolve congestion states in the network. This is identified as “Congestion Solving Sub-function” in Fig. 40. Both of the latter two sub-functions are influenced by two parameters that deal respectively, with service rate adjustments and admission control adjustments. These two are identified as “Service Rate Change” and “Admission Control Change” in the composition hierarchy of Fig. 40.

Finally, as we described earlier, the TF Sub-function of the Service Management Function provides the programmability to define the multiplexing factors for traffic demand estimations. This sub-function is influenced by two parameters that specify the criteria by which a service is considered to enjoy *almost satisfied* and *fully satisfied* rates. These two are identified as “AS Factor” and “FS Factor” respectively in the composition hierarchy shown in Figure 40.

5.3.2 Traffic Engineering Function

In the TEQUILA architecture, the Traffic Engineering Function is integrated by the Network Dimensioning (ND) Sub-function and Dynamic Resource Management (DRsM) Sub-function. These two define the next level in the composition hierarchy of the Traffic Engineering Function shown in Figure 40.

As we described earlier, the ND Sub-function is in charge of accommodating traffic estimations into the physical network resources. The system composition with this regard is integrated by first, a sub-function that assesses the programmability to influence the qualitative calculation of the resources to satisfy the minimum anticipated demand. This sub-function is defined as “Minimum Demand Sub-function” in the composition hierarchy shown in Figure 40. Following on with the latter sub-function, it is influenced by four parameters that define the next level of the composition hierarchy. The “Path Hop Limit” parameter is defined to cope with administrative constraints of the number of hops for the LSPs. The “Network Utilization” parameter of the hierarchy defines the criteria of the utilization of the overall network. The “Hop-count Estimation” parameter of the composition hierarchy is defined to address the considerations of hop-count as the means to estimate link delay and packet losses. Finally, the “Explicit Allocation” parameter influencing the Minimum Demand Sub-function has been defined to assess administrative decisions regarding explicit LSP, and BW allocation in the core network.

Following on with the “ND Sub-function” of the TEQUILA approach, this is integrated by a sub-function that influences the distribution of resources once the ND has allocated the minimum demand during the ND process. This sub-function defined as “Extra Capacity Sub-function”, together with the “Minimum Demand Sub-function” described above, both compose the next level from the “ND Sub-function” of the hierarchy shown in Figure 40. The “Extra Capacity Sub-function” is influenced by two parameters; first, the “Distribution Capacity” parameter which is defined to influence the resource calculation of spare capacity in the core links; and second, the “Reduction Capacity” parameter that achieves the programmability on how to reduce the capacity allocated to each PHB under link capacity shortage.

Finally, the “DRsM Sub-function” of the Traffic Engineering Function is influenced by two parameters. The parameter “Threshold Setting” is used to define reactions to statistical fluctuations of traffic load. Finally, the parameter “Allocation Modification” is defined to cope with the actual allocation as a result of statistical fluctuations. These representations are also shown in the hierarchical composition of Figure 40.

5.4 Goal Refinement for QoS Management

With this Sub-section we start the execution of the policy refinement process for QoS Management. Namely, we describe the goal refinement process for the TEQUILA approach. Throughout this Chapter, we make use of our prototype implementation to carry assess the refinement process.

5.4.1 Starting Goal Refinement for QoS Management

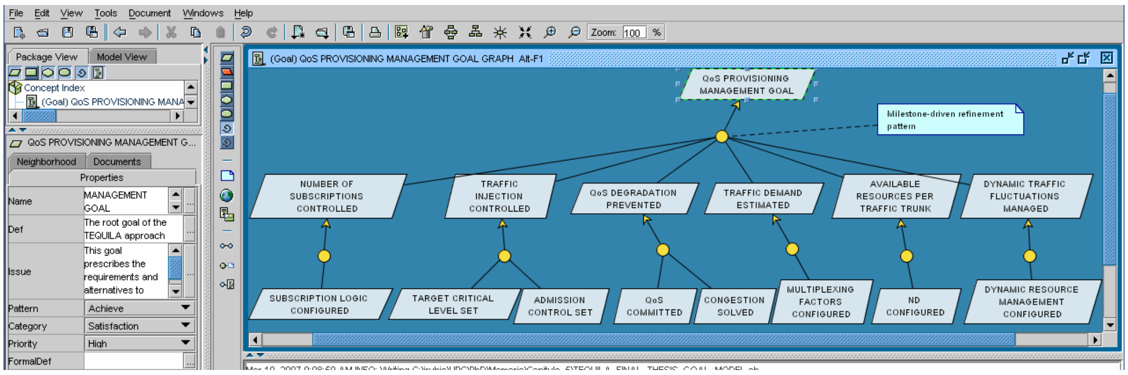
It is a matter of realistic judgment to decide which high-level guidelines should define the view of QoS delivery. Different methods and mechanisms can be found in the literature [Bar06] to specify High-level Goals and to provide indicators to assess IT performance related to them. In the context of our scenario, these represent the High-level Goals with which the Administrator Developer intends to control QoS provisioning. As our methodological approach prescribes, a key issue during the definition of High-level Goals is the association of High-level Goals with the system functionality. Namely, it is imperative that High-level goals should be aligned with the scope of the target system, in this case with the TEQUILA approach. In addition, High-level Goals should be consistent with the functionalities designed/implemented by the TEQUILA approach. In our QoS management scenario the Administrator Developer starts the goal refinement process defining the following High-level Goals, following the functionality of the TEQUILA approach:

- *Number of Subscriptions Controlled*
- *Traffic Injection Controlled*
- *QoS Degradation Prevented*
- *Traffic Demand Estimated*
- *Available Resources per-Traffic Trunk Calculated*
- *Dynamic Traffic Fluctuations Managed*

The next Section assesses goal refinement for the above High-level goals following the guidelines of our methodological approach, and making use of our prototype implementation.

5.4.2 Assessing Goal Refinement for QoS Management

Having defined the to-be-refined High-level goals, these are linked to the root goal of the goal graph structure. For this, the Administrator Developer defines the "QoS Provisioning Management" root goal. The developer makes use of the Objectiver support to store this goal in the database for further use; see Package View box in the upper-left part of Figure 41. At this initial step of the refinement process the administrator developer follows the KAOS methodology to bring a milestone-driven refinement pattern to link the root goal with the six High-level goals defined to control QoS provisioning described earlier. This allows systematising goal refinements for QoS Management and establishing pre-defined temporal prescription of goals; see the Properties box in the bottom-left part of Figure 41, specially the Achieve pattern formalisation for the "QoS Provisioning Management" goal. Every goal instance throughout the goal-graph contains similar properties like temporal prescription, refinements, etc. The goal graph structure is then built upon six sub-trees for which further refinements are carried out taking into account the system composition hierarchy. The following is a brief description of the composition-aware refinement level shown at the bottom of the goal graph shown in Fig. 41.


File
Edit
View
Tools
Document
Windows
Help
Zoom: 100 %
Package View
Model View
Concept Index
(Goal) QoS PROVISIONING MANA
QoS PROVISIONING MANAGEMENT G...
Neighborhood
Documents
Properties
(Goal) QoS PROVISIONING MANAGEMENT GOAL GRAPH Alt-F1
QoS PROVISIONING MANAGEMENT GOAL
Milestone-driven refinement pattern
NUMBER OF SUBSCRIPTIONS CONTROLLED
TRAFFIC INJECTION CONTROLLED
QoS DEGRADATION PREVENTED
TRAFFIC DEMAND ESTIMATED
AVAILABLE RESOURCES PER TRAFFIC TRUNK
DYNAMIC TRAFFIC FLUCTUATIONS MANAGED
SUBSCRIPTION LOGIC CONFIGURED
TARGET CRITICAL LEVEL SET
ADMISSION CONTROL SET
QoS COMMITTED
CONGESTION SOLVED
MULTIPLEXING FACTORS CONFIGURED
ND CONFIGURED
DYNAMIC RESOURCE MANAGEMENT CONFIGURED

Name	MANAGEMENT GOAL
Def	The root goal of the TEQUILA approach
Issue	This goal prescribes the requirements and alternatives to
Pattern	Achieve
Category	Satisfaction
Priority	High
FormalDef	

Figure 41. Goal graph for the QoS Provisioning Management goal

The High-level Goal *Number of Subscriptions Controlled* has direct impact on traffic predictions. In addition, it is directly influenced by the "Service Subscription Sub-function" of the TEQUILA approach given that the latter is used to control the acceptance, rejection/negotiation of service subscriptions. This way, the developer refines the "Number of Subscriptions Controlled" into the "Subscription Logic Configured" goal.

Regarding the *Traffic Injection Controlled* goal, this has been established to address the trade-off between maximizing the traffic entering the network and the quality of service enjoyed by the active services. In the context of our QoS Management scenario, this aspect is influenced by the "Target Critical Level" and the "Max Admission Threshold", both parameters of the SLS-I Sub-function of TEQUILA. For this reason, the administrator developer refines the "Traffic Injection Controlled" High-level Goal is refined into the goals "Target Critical Level Set" and "Admission Control Set".

The High-level Goal *QoS Degradation Prevented* is defined to guarantee active services to enjoy their contracted Quality of Service. In order to prevent QoS degradation, the QoS management approach implements active services rate change and congestion avoidance mechanisms, both controlled by the QoS Commitment Sub-function and Congestion Solving Sub-function of the TEQUILA approach. In this sense, the Administrator Developer refines the "QoS Degradation Prevented" goal into the "QoS Committed" and "Congestion Solved" goals.

Following on, the High-level Goal *Traffic Demand Estimated* has been defined to influence the allocation of physical resources given that the dimensioning process considers minimum and maximum bounds of traffic estimations. Once the latter bounds are influenced by the multiplexing factors defined by the TF Sub-function of TEQUILA, the "Traffic Demand Estimated" is refined into the goal "Multiplexing FactorsConfigured" by the Administrator Developer.

Regarding the *Available Resources per-Traffic Trunk Calculated* High-level Goal, it has direct impact on the subscription and invocation admission control mechanisms in the sense that these calculations are contained in the Resource Availability Matrix provided by the Traffic Engineering Function to the Service Management Function. Moreover, a key role for the calculation of these resources is the ND Sub-function and its associated sub-functions. For instance, ND Sub-function influences the qualitative way to allocate both, the minimum demand, and the remaining resources of the network. For this reason, the developer refines the "Available Resources per-Traffic Trunk Calculated" High-level Goal into the "ND Configured" goal as we show in the lower part of Fig. 41.

Finally, the High-level Goal *Dynamic Traffic Fluctuations Managed* has direct impact on how the core network reacts to statistical traffic fluctuations. The QoS Management approach considers dynamic threshold and allocation management mechanisms to deal with these situations. This aspect is controlled by the DRsM Sub-function of the TEQUILA approach and consequently, the developer refines the former goal into the "Dynamic Resource Management Configured" goal as we show in the lower part of Fig. 41.

For the next phase of the goal refinement process we have elaborated three generic goal-graph structures driven by the QoS-oriented system composition hierarchy of our application domain. The Figure 42 shows the Service Management Configured goal-graph which has been refined considering the Service Management Sub-function of the TEQUILA approach. The goal graph represents the different strategies that the system is capable to achieve for the Service Management Function. Basically, this goal graph structure has been built upon three sub-trees defined by the SLS-S Sub-function, SLS-I Sub-function, and TF Sub-function. For this level of refinement, the administrator developer brings into this specific context a milestone-driven refinement pattern for goal refinement (see Fig. 42).

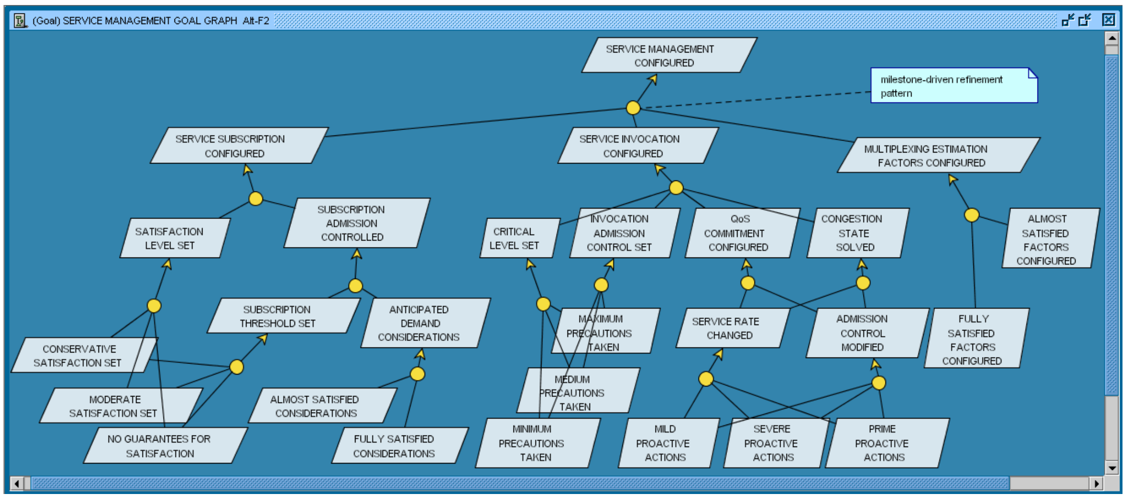

(Goal) SERVICE MANAGEMENT GOAL GRAPH Alt-F2
SERVICE MANAGEMENT CONFIGURED
milestone-driven refinement pattern
SERVICE SUBSCRIPTION CONFIGURED
SERVICE INVOCATION CONFIGURED
MULTIPLEXING ESTIMATION FACTORS CONFIGURED
SATISFACTION LEVEL SET
SUBSCRIPTION ADMISSION CONTROLLED
CRITICAL LEVEL SET
INVOCATION ADMISSION CONTROL SET
QoS COMMITMENT CONFIGURED
CONGESTION STATE SOLVED
ALMOST SATISFIED FACTORS CONFIGURED
SUBSCRIPTION THRESHOLD SET
ANTICIPATED DEMAND CONSIDERATIONS
MAXIMUM PRECAUTIONS TAKEN
SERVICE RATE CHANGED
ADMISSION CONTROL MODIFIED
FULLY SATISFIED FACTORS CONFIGURED
CONSERVATIVE SATISFACTION SET
MODERATE SATISFACTION SET
ALMOST SATISFIED CONSIDERATIONS
MEDIUM PRECAUTIONS TAKEN
NO GUARANTEES FOR SATISFACTION
FULLY SATISFIED CONSIDERATIONS
MINIMUM PRECAUTIONS TAKEN
MILD PROACTIVE ACTIONS
SEVERE PROACTIVE ACTIONS
PRIME PROACTIVE ACTIONS

Figure 42. Service Management Configured goal graph

Similar goal-graphs have been defined for the Traffic Engineering Function. More specifically, the Figure 43 shows the Network Dimensioning Configured goal-graph structure that incorporates the requirements and alternatives to dimension both, the minimum traffic estimations and the extra remaining capacity for the links of the physical network. This goal graph has been built upon the composition hierarchy for the ND Sub-function of the TEQUILA approach. On the other hand, the Figure 44 shows the corresponding Dynamic Resource Management (DRsM) Configured goal graph, based on the DRsM Sub-function of the TEQUILA approach. This goal graph formalises the requirements and alternatives to control statistical fluctuations of traffic in the core links of the physical network. The DRsM Configured goal graph structure is built upon two sub-trees that correspond to threshold monitoring and BW re-allocation tasks.

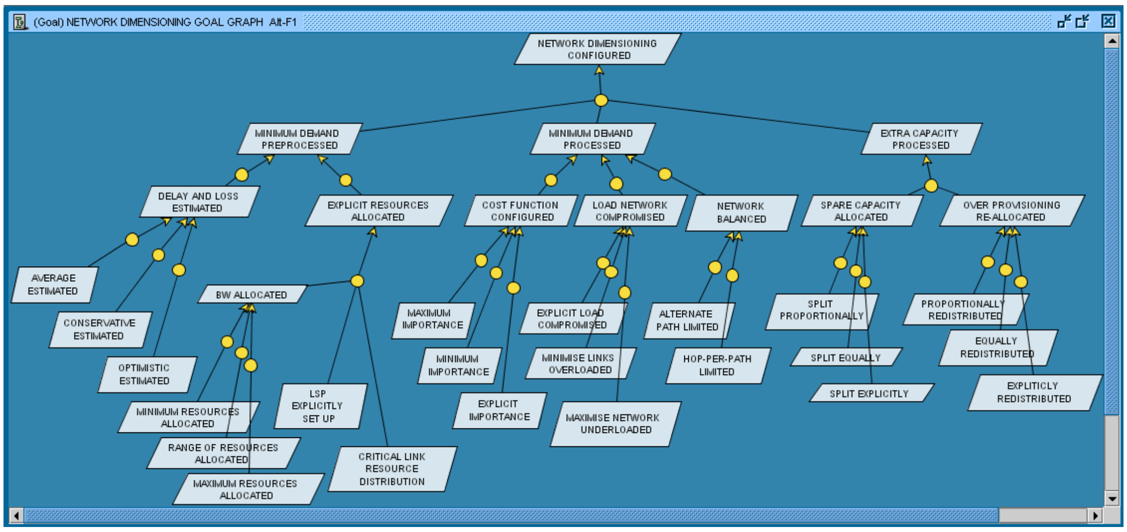


Figure 43. Network Dimensioning Configured goal graph

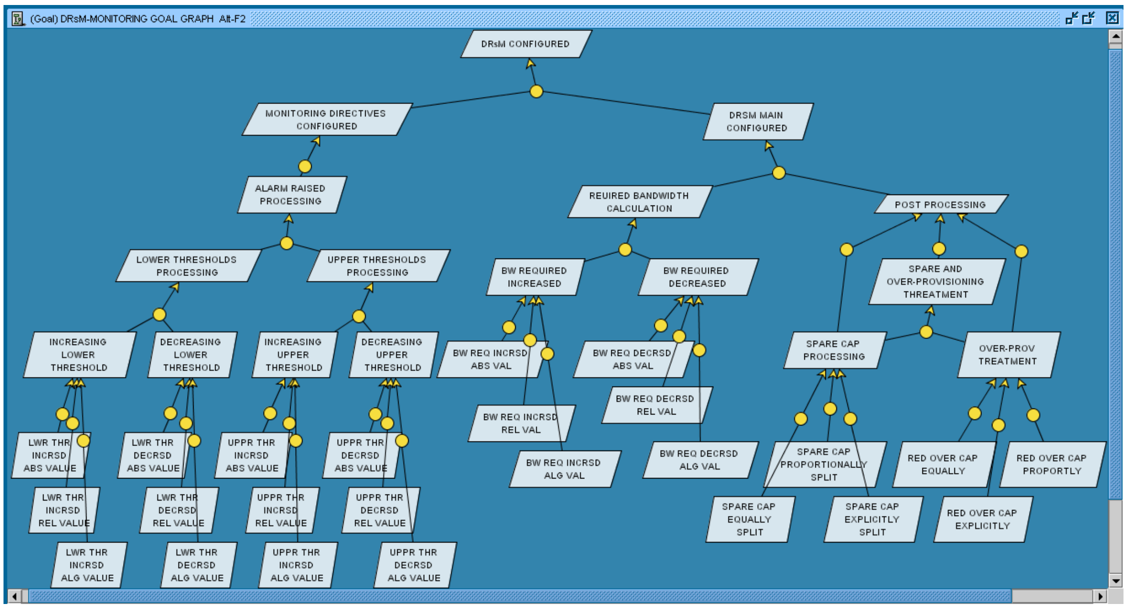


Figure 44. Dynamic Resource Management Configured goal graph

5.5 Executing Goal Selection for QoS Management

The above High-level Goals formalise the guidelines that the Administrator Developer has defined to manage QoS provisioning. During the operation of the policy system, the Administrator Consultant should acknowledge these guidelines to define the operative or "particular" view of QoS provisioning through Goal Selection. Similar to the definition of High-level Goals, the operative or "particular" view of QoS provisioning depends on realistic judgement, statistical data or previous administrative experiences. This section describes a goal selection process tailored to mange QoS provisioning. This section is divided in two sections. We first provide a holistic and realistic view of QoS provisioning and finally, we use our prototype implementation to assess Goal Selection aligned to the former.

5.5.1 Defining a Holistic View for QoS Management

We have defined a holistic view for QoS Management following the principles of the TEQUILA approach. Basically, this view is a reproduction of the High-level Goals that define an operative or "particular" view of QoS provisioning. This operative particular view is summarised in Figure 45 for which a brief description is provided thereafter.

Number of Subscriptions Controlled	Available Resources per-TT Calculated	QoS Degratation Prevented	Traffic Demand Estimated	Dynamic Traffic Fluctuations Managed
Maximise subscriptions with the **Highest Confidence** to provide the agreed QoS	Allocate Minimum demand with **Average hop-count** estimations **Minimise** link overloading	**Mild Proactive** actions when traffic is between the resources **available at congestion** and the **maximum**	**Almost** and **Fully Satisfied** rates are **100%** the subscribed ones for **real-time traffic**	**Increase or decrease** resources by **10%** the range provided by ND **when load fluctuates** **10%** the range provided by ND
Traffic Injection Controlled **Maximise the Traffic** injected into the network	Redistribute over-provisioning and spare capacity **proportionally**	**Prime Proactive** actions at **congestion**	Almost and Fully satisfied rates are **60% and 85%** respectivelly for **other type of traffic**	**Redistribute the changes proportionally** to every PHB Redistribute **spare capacity proportionally** to every PHB

Figure 45. Particular view of QoS provisioning

The first column concerns with the view of QoS delivery with respect to the *Number of Subscriptions Controlled* and the *Traffic Injection Controlled*. In this example the consultant opts to maximise the number of subscriptions at the Highest Confidence levels of providing the agreed Quality of Service. This High-level Goal implies that the underlying components of the TEQUILA system would make sure that most of the accepted subscriptions will enjoy their contracted rates and then congestion occurrence would be highly unlikely. Under these circumstances the administrator opts to maximize the Traffic Injected into to the network given that the latter is linked to the accepted subscriptions whose QoS satisfaction is highly likely.

Regarding the High-level Goal *Calculated Available Resources per-TT*, the consultant defines the view on how to influence the calculation algorithms of the Network Dimensioning process.

- For delay and losses purposes, the administrator opts for considering the average delay and packet loss induced by the links along the paths as the base to establish hop-count constraints for every link along the network.
- Regarding resource allocation, the administrator opts for uniformly distributing the predicted load amongst all the available links in the network; namely, minimizing that some links become overloaded while others are under-loaded.
- Finally, once the minimum estimated demand has been allocated, the administrator opts to distribute the over-provisioned and spare capacity proportionally to every PHB.

Regarding the High-level Goal *QoS Degradation Prevented*, the consultant opts to enforce “mild” proactive actions when the traffic injected into the network is between the available resources at congestion and the maximum available resources for the Traffic Trunks (TT). In addition, “Prime” proactive actions should take place during congestion.

For the High-level Goal *Traffic Demand Estimated*, the consultant defines multiplexing factors that define how clients are to enjoy Almost Satisfied (AS) and Fully Satisfied (FS) rates. This particular administrative view opts to differentiate between service types, providing 100% of the subscribed rates to AS and FS in case of real-time traffic. In contrast, 60% and 85% of the subscribed rates may be considered for AS and FS respectively for other types of traffic.

Finally, with regard to the *Dynamic Traffic Fluctuations Managed*, the consultant opts to increase or decrease resources allocations when traffic fluctuations are worth noticeable of about 10% the dynamic range provided by Network Dimensioning (ND). In this sense, the consultant opts for allocation changes in ratios of 10% the range provided by ND. In addition, the consultant opts to decrease/increase the previously increased/decreased resources, proportionally to every PHBs sharing the core links. Finally, the spare link capacity is distributed proportionally amongst the PHBs.

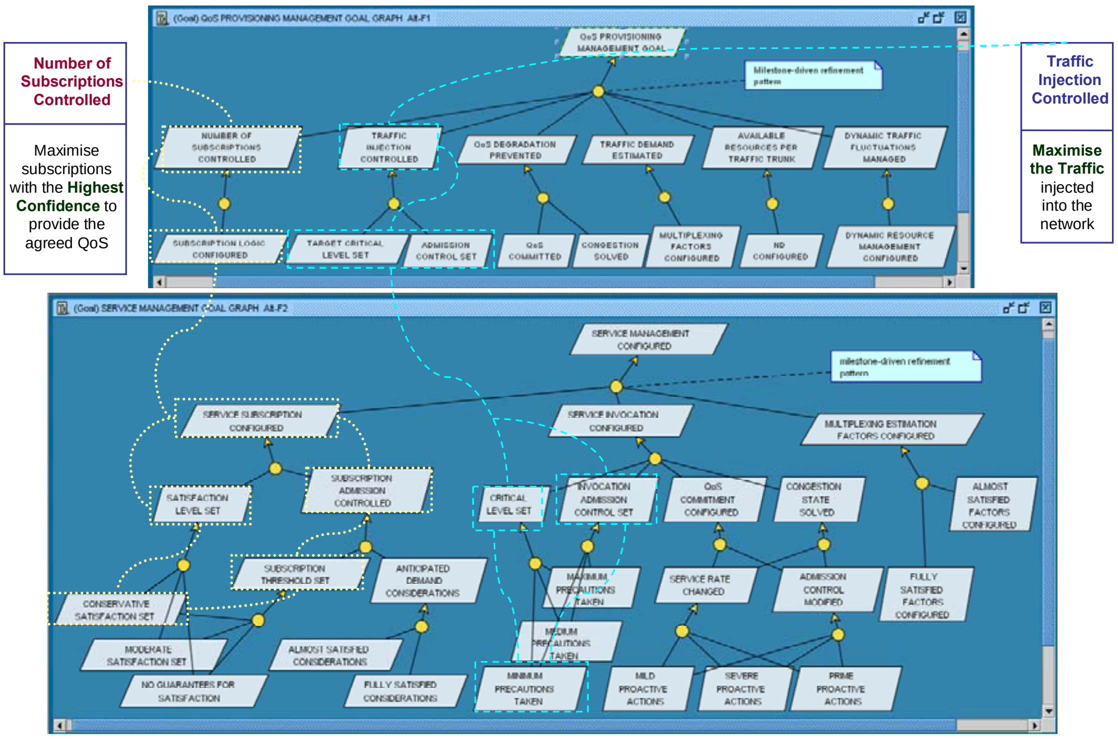
5.5.2 Assessing Goal Selection for QoS Management

As we mentioned earlier, Goal graph structures provide potential information for interpretation about how goals should be achieved. Interpretations are materialised with Goal Selections. In this sub-Section we describe the selections that materialise the administrative view of QoS Management in our application scenario.

Our prototype provides the means to browse through goal graph structures. Since the target is to refine a holistic view for QoS Management. For this, the consultant should interpret the options for each goal sub-tree of the QoS Provisioning Management goal, and select the options that better reflect the administrative view of QoS Management.

For the sub-tree defined by the High-level Goal *Number of Subscriptions Controlled* shown in the left part of Figure 46, the system guides the consultant to the “Service Subscription Configured” goal, the latter included in the Service Management goal graph. At this moment of the selection, the consultant interprets that selecting “Conservative Satisfaction Settings” for both, the “Satisfaction Level Set” goal and the “Subscription Admission Controlled” goal, reflects the view “Maximize subscriptions with the Highest Confidence to provide the agreed QoS”. For this selection the consultant selects the pattern of goals marked with dotted lines in Figure 46.

For the sub-tree defined by the High-level Goal *Traffic Injection Controlled* shown in the right part of Figure 46, the system guides the consultant to the goals “Critical Level Set” and “Invocation Admission Control Set”, the latter goals included in the Service Management goal graph. At this moment of the selection, the consultant interprets that selecting “Minimum Precautions Taken” reflects the view “Maximize the Traffic injected into the network”. The consultant correlates this selection to the fact that service subscriptions are only accepted with the highest confidence to provide the agreed QoS. Given that service invocations are directly influenced by the number of subscribed customers, the consultant opts to take minimum precautions concerning service invocations control. This selection would cause that the traffic injected into the network due to invocations is maximized. For this selection the consultant selects the pattern of goals marked with discontinuous lines in Figure 46.



Number of Subscriptions Controlled
Maximise subscriptions with the Highest Confidence to provide the agreed QoS
Traffic Injection Controlled
Maximise the Traffic injected into the network
(Goal) QoS PROVISIONING MANAGEMENT GOAL GRAPH A8-F1
QoS PROVISIONING MANAGEMENT GOAL
Milestone-driven refinement pattern
NUMBER OF SUBSCRIPTIONS CONTROLLED
TRAFFIC INJECTION CONTROLLED
QoS DEGRADATION PREVENTED
TRAFFIC DEMAND ESTIMATED
AVAILABLE RESOURCES PER TRAFFIC TRUNK
DYNAMIC TRAFFIC FLUCTUATIONS MANAGED
SUBSCRIPTION LOGIC CONFIGURED
TARGET CRITICAL LEVEL SET
ADMISSION CONTROL SET
QoS COMMITTED
CONGESTION SOLVED
MULTIPLEXING FACTORS CONFIGURED
ND CONFIGURED
DYNAMIC RESOURCE MANAGEMENT CONFIGURED
(Goal) SERVICE MANAGEMENT GOAL GRAPH A8-F2
SERVICE MANAGEMENT CONFIGURED
milestone-driven refinement pattern
SERVICE SUBSCRIPTION CONFIGURED
SERVICE INVOCATION CONFIGURED
MULTIPLEXING ESTIMATION FACTORS CONFIGURED
SATISFACTION LEVEL SET
SUBSCRIPTION ADMISSION CONTROLLED
CRITICAL LEVEL SET
INVOCATION ADMISSION CONTROL SET
QoS COMMITMENT CONFIGURED
CONGESTION STATE SOLVED
ALMOST SATISFIED FACTORS CONFIGURED
SUBSCRIPTION THRESHOLD SET
ANTICIPATED DEMAND CONSIDERATIONS
MAXIMUM PRECAUTIONS TAKEN
SERVICE RATE CHANGED
ADMISSION CONTROL MODIFIED
FULLY SATISFIED FACTORS CONFIGURED
CONSERVATIVE SATISFACTION SET
MEDIUM PRECAUTIONS TAKEN
MODERATE SATISFACTION SET
ALMOST SATISFIED CONSIDERATIONS
NO GUARANTEES FOR SATISFACTION
FULLY SATISFIED CONSIDERATIONS
MINIMUM PRECAUTIONS TAKEN
MILD PROACTIVE ACTIONS
SEVERE PROACTIVE ACTIONS
PRIME PROACTIVE ACTIONS

Figure 46. Selections for controlling subscription and traffic injection

The same approach is taken for the remaining selections. The pattern of goals marked with dotted lines shown in the left part of Figure 47 shows the selection concerning with the High-level Goal *QoS Degradation Prevented*. Here the consultant selects two strategies; a mild proactive selection deals with reactions to the critical level settings. The prime action selection instead handles potential congestion situations. Both selections involve service rate and admission control settings.

Similarly, the discontinuous lines shown in the right part of Figure 47 show the selection of goals for the High-level Goal *Traffic Demand Estimated*. In this case, the selection is constrained with specific values introduced as attributes for the lowest-level goals, e.g. the Almost Satisfied Factors Configured goal would have as attribute the value of 100% for real-time traffic, and 60% for other type of traffic.

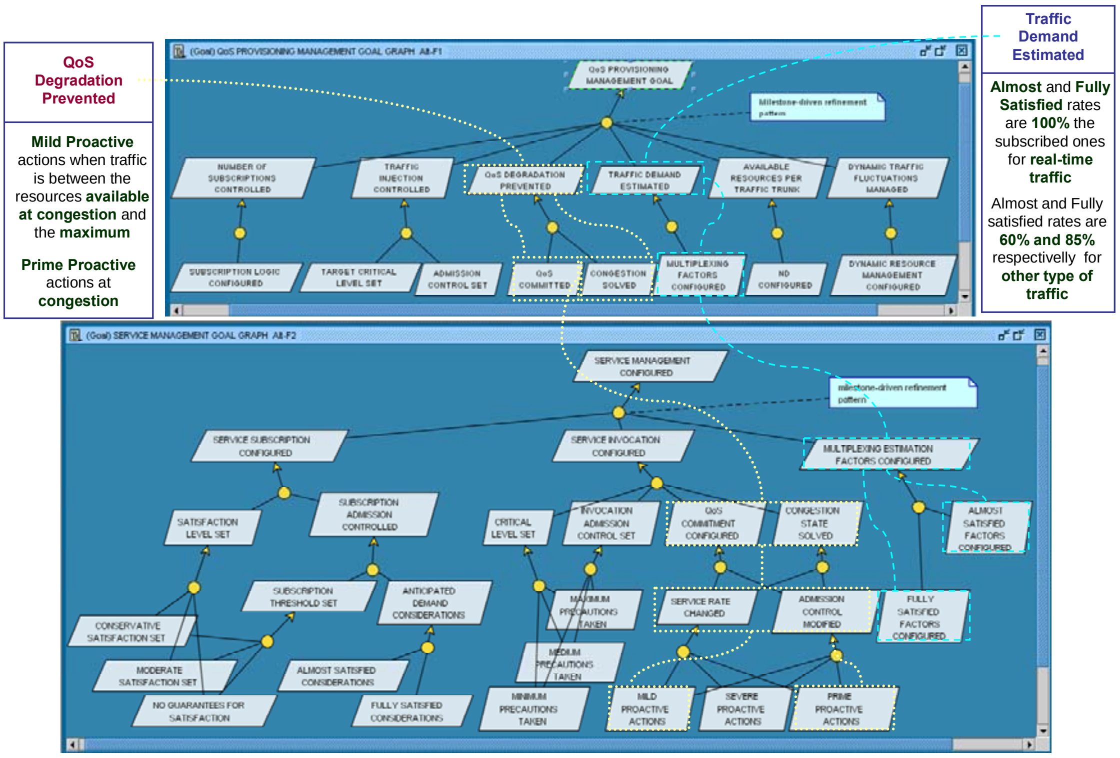

QoS Degradation Prevented
Mild Proactive actions when traffic is between the resources available at congestion and the maximum
Prime Proactive actions at congestion
(Goal) QoS PROVISIONING MANAGEMENT GOAL GRAPH All-F1
QoS PROVISIONING MANAGEMENT GOAL
Milestone-driven refinement pattern
NUMBER OF SUBSCRIPTIONS CONTROLLED
TRAFFIC INJECTION CONTROLLED
QoS DEGRADATION PREVENTED
TRAFFIC DEMAND ESTIMATED
AVAILABLE RESOURCES PER TRAFFIC TRUNK
DYNAMIC TRAFFIC FLUCTUATIONS MANAGED
SUBSCRIPTION LOGIC CONFIGURED
TARGET CRITICAL LEVEL SET
ADMISSION CONTROL SET
QoS COMMITTED
CONGESTION SOLVED
MULTIPLEXING FACTORS CONFIGURED
ND CONFIGURED
DYNAMIC RESOURCE MANAGEMENT CONFIGURED
Traffic Demand Estimated
Almost and Fully Satisfied rates are 100% the subscribed ones for real-time traffic
Almost and Fully satisfied rates are 60% and 85% respectivelly for other type of traffic
(Goal) SERVICE MANAGEMENT GOAL GRAPH All-F2
SERVICE MANAGEMENT CONFIGURED
milestone-driven refinement pattern
SERVICE SUBSCRIPTION CONFIGURED
SERVICE INVOCATION CONFIGURED
MULTIPLEXING ESTIMATION FACTORS CONFIGURED
SATISFACTION LEVEL SET
SUBSCRIPTION ADMISSION CONTROLLED
CRITICAL LEVEL SET
INVOCATION ADMISSION CONTROL SET
QoS COMMITMENT CONFIGURED
CONGESTION STATE SOLVED
ALMOST SATISFIED FACTORS CONFIGURED
SUBSCRIPTION THRESHOLD SET
ANTICIPATED DEMAND CONSIDERATIONS
MAXIMUM PRECAUTIONS TAKEN
SERVICE RATE CHANGED
ADMISSION CONTROL MODIFIED
FULLY SATISFIED FACTORS CONFIGURED
CONSERVATIVE SATISFACTION SET
MEDIUM PRECAUTIONS TAKEN
MODERATE SATISFACTION SET
ALMOST SATISFIED CONSIDERATIONS
NO GUARANTEES FOR SATISFACTION
FULLY SATISFIED CONSIDERATIONS
MINIMUM PRECAUTIONS TAKEN
MILD PROACTIVE ACTIONS
SEVERE PROACTIVE ACTIONS
PRIME PROACTIVE ACTIONS

Figure 47. Selection for controlling QoS degradation and traffic estimations

For the High-level Goal *Available Resources per-Traffic Trunk Calculated*, the system guides the consultant to the ND Configured goal, which has been elaborated in the Network Dimensioning Configured goal graph. The pattern of selected goals shown in the Figure 48 specialises the "particular" view of the consultant with this regard: (1) the bottom-left part shows the selection that specialises the sub-view "Allocate minimum demand with average hop-count estimations", i.e. "Average Estimated" selection for the "Delay and Loss Estimated" goal; (2) the sub-view "Minimize link overloading" is in turn covered with the selection of "Minimise Links Overloaded" in the sub-tree defined by the "Load Network Compromised" sub-goal; (3) finally, the sub-view "Redistribute over-provisioning and spare capacity proportionally" is specialised by the selection "Split Proportionally" and "Proportionally Redistributed" respectively for the goals "Spare Capacity Allocated" and "Over Provisioning re-Allocated", both refinements of the "Extra Capacity Processed" goal.

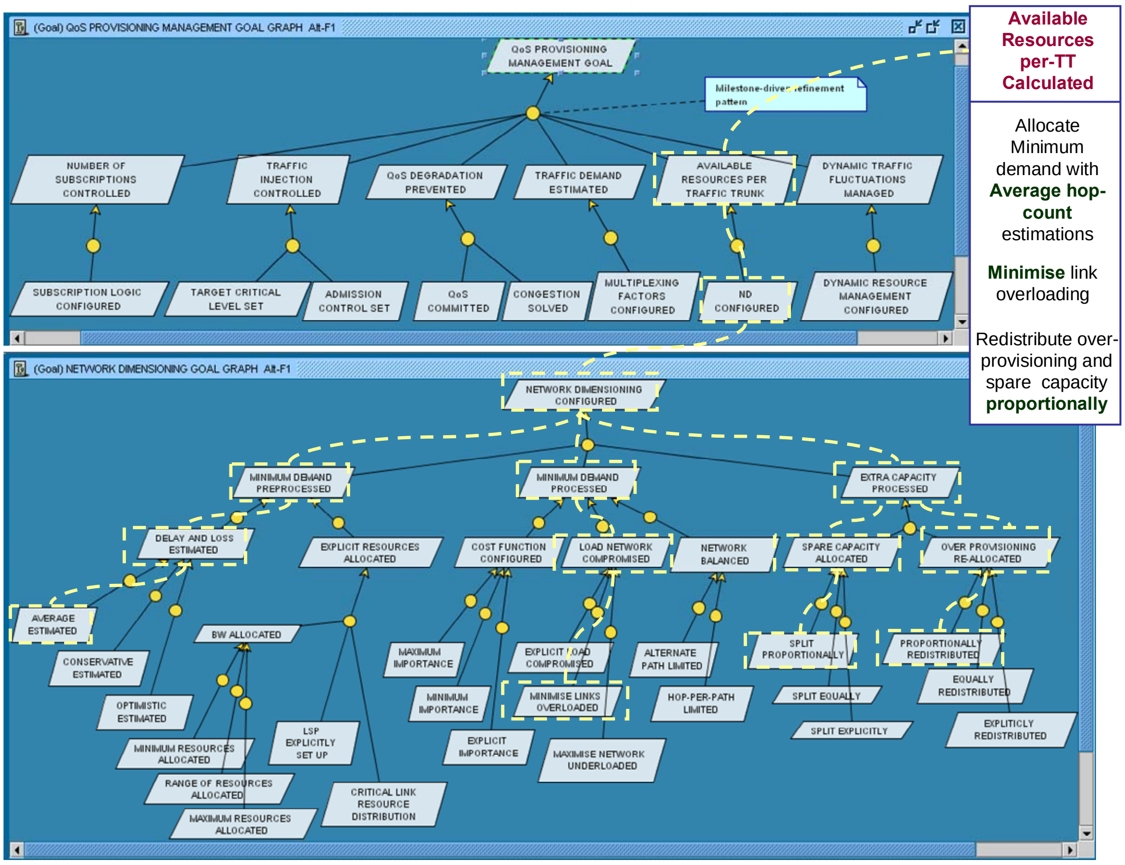

(Goal) QoS PROVISIONING MANAGEMENT GOAL GRAPH Alt-F1
QoS PROVISIONING MANAGEMENT GOAL
Milestone-driven refinement pattern
NUMBER OF SUBSCRIPTIONS CONTROLLED
TRAFFIC INJECTION CONTROLLED
QoS DEGRADATION PREVENTED
TRAFFIC DEMAND ESTIMATED
AVAILABLE RESOURCES PER TRAFFIC TRUNK
DYNAMIC TRAFFIC FLUCTUATIONS MANAGED
SUBSCRIPTION LOGIC CONFIGURED
TARGET CRITICAL LEVEL SET
ADMISSION CONTROL SET
QoS COMMITTED
CONGESTION SOLVED
MULTIPLEXING FACTORS CONFIGURED
ND CONFIGURED
DYNAMIC RESOURCE MANAGEMENT CONFIGURED
(Goal) NETWORK DIMENSIONING GOAL GRAPH Alt-F1
NETWORK DIMENSIONING CONFIGURED
MINIMUM DEMAND PREPROCESSED
MINIMUM DEMAND PROCESSED
EXTRA CAPACITY PROCESSED
DELAY AND LOSS ESTIMATED
EXPLICIT RESOURCES ALLOCATED
COST FUNCTION CONFIGURED
LOAD NETWORK COMPROMISED
NETWORK BALANCED
SPARE CAPACITY ALLOCATED
OVER PROVISIONING RE-ALLOCATED
AVERAGE ESTIMATED
CONSERVATIVE ESTIMATED
OPTIMISTIC ESTIMATED
BW ALLOCATED
MAXIMUM IMPORTANCE
MINIMUM IMPORTANCE
EXPLICIT LOAD COMPROMISED
MINIMISE LINKS OVERLOADED
ALTERNATE PATH LIMITED
HOP-PER-PATH LIMITED
SPLIT PROPORTIONALLY
SPLIT EQUALLY
SPLIT EXPLICITLY
PROPORTIONALLY REDISTRIBUTED
EQUALLY REDISTRIBUTED
EXPLITICLY REDISTRIBUTED
MINIMUM RESOURCES ALLOCATED
RANGE OF RESOURCES ALLOCATED
MAXIMUM RESOURCES ALLOCATED
LSP EXPLICITLY SET UP
CRITICAL LINK RESOURCE DISTRIBUTION
EXPLICIT IMPORTANCE
MAXIMISE NETWORK UNDERLOADED
Available Resources per-TT Calculated
Allocate Minimum demand with **Average hop-count** estimations
Minimise link overloading
Redistribute over-provisioning and spare capacity **proportionally**

Figure 48. Selection for "Available Resources per-TT Calculated"

Finally for the High-level Goal *Dynamic Traffic Fluctuations Managed*, the system guides the consultant to the graph elaborated for the "DRsM Configured" goal. The selection for this goal with respect to the "particular" view of our scenario is shown with dotted lines in Figure 49. The goal graph suggests that the management of traffic fluctuations should involve monitoring and re-allocation tasks, both assessed by the sub-trees defined by the "Monitoring Directives Configured" and "DRsM Main Configured" goals. The selections for the former target the configuration of thresholds to react to statistical fluctuations of traffic in the core routers' links. In other words, this selection is intended to determine when the system should increase or decrease resources as stated by the first sub-view shown in the upper-right part of Figure 49. Traffic fluctuations are policed either as increments or decrements for which upper and lower threshold selections are required. The consultant selects then the goals "Lwr Thr Incrsd Rel Value", "Lwr Thr Decrsd Rel Value", "Uppr Thr Incrsd Rel Value", "Uppr Thr Decrsd Rel Value". The latter goals are associated with an attribute specifying the relative value of configuration as prescribed by the sub-view (10% of the range provided by the ND service).

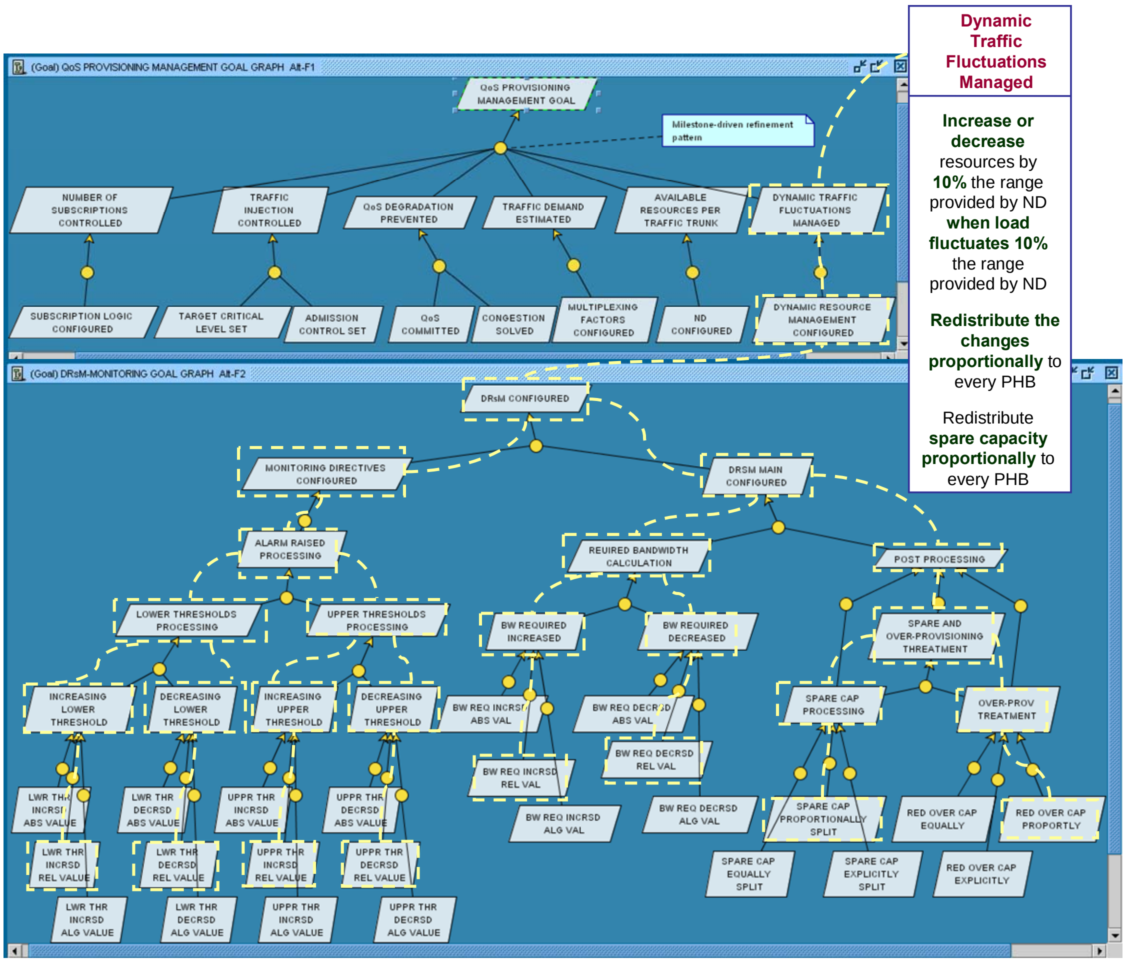

(Goal) QoS PROVISIONING MANAGEMENT GOAL GRAPH Alt-F1
QoS PROVISIONING MANAGEMENT GOAL
Milestone-driven refinement pattern
NUMBER OF SUBSCRIPTIONS CONTROLLED
TRAFFIC INJECTION CONTROLLED
QoS DEGRADATION PREVENTED
TRAFFIC DEMAND ESTIMATED
AVAILABLE RESOURCES PER TRAFFIC TRUNK
DYNAMIC TRAFFIC FLUCTUATIONS MANAGED
SUBSCRIPTION LOGIC CONFIGURED
TARGET CRITICAL LEVEL SET
ADMISSION CONTROL SET
QoS COMMITTED
CONGESTION SOLVED
MULTIPLEXING FACTORS CONFIGURED
ND CONFIGURED
DYNAMIC RESOURCE MANAGEMENT CONFIGURED
Dynamic Traffic Fluctuations Managed
Increase or decrease resources by 10% the range provided by ND when load fluctuates 10% the range provided by ND
Redistribute the changes proportionally to every PHB
Redistribute spare capacity proportionally to every PHB
(Goal) DRsM-MONITORING GOAL GRAPH Alt-F2
DRsM CONFIGURED
MONITORING DIRECTIVES CONFIGURED
DRSM MAIN CONFIGURED
ALARM RAISED PROCESSING
REUIRED BANDWIDTH CALCULATION
POST PROCESSING
LOWER THRESHOLDS PROCESSING
UPPER THRESHOLDS PROCESSING
BW REQUIRED INCREASED
BW REQUIRED DECREASED
SPARE AND OVER-PROVISIONING THREATMENT
INCREASING LOWER THRESHOLD
DECREASING LOWER THRESHOLD
INCREASING UPPER THRESHOLD
DECREASING UPPER THRESHOLD
BW REQ INCRSD ABS VAL
BW REQ DECRSD ABS VAL
SPARE CAP PROCESSING
OVER-PROV TREATMENT
BW REQ INCRSD REL VAL
BW REQ DECRSD REL VAL
LWR THR INCRSD ABS VALUE
LWR THR DECRSD ABS VALUE
UPPR THR INCRSD ABS VALUE
UPPR THR DECRSD ABS VALUE
BW REQ INCRSD ALG VAL
BW REQ DECRSD ALG VAL
SPARE CAP PROPORTIONALLY SPLIT
RED OVER CAP EQUALLY
RED OVER CAP PROPORTLY
LWR THR INCRSD REL VALUE
LWR THR DECRSD REL VALUE
UPPR THR INCRSD REL VALUE
UPPR THR DECRSD REL VALUE
SPARE CAP EQUALLY SPLIT
SPARE CAP EXPLICITLY SPLIT
RED OVER CAP EXPLICITLY
LWR THR INCRSD ALG VALUE
LWR THR DECRSD ALG VALUE
UPPR THR INCRSD ALG VALUE
UPPR THR DECRSD ALG VALUE

Figure 49. Selection for “Dynamic Traffic Fluctuations Managed”

For the sub-views concerning reallocation of resources, the consultant is guided to achieve the selection to increase the resources for the PHBs sharing the core router’s links by selecting the “Bw Req Incrsd Rel Val” goal, and also to deduce these resources from the complementary PHBs sharing each core router’s link by selecting the “Bw Req Decrasd Rel Val” goal. Both goals are associated with an attribute specifying the relative value of these modifications as prescribed by the sub-view (10% of the range provided by the ND service). For the sub-views concerning the redistribution of resources, the consultant is guided to achieve this proportionally amongst the PHBs sharing the core router links by selecting the goals “Spare Cap Proportionally Split” and “Red Over Cap Proptly”.

In this sub-Section we have outlined a complete Goal Selection process for a realistic view of QoS Management. In a policy creation environment the target prototype should enable this view to be effectively translated into enforceable policies aligned to these High-level Goals. This process of our application scenario is described in the following sections.

5.6 Automated Acquisition of Enforceable Policies

This Section describes the automated acquisition of enforceable policies of our application scenario. We initially describe the pre-conditions for this process and then present the execution of the application scenario up to the acquisition of enforceable policies.

5.6.1 Pre-conditions for Automated Policy Acquisition

As we described earlier, the following information should be provided to automate the acquisition of policies. A brief description of these is provided thereafter.

- Documentation of the goal graph structures for the managed system
- Documentation of the System Model

5.6.1.1 Documentation of the goal graph structures

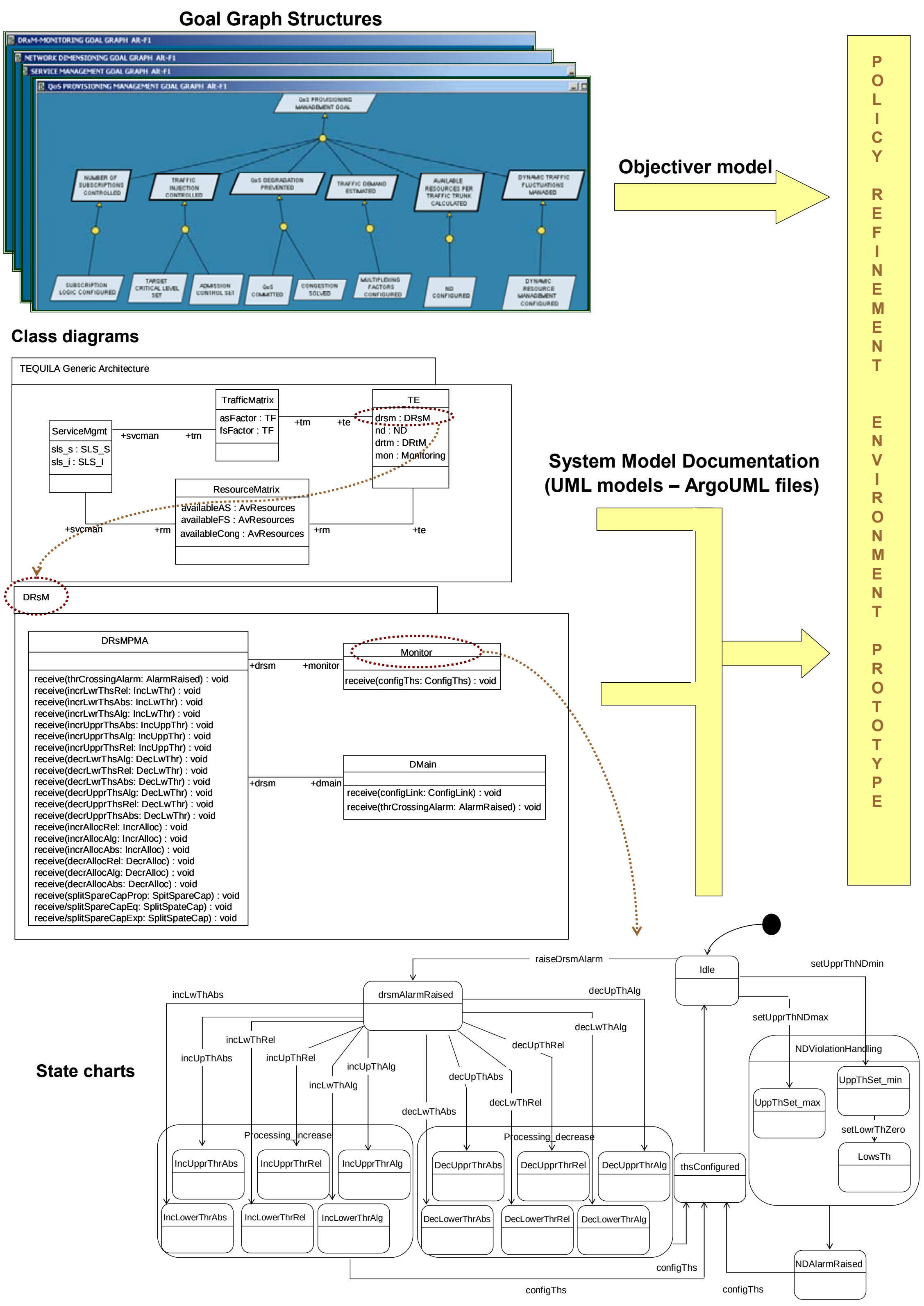
The elaboration of the goal graph structures is a process that has been extensively described in the last sections. Moreover, it is mandatory that this information should be available when the automated acquisition of policies action is required. For the execution of our policy refinement scenario we have provided the Objectiver [Obj] model that corresponds to the KAOS goal graphs “QoS Provisioning Management”, “Service Management”, “Network Dimensioning” and “DRsM Monitoring” described in Section 5.4. This scenario of preconditions is graphically illustrated in the upper part of Figure 50.

5.6.1.2 Documentation of the System Model

The System Model is also a mandatory pre-condition to automate the acquisition of policies. This is assessed by documenting the managed objects’ behaviour and providing their object distribution within the managed system.

5.6.1.2.1 Documentation of the behaviour of the managed objects

This data is provided using standard UML notations. This is an added value of our solution given that the Administrator Developer is enabled to document the System Model using standard modelling techniques such as class diagrams, collaboration diagrams and state charts. For example, the left-lower part of Figure 50 shows a simplified representation of a selection of class diagrams of the TEQUILA architecture together with a simplified representation of the behaviour of the DRsM Monitoring components of the TE functionality, represented as a state chart. For instance, the left part of the state chart shows the different strategies with which the component calculates new thresholds for statistical fluctuations of traffic; strategies are represented as states for which different actions should be taken to lead the monitoring component to the corresponding states. The right part of the state chart demonstrates for example the handling of violations when the constraints imposed by the Network Dimensioning component are breached. Similar representations to the above have been modeled for the rest TE components and the Service Management components of TEQUILA. ArgoUML [Arg06] models are direct inputs for our policy refinement prototype. The Behaviour Manager component of our solution internally translates these into code for automated reactive analysis.



Goal Graph Structures
DRsM-MONITORING GOAL GRAPH AR-F1
NETWORK DIMENSIONING GOAL GRAPH AR-F1
SERVICE MANAGEMENT GOAL GRAPH AR-F1
QoS PROVISIONING MANAGEMENT GOAL GRAPH AR-F1
QoS PROVISIONING MANAGEMENT GOAL
NUMBER OF SUBSCRIPTIONS CONTROLLED
TRAFFIC INJECTION CONTROLLED
QoS DEGRADATION PREVENTED
TRAFFIC DEMAND ESTIMATED
AVAILABLE RESOURCES PER TRAFFIC TRUNK CALCULATED
DYNAMIC TRAFFIC FLUCTUATIONS MANAGED
SUBSCRIPTION LOGIC CONFIGURED
TARGET CRITICAL LEVEL SET
ADMISSION CONTROL SET
QoS COMMITTED
CONGESTION SOLVED
MULTIPLEXING FACTORS CONFIGURED
ND CONFIGURED
DYNAMIC RESOURCE MANAGEMENT CONFIGURED
Objectiver model
POLICY REFINEMENT ENVIRONMENT PROTOTYPE
Class diagrams
TEQUILA Generic Architecture
TrafficMatrix
asFactor : TF
fsFactor : TF
TE
drsm : DRsM
nd : ND
drtm : DRtM
mon : Monitoring
ServiceMgmt
sls_s : SLS_S
sls_i : SLS_I
+svcman
+tm
+tm
+te
ResourceMatrix
availableAS : AvResources
availableFS : AvResources
availableCong : AvResources
+svcman
+rm
+rm
+te
System Model Documentation
(UML models – ArgoUML files)
DRsM
DRsMPMA
receive(thrCrossingAlarm: AlarmRaised) : void
receive(incrLwrThsRel: IncLwThr) : void
receive(incrLwrThsAbs: IncLwThr) : void
receive(incrLwrThsAlg: IncLwThr) : void
receive(incrUpprThsAbs: IncUppThr) : void
receive(incrUpprThsAlg: IncUppThr) : void
receive(incrUpprThsRel: IncUppThr) : void
receive(decrLwrThsAlg: DecLwThr) : void
receive(decrLwrThsRel: DecLwThr) : void
receive(decrLwrThsAbs: DecLwThr) : void
receive(decrUpprThsAlg: DecLwThr) : void
receive(decrUpprThsRel: DecLwThr) : void
receive(decrUpprThsAbs: DecLwThr) : void
receive(incrAllocRel: IncrAlloc) : void
receive(incrAllocAlg: IncrAlloc) : void
receive(incrAllocAbs: IncrAlloc) : void
receive(decrAllocRel: DecrAlloc) : void
receive(decrAllocAlg: DecrAlloc) : void
receive(decrAllocAbs: DecrAlloc) : void
receive(splitSpareCapProp: SplitSpareCap) : void
receive/splitSpareCapEq: SplitSpateCap) : void
receive/splitSpareCapExp: SplitSpateCap) : void
+drsm
+monitor
Monitor
receive(configThs: ConfigThs) : void
+drsm
+dmain
DMain
receive(configLink: ConfigLink) : void
receive(thrCrossingAlarm: AlarmRaised) : void
State charts
raiseDrsmAlarm
Idle
setUpprThNDmin
drsmAlarmRaised
incLwThAbs
decUpThAlg
setUpprThNDmax
incLwThRel
decLwThAlg
incUpThAbs
incUpThRel
decUpThRel
NDViolationHandling
incUpThAlg
incLwThAlg
decUpThAbs
UppThSet_min
decLwThRel
UppThSet_max
decLwThAbs
setLowrThZero
Processing_increase
Processing_decrease
LowsTh
IncUpprThrAbs
IncUpprThrRel
IncUpprThrAlg
DecUpprThrAbs
DecUpprThrRel
DecUpprThrAlg
thsConfigured
IncLowerThrAbs
IncLowerThrRel
IncLowerThrAlg
DecLowerThrAbs
DecLowerThrRel
DecLowerThrAlg
configThs
NDAlarmRaised
configThs
configThs

Figure 50. Pre-conditions for Automated Policy Acquisition

5.6.1.2.2 Documentation of the Object Distribution

The Object Distribution, part of the System Model, is also a mandatory pre-condition for the automated acquisition of policies. It is necessary for the appropriate operation of the system and to make sure that the refined policies to be appropriately deployed onto the policy system.

The principle for the formalisation of this information in the context of TEQUILA approach is that enforceable policies are deployed onto two main branches of objects; centralised managers represented by the Domain Scope Expression /TEQUILA/Managers and distributed managed objects represented by the Domain Scope Expression /TEQUILA/ManagedObjects. We have populated the inventory component with a very basic distribution coping with the functionality ad-hoc to the execution of our application scenario. The basic distribution of these two main branches is shown in Figure 51. The centralised managers are those components achieving the Network Dimensioning Sub-function and the Service Subscription Sub-function of TEQUILA, both formalised with as /TEQUILA/Managers/ND and /TEQUILA/Managers/SSM respectively. The components assessing the Dynamic Resource Management Sub-function pertain to the /TEQUILA/ManagedObjects/Links/Router branch. Similarly, the objects achieving the Service Invocation Sub-function are included in the /TEQUILA/ManagedObjects/Routers domain.

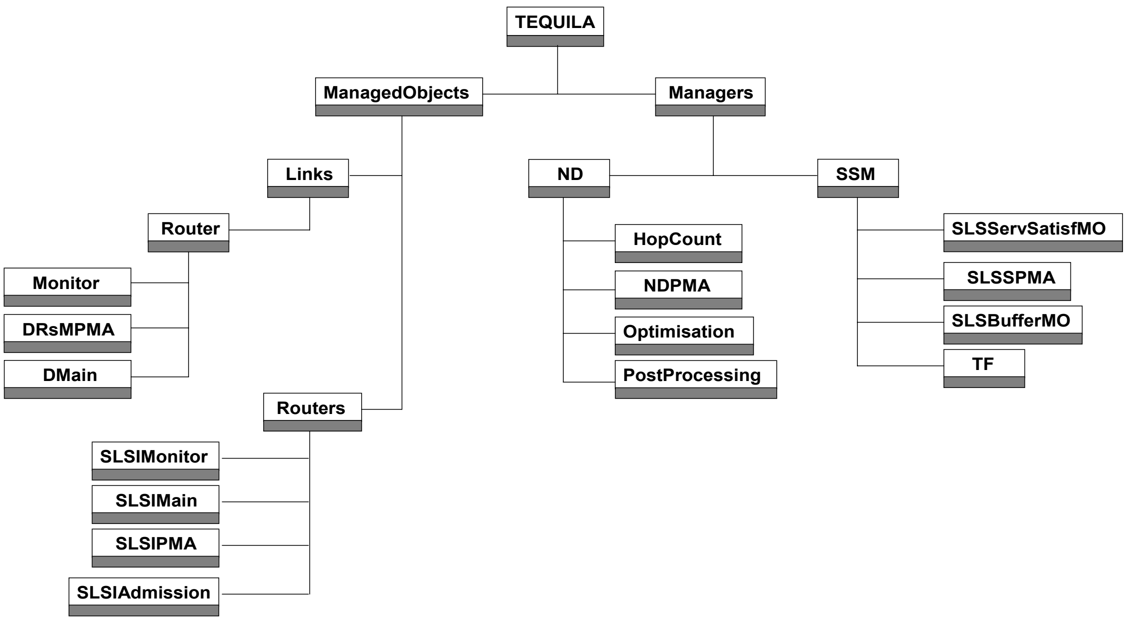
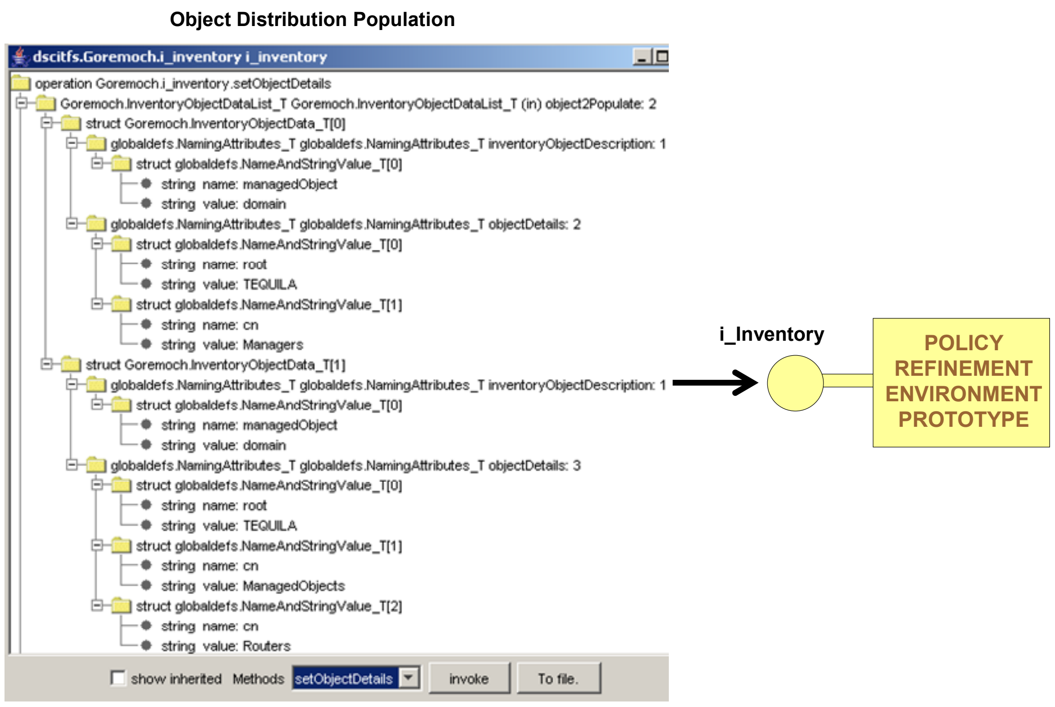


Figure 51. Object Distribution for our refinement scenario

For this application scenario we have documented this information making use of the inventory component interface. For example, the information introduced to the i_Inventory interface shown in Figure 52 formalises two basic domains of our scenario;

TEQUILA/Managers and TEQUILA/ManagedObjects/Routers. A similar approach has been carried out to populate the rest branches of the distribution shown in Figure 51.


Object Distribution Population
i_Inventory
POLICY REFINEMENT ENVIRONMENT PROTOTYPE
show inherited
Methods
setObjectDetails
invoke
To file.

```
dscitfs.Goremoch.i_inventory i_inventory
operation Goremoch.i_inventory.setObjectDetails
  Goremoch.InventoryObjectDataList_T Goremoch.InventoryObjectDataList_T (in) object2Populate: 2
    struct Goremoch.InventoryObjectData_T[0]
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T inventoryObjectDescription: 1
        struct globaldefs.NameAndStringValue_T[0]
          string name: managedObject
          string value: domain
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T objectDetails: 2
        struct globaldefs.NameAndStringValue_T[0]
          string name: root
          string value: TEQUILA
        struct globaldefs.NameAndStringValue_T[1]
          string name: cn
          string value: Managers
    struct Goremoch.InventoryObjectData_T[1]
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T inventoryObjectDescription: 1
        struct globaldefs.NameAndStringValue_T[0]
          string name: managedObject
          string value: domain
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T objectDetails: 3
        struct globaldefs.NameAndStringValue_T[0]
          string name: root
          string value: TEQUILA
        struct globaldefs.NameAndStringValue_T[1]
          string name: cn
          string value: ManagedObjects
        struct globaldefs.NameAndStringValue_T[2]
          string name: cn
          string value: Routers
```

Figure 52. Documentation of managed Object Distribution

5.6.2 Managing Goal Selections

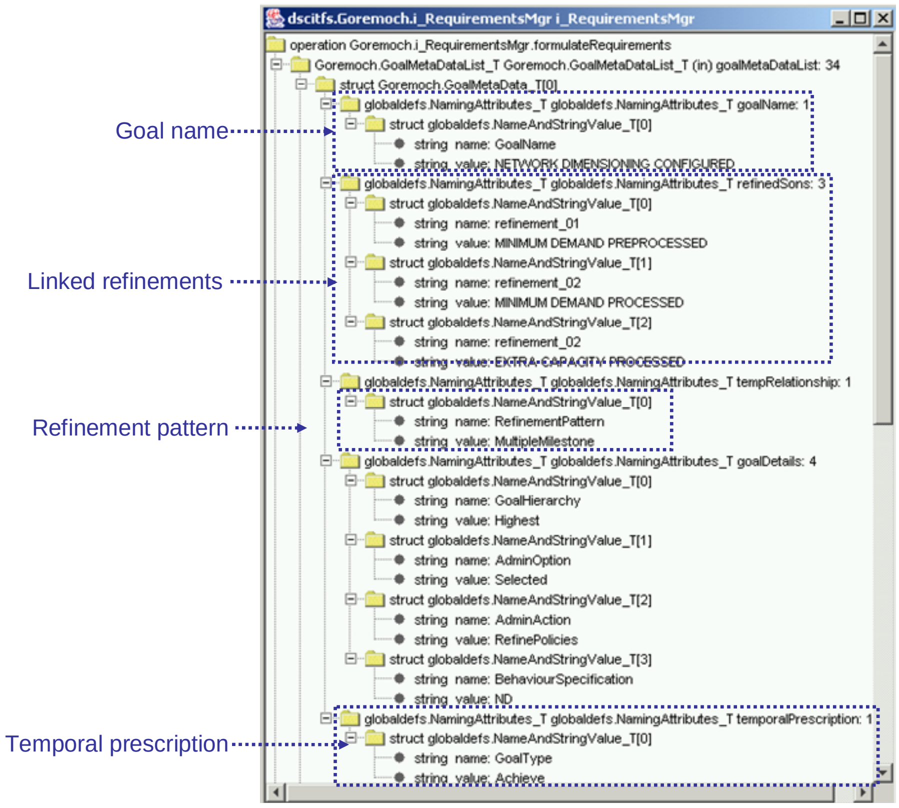
The automated acquisition of policies starts when Goal Selections are provided to the implementation prototype by the Administrator Consultant. The selections could be provided in two fashions:

1. *From the Objectiver tookit*. The selection of goals is documented in the Objectiver toolkit and an internal application (plug-in) is in charge of providing the selection to the prototype implementation as generic Objectiver objects through the i_GoalManager interface.
2. *From the Goal Manager*. The selection of goals is also documented in the Objectiver toolkit but the acquisition of policies is started in the Goal Manager by pointing out the highest-level goal of the goal selection.

For the execution of our application scenario we have opted for the second option. Having documented the Goal Selection described in Section 5.5 which defines the operative or “particular” view of QoS provisioning, we have pointed out the “QoS Provisioning Management Goal” to the Goal Manager through the i_GoalManager interface. It is worth mentioning that large-scale refinements could involve several Goal Selections committing to several views of QoS Management, most probably involving different constraints or situations of applicability. For this application scenario we have only provided the Goal Selection of our application scenario.

For the above Goal Selection, the prototype implementation integrates the Goal-aware information that it will use to acquire enforceable policies aligned to such selection. This Goal-aware information is integrated in an object-oriented entity referred to as "GoalMetaData_List". The following is a brief description of the "GoalMetaData_List" acquired during the execution of our application scenario. The fragments shown throughout this brief explanation have been obtained making use of the facilities provided by the Distributed Software Component (DSC) development framework [Mee00] on which our prototype was implemented.

The execution of the application scenario has involved altogether the goal graph structures for the QoS Provisioning Management, Service Management, Network Dimensioning, and DRsM-Monitoring. The Figure 53 shows for example a fragment of the GoalMetaData_List structure that belongs to the Network Dimensioning Goal Graph. This fragment shows the object-oriented composition of the "NETWORK DIMENSIONING CONFIGURED" goal (see goal name pointer in Fig. 53). The composition is integrated by relevant information that is further used to establish relationships amongst the goals. For example, the "Achieve" temporal prescription (see temporal prescription pointer in Fig. 53), and the "MultipleMilestone" refinement pattern (see refinement pattern pointer in Fig. 53), prescribe that the NETWORK DIMENSIONING CONFIGURED goal should be achieved in multiple phases. The refinements of this particular goal are also included in the composition (see Link refinements pointer in Fig. 53). In this particular case of our application scenario the refinements are the goals MINIMUM DEMAND PREPROCESSED, MINIMUM DEMAND PROCESSED, and EXTRA CAPACITY PROCESSED. Similar compositions for the latter refinements and for the rest goals belonging to the Network Dimensioning goal-graph have been integrated in the "GoalMetaData_List" acquired during the execution of our application scenario.

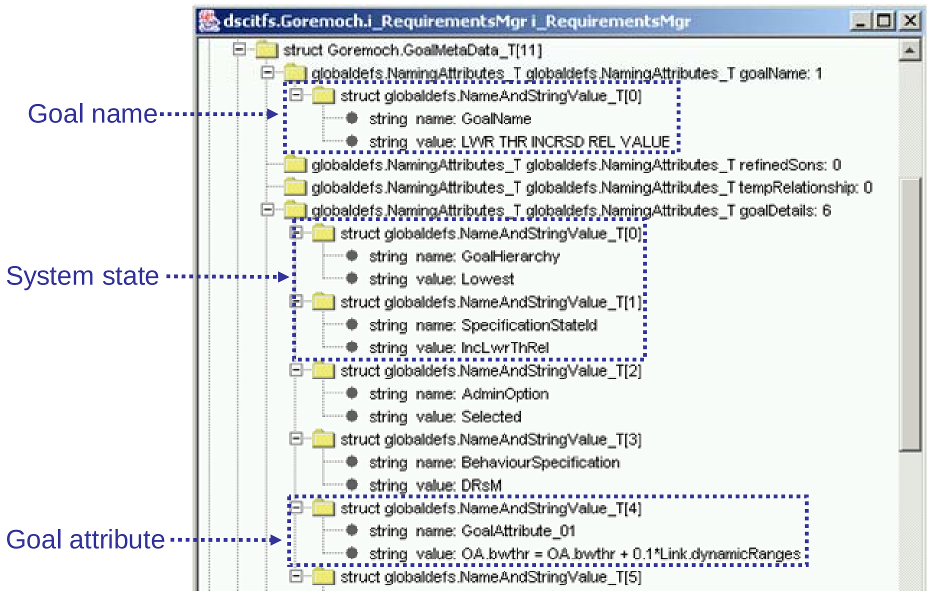


```
operation Goremoch.i_RequirementsMgr.formulateRequirements
  Goremoch.GoalMetaDataList_T Goremoch.GoalMetaDataList_T (in) goalMetaDataList: 34
    struct Goremoch.GoalMetaData_T[0]
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T goalName: 1
        struct globaldefs.NameAndStringValue_T[0]
          string  name: GoalName
          string  value: NETWORK DIMENSIONING CONFIGURED
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T refinedSons: 3
        struct globaldefs.NameAndStringValue_T[0]
          string  name: refinement_01
          string  value: MINIMUM DEMAND PREPROCESSED
        struct globaldefs.NameAndStringValue_T[1]
          string  name: refinement_02
          string  value: MINIMUM DEMAND PROCESSED
        struct globaldefs.NameAndStringValue_T[2]
          string  name: refinement_02
          string  value: EXTRA CAPACITY PROCESSED
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T tempRelationship: 1
        struct globaldefs.NameAndStringValue_T[0]
          string  name: RefinementPattern
          string  value: MultipleMilestone
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T goalDetails: 4
        struct globaldefs.NameAndStringValue_T[0]
          string  name: GoalHierarchy
          string  value: Highest
        struct globaldefs.NameAndStringValue_T[1]
          string  name: AdminOption
          string  value: Selected
        struct globaldefs.NameAndStringValue_T[2]
          string  name: AdminAction
          string  value: RefinePolicies
        struct globaldefs.NameAndStringValue_T[3]
          string  name: BehaviourSpecification
          string  value: ND
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T temporalPrescription: 1
        struct globaldefs.NameAndStringValue_T[0]
          string  name: GoalType
          string  value: Achieve
```

Figure 53. Fragment of the NETWORK DIMENSIONING CONFIGURED goal

Following on with the content of the "GoalMetaData_List" of our application scenario, the prototype handles the graph's relevant information of the QoS Provisioning Management, Service Management, and DRsM-Monitoring. This information contains the information necessary to automate the acquisition of enforceable policies. For example, during the goal selection, some goals admit specific values ad-hoc to the administrative criteria. This is the case for example of the goal "LWR THR INCRSD REL VALUE" which has been selected as part of the High-level Goal "Dynamic Traffic Fluctuations Managed" in our application scenario. A fragment of this goal composition is shown in Figure 54. Here the ad-hoc value is integrated in the GoalAttribute field of the composition (see goal attribute pointer in Fig. 54). The latter specialises the relative value with which the threshold settings will be increased in the DRsM Sub-function of the TEQUILA approach. In addition, lowest-level goals are identified with system state predicates. This information is also handled for the "LWR THR INCRSD REL VALUE" (see system state pointer in Fig 54), and for the rest lowest-level goals of the application scenario.

Similar to the two examples provided above, the prototype has produced a total of 118 goal compositions for our application scenario. The following Section provides the execution of the Requirements Manager once this information is submitted through its i_RequirementsMgr interface.



```
dscitfs.Goremoch.i_RequirementsMgr i_RequirementsMgr
⊟ struct Goremoch.GoalMetaData_T[11]
   ⊟ globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T goalName: 1
      ⊟ struct globaldefs.NameAndStringValue_T[0]
           ● string  name: GoalName
           ● string  value: LWR THR INCRSD REL VALUE
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T refinedSons: 0
      globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T tempRelationship: 0
   ⊟ globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T goalDetails: 6
      ⊟ struct globaldefs.NameAndStringValue_T[0]
           ● string  name: GoalHierarchy
           ● string  value: Lowest
      ⊟ struct globaldefs.NameAndStringValue_T[1]
           ● string  name: SpecificationStateId
           ● string  value: IncLwrThRel
      ⊟ struct globaldefs.NameAndStringValue_T[2]
           ● string  name: AdminOption
           ● string  value: Selected
      ⊟ struct globaldefs.NameAndStringValue_T[3]
           ● string  name: BehaviourSpecification
           ● string  value: DRsM
      ⊟ struct globaldefs.NameAndStringValue_T[4]
           ● string  name: GoalAttribute_01
           ● string  value: OA.bwthr = OA.bwthr + 0.1*Link.dynamicRanges
      ⊟ struct globaldefs.NameAndStringValue_T[5]
```

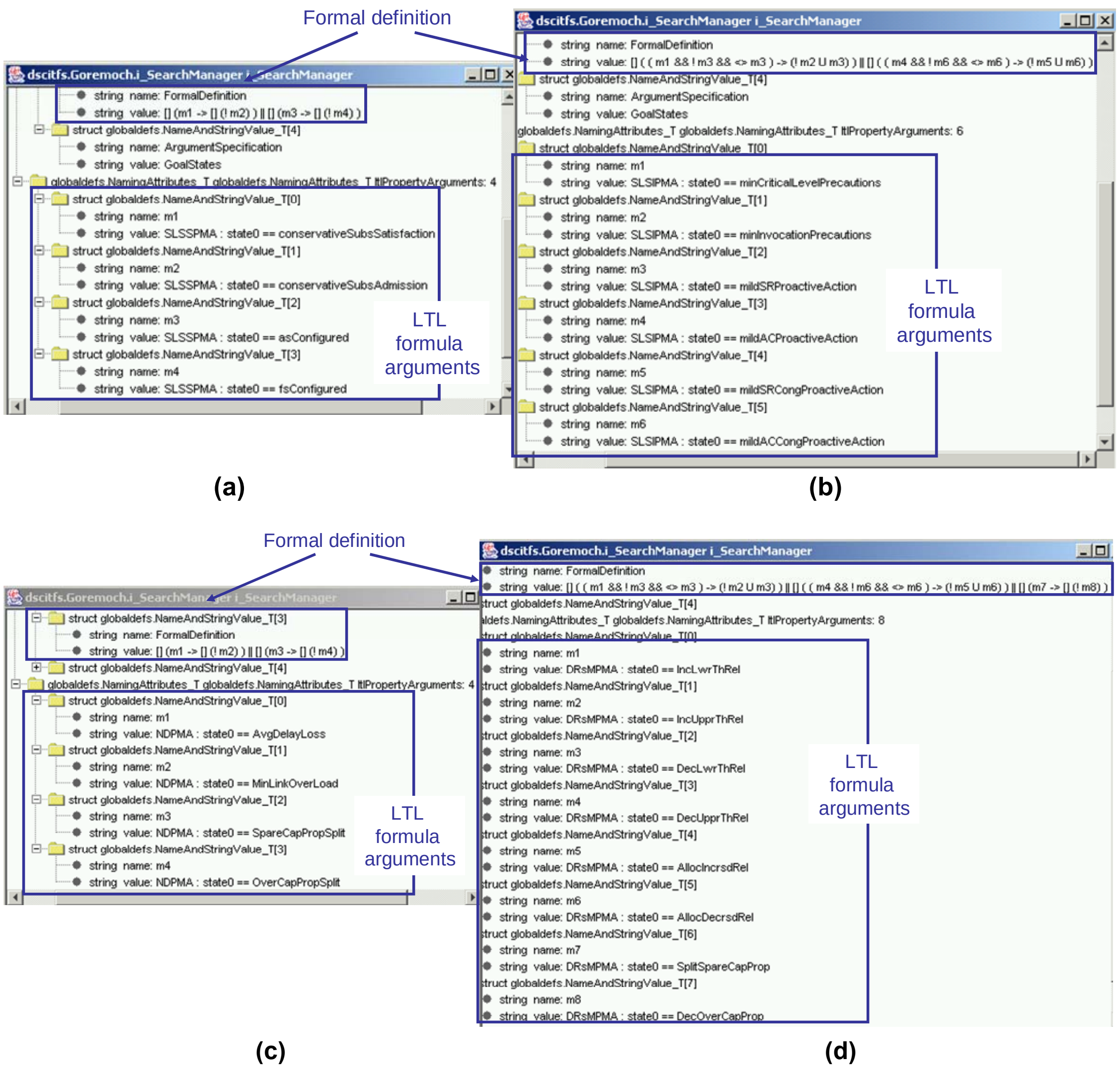
Figure 54. Fragment of the LWR THR INCRSD REL VALUE goal

5.6.3 Establishing Temporal Relationships

Coming back to the principles of our refinement framework, the Requirements Manager uses the information included in the "GoalMetaData_List" structures to establish temporal relationships between lowest-level goals. These relationships logically entail the fulfillment of higher-level goals. Following on, this manager produces a representation of these relationships with formal notations suitable for use with automated analysis tools.

In Figure 55 we show fragments of the temporal relationships produced in run-time by the Requirements Manager during the execution of our application scenario. The most relevant are the formal definitions of these relationships expressed in Linear Temporal Logic (see formal definition pointers in Fig 55). In practical terms, these LTL (Linear Temporal Logic) behavioural formulae represent the ordering in the fulfillment of lowest-level goals or the ordering in which system states should be reached. In other words, these represent the ordering of system states that the corresponding sub-systems should commit to as to fulfill with the "particular" view of QoS Management of our scenario.

For example, the formal representation shown in Figure 55a (see formal definition pointer in Fig. 55a) characterises the temporal ordering of four specific goal states of the Service Subscription Sub-function of the TEQUILA architecture. These goal states are also abstracted and shown as arguments in the figure (see LTL formula arguments pointer in Fig. 55a). These arguments correspond to the actual system state identifications of the lowest-level goals within the System Model. Similarly, the Requirements Manager produces the representations and arguments for the Service Invocation function (Fig 54b), the Network Dimension (Fig. 55c), and the Dynamic Resource Management function (Fig. 55d) accordingly.



```
dscitfs.Goremoch.i_SearchManager i_SearchManager
    string name: FormalDefinition
    string value: [] (m1 -> [] (! m2) ) || [] (m3 -> [] (! m4) )
  struct globaldefs.NameAndStringValue_T[4]
    string name: ArgumentSpecification
    string value: GoalStates
globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T ltlPropertyArguments: 4
  struct globaldefs.NameAndStringValue_T[0]
    string name: m1
    string value: SLSSPMA : state0 == conservativeSubsSatisfaction
  struct globaldefs.NameAndStringValue_T[1]
    string name: m2
    string value: SLSSPMA : state0 == conservativeSubsAdmission
  struct globaldefs.NameAndStringValue_T[2]
    string name: m3
    string value: SLSSPMA : state0 == asConfigured
  struct globaldefs.NameAndStringValue_T[3]
    string name: m4
    string value: SLSSPMA : state0 == fsConfigured
```

```
dscitfs.Goremoch.i_SearchManager i_SearchManager
    string name: FormalDefinition
    string value: [] ( ( m1 && ! m3 && <> m3 ) -> (! m2 U m3) ) || [] ( ( m4 && ! m6 && <> m6 ) -> (! m5 U m6) )
struct globaldefs.NameAndStringValue_T[4]
    string name: ArgumentSpecification
    string value: GoalStates
globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T ltlPropertyArguments: 6
struct globaldefs.NameAndStringValue_T[0]
    string name: m1
    string value: SLSIPMA : state0 == minCriticalLevelPrecautions
struct globaldefs.NameAndStringValue_T[1]
    string name: m2
    string value: SLSIPMA : state0 == minInvocationPrecautions
struct globaldefs.NameAndStringValue_T[2]
    string name: m3
    string value: SLSIPMA : state0 == mildSRProactiveAction
struct globaldefs.NameAndStringValue_T[3]
    string name: m4
    string value: SLSIPMA : state0 == mildACProactiveAction
struct globaldefs.NameAndStringValue_T[4]
    string name: m5
    string value: SLSIPMA : state0 == mildSRCongProactiveAction
struct globaldefs.NameAndStringValue_T[5]
    string name: m6
    string value: SLSIPMA : state0 == mildACCongProactiveAction
```

```
dscitfs.Goremoch.i_SearchManager i_SearchManager
  struct globaldefs.NameAndStringValue_T[3]
    string name: FormalDefinition
    string value: [] (m1 -> [] (! m2) ) || [] (m3 -> [] (! m4) )
  struct globaldefs.NameAndStringValue_T[4]
globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T ltlPropertyArguments: 4
  struct globaldefs.NameAndStringValue_T[0]
    string name: m1
    string value: NDPMA : state0 == AvgDelayLoss
  struct globaldefs.NameAndStringValue_T[1]
    string name: m2
    string value: NDPMA : state0 == MinLinkOverLoad
  struct globaldefs.NameAndStringValue_T[2]
    string name: m3
    string value: NDPMA : state0 == SpareCapPropSplit
  struct globaldefs.NameAndStringValue_T[3]
    string name: m4
    string value: NDPMA : state0 == OverCapPropSplit
```

```
dscitfs.Goremoch.i_SearchManager i_SearchManager
string name: FormalDefinition
string value: [] ( ( m1 && ! m3 && <> m3 ) -> (! m2 U m3) ) || [] ( ( m4 && ! m6 && <> m6 ) -> (! m5 U m6) ) || [] (m7 -> [] (! m8) )
truct globaldefs.NameAndStringValue_T[4]
ldefs.NamingAttributes_T globaldefs.NamingAttributes_T ltlPropertyArguments: 8
truct globaldefs.NameAndStringValue_T[0]
string name: m1
string value: DRsMPMA : state0 == IncLwrThRel
truct globaldefs.NameAndStringValue_T[1]
string name: m2
string value: DRsMPMA : state0 == IncUpprThRel
truct globaldefs.NameAndStringValue_T[2]
string name: m3
string value: DRsMPMA : state0 == DecLwrThRel
truct globaldefs.NameAndStringValue_T[3]
string name: m4
string value: DRsMPMA : state0 == DecUpprThRel
truct globaldefs.NameAndStringValue_T[4]
string name: m5
string value: DRsMPMA : state0 == AllocIncrsdRel
truct globaldefs.NameAndStringValue_T[5]
string name: m6
string value: DRsMPMA : state0 == AllocDecrsdRel
truct globaldefs.NameAndStringValue_T[6]
string name: m7
string value: DRsMPMA : state0 == SplitSpareCapProp
truct globaldefs.NameAndStringValue_T[7]
string name: m8
string value: DRsMPMA : state0 == DecOverCapProp
```

Figure 55. Fragment of temporal relationships of application scenario

5.6.4 Enforcing System Behaviour

In the context of our refinement framework, the main purpose of abstracting temporal relationships of goals and representing them with formal notations such as LTL, is to enable the Search Manager to apply automatic analysis techniques to find the necessary behaviour that the managed system should exhibit as to commit with the high-level goals adhered to the system. In the context of our application scenario, this corresponds to the system behaviour that the internal components of the Service Subscription, the Service Invocation, Network Dimensioning, and Dynamic Resource Management Sub-functions should exhibit as to commit with our "particular" view of QoS Management.

A brief example is the partial representation of system behaviour shown in the left part of Figure 56, produced by the Search Manager during the execution of our application scenario. This corresponds to the Dynamic Resource Management (DRsM) Sub-function. More concretely, it shows the restricted behaviour that the components assessing the DRsM Sub-function, should exhibit to fulfill the milestone goals m4 “DecUpprThRel” and m5 “AllocIncrsdRel” included in the temporal relationships provided by the Requirements Manager described above.

For instance, in order to better describe the restricted system behaviour of the left-part of Fig. 56, we show its interpretation as state changes in the right-part of the same Fig. 56. For instance, in order for the milestone goal m4 “DecUpprThRel” to be achieved, the following state changes should be exhibited:

- The Monitor component should exhibit a transition from state thresholdCrossingDwn to state thrCrossdDwnAlarmRaised as a result of the event thrsCrossingAlarmDwn (see pointer to thrsCrossingAlarmDwn in Fig. 56). In other words, the Monitor component must detect a downwards threshold crossing and it should issue the event thrsCrossingAlarmDwn.
- As a consequence of this previous event, the DRsMPMA component must enforce the action decrUpprThRel to transition from the drsmAlarmRaised state to the target “DecUpprThRel” state. In other words, the DRsMPMA must enforce decrements of the upper threshold by a relative value.
- Note that the event “notification for decrUpprThRel” generated in the DRsMPMA enables the Monitor component to transition from the state thrCrossdDwnAlarmRaised to UpprThrDecreased.

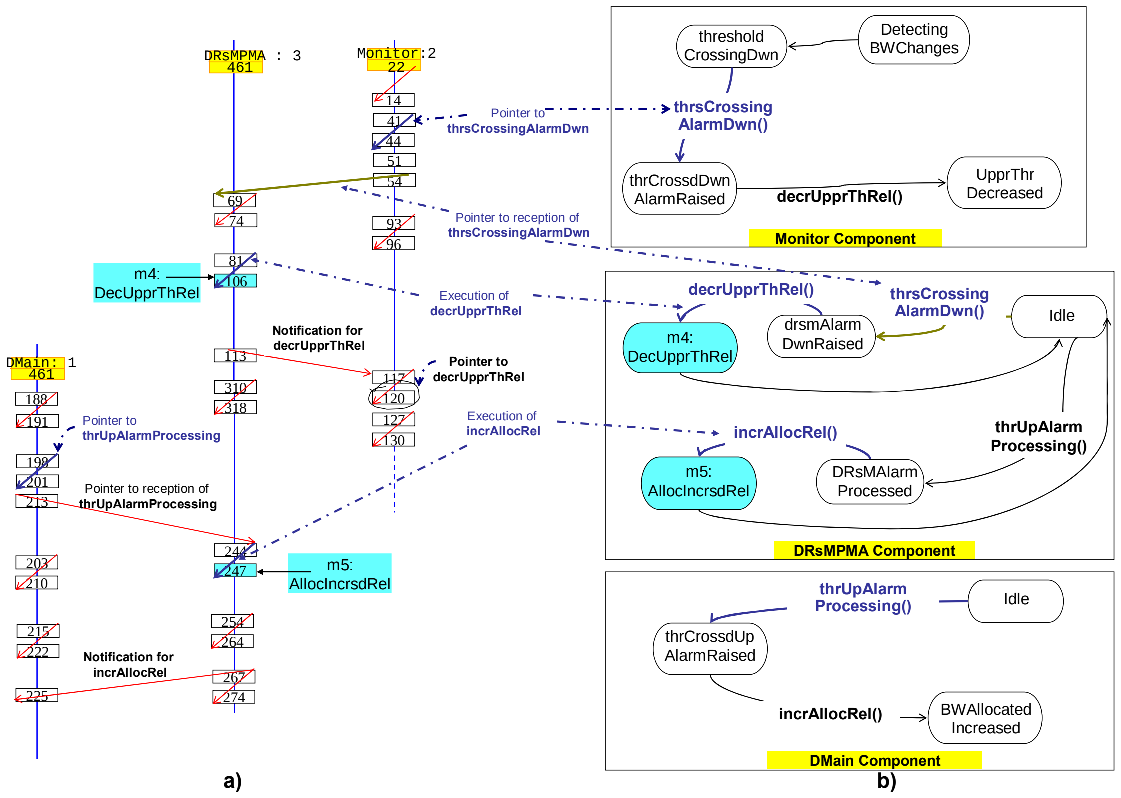


Figure 56. Enforcing system behaviour a) visual report b) interpretation

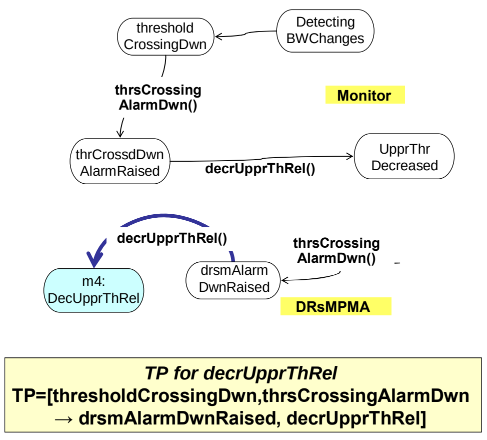
The same applies for the milestone m5 “AllocIncrsdRel”. In this case the DMain component should issue the event thrUpAlarmProcessing for which the DRsMPMA must enforce the incrAllocRel action to commit with the target goal state “AllocIncrsdRel”. In other words, the BW allocation should be re-allocated by a relative value when an upper threshold is crossed.

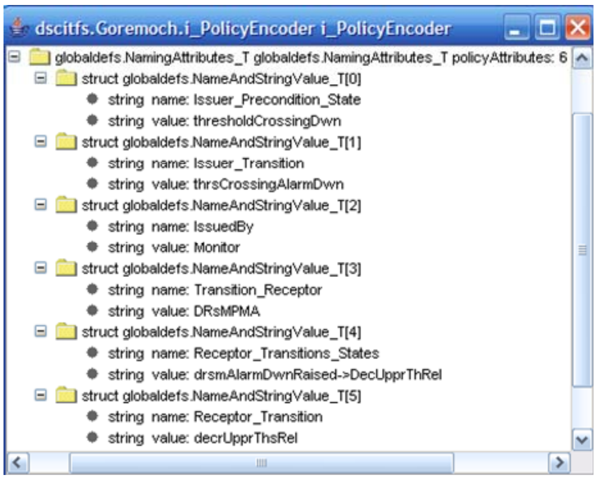
Similarly to m4 and m5, the Search Manager has acquired the restricted behaviour for the remaining milestones of the DRsM Sub-functions and the other sub-functions of our application scenario.

5.6.5 Applying the Translation Process

Following the principles of our policy refinement framework, having obtained the restricted system behaviour that the managed objects should exhibit, the Search Manager takes the first step towards the acquisition of enforceable policies. It identifies the transition plans that should reproduce such restricted behaviour in the policy system and applies a translation process to produce structures of the format Event-Condition-Action. In our framework we defined a transition plan as a sub-section of a system trace execution that includes a policy-controlled state transition. In this Section we show this step of the refinement process for our application scenario.

The Figure 57a shows the transition plan belonging to the policy-controlled transition “decrUpprThRel” which was committed to fulfil the goal state m4 in our previous description. This transition plan prescribes that on the occurrence of a “thresholdCrossingDwn” state in the Monitor component, preceding the transition “thrsCrossingAlarmDwn”, the DRsMPMA must enforce the transition “decrUpprThRel”. This information is specialised in an object-oriented composition by the Search Manager and is represented in Fig. 57b. Basically, the transition plan is specialised into the key attributes “Issuer_Precondition_State, Issuer_Transition, IssuedBy, Transition_Receptor, Receptor_Transitions_States, and Receptor_Transition”. In the transition plan “decrUpprThRel”, the former attributes have been assigned to “thresholdCrossingDwn”, “thrsCrossingAlarmDwn”, “Monitor”, “DRsMPMA”, “drsmAlarmDwnRaised -> DecUpprThRel”, and “decrUpprThsRel” respectively. Similar to this process, the rest of the transition plans of our validation scenario have been produced. It is worth noticing that this information has been acquired automatically by the implementation prototype during the execution of our application scenario.



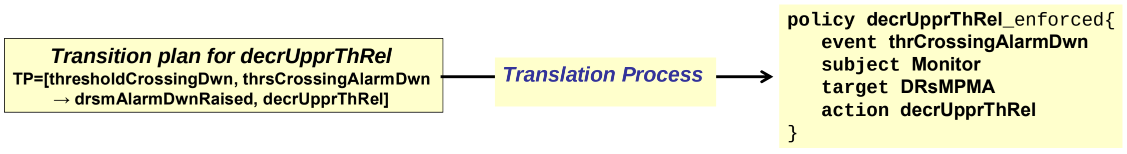


```
dscitfs.Goremoch.i_PolicyEncoder i_PolicyEncoder
globaldefs.NamingAttributes_T globaldefs.NamingAttributes_T policyAttributes: 6
  struct globaldefs.NameAndStringValue_T[0]
    string name: Issuer_Precondition_State
    string value: thresholdCrossingDwn
  struct globaldefs.NameAndStringValue_T[1]
    string name: Issuer_Transition
    string value: thrsCrossingAlarmDwn
  struct globaldefs.NameAndStringValue_T[2]
    string name: IssuedBy
    string value: Monitor
  struct globaldefs.NameAndStringValue_T[3]
    string name: Transition_Receptor
    string value: DRsMPMA
  struct globaldefs.NameAndStringValue_T[4]
    string name: Receptor_Transitions_States
    string value: drsmAlarmDwnRaised->DecUpprThRel
  struct globaldefs.NameAndStringValue_T[5]
    string name: Receptor_Transition
    string value: decrUpprThsRel
```

a) b)

Figure 57. A transition plan of our validation scenario

Following on with the scenario execution in the context of our refinement framework, the information described above is further used to acquire Event-Condition-Action (ECA) obligation policies. For this, the Translation Process considers that the managed object that exhibited the triggering state transition is mapped to the subject; that Actions are interpreted as transitions executed on a managed object; and that the managed object that reacted to the triggering condition is mapped to the target. A graphical representation of the Translation Process and their application to the transition plan "decrUpprThRel" of our scenario are shown in Figure 58. In a similar manner, the rest of the transition plans of the DRsM Sub-function and the rest Sub-functions of our application scenario have been processed automatically by the prototype implementation.


Transition plan for decrUpprThRel
TP=[thresholdCrossingDwn, thrsCrossingAlarmDwn → drsmAlarmDwnRaised, decrUpprThRel]
Translation Process

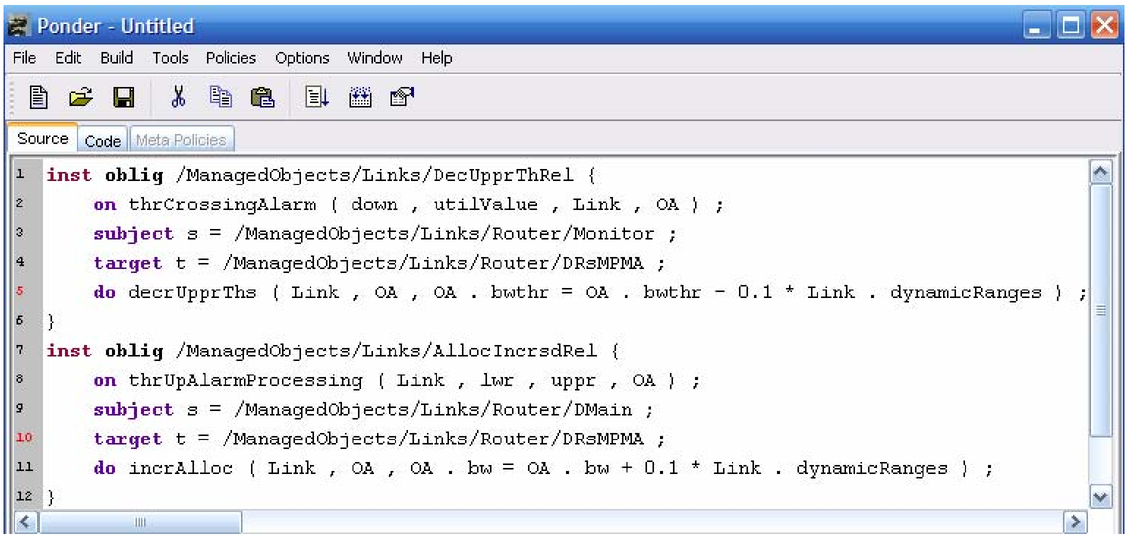
```
policy decrUpprThRel_enforced{
   event thrCrossingAlarmDwn
   subject Monitor
   target DRsMPMA
   action decrUpprThRel
}
```

Figure 58. A translation process in our application scenario

5.6.6 Encoding Policies

Finally, the Policy Encoder produces the enforceable Ponder policies of the application scenario taking as input the result of the translation process described above. The Policy Encoder maps the subject and target attributes to the managed entities according to the Object Distribution of the application scenario. It also includes the events that the policy system reacts to, and the actions that the policy system is capable of enforcing. The Figure 59 shows the two policies that have been encoded to commit with m4 and m5 of the DRsM Sub-function described in previous sections. It is worth mentioning that these policies have been retrieved from the policy repository after the completion of the scenario, making use of the Ponder GUI attached to the Policy Encoder component.

The first policy, namely "/ManagedObjects/Links/DecUpprThRel", is the lowest level policy refined to enforce the transition "decrUpprThRel" of our scenario. In the TEQUILA approach, this policy is triggered when a threshold is crossed downwards, namely when the event "thrCrossingAlarm(down , utilValue , Link , OA)" occurs in the Monitoring components of the internal routers "subject =/ManagedObjects/Links/Router/Monitor". Under these circumstances, the Policy Manager Agents of the DRsM components, namely the target "/ManagedObjects/Links/Router/DRsMPMA" modifies the threshold settings reducing its value by a relative value equivalent to the 10% of the dynamic range provided by the Network Dimensioning (ND) component. This is achieved with the action "decrUpprThs (Link , OA , OA . bwthr = OA . bwthr - 0.1 * Link . dynamicRanges)". Similar encodings were carried out for the rest of the policies in the application scenario for which the next section provides a detailed description.

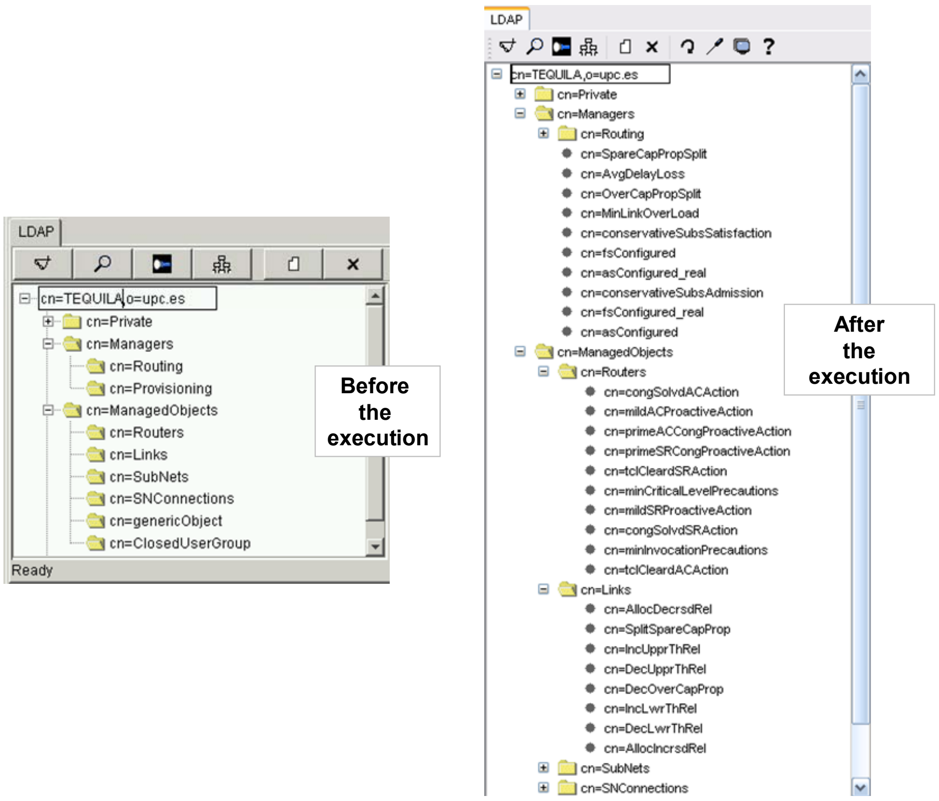

Ponder - Untitled
File Edit Build Tools Policies Options Window Help
Source Code Meta Policies

```
inst oblig /ManagedObjects/Links/DecUpprThRel {
    on thrCrossingAlarm ( down , utilValue , Link , OA ) ;
    subject s = /ManagedObjects/Links/Router/Monitor ;
    target t = /ManagedObjects/Links/Router/DRsMPMA ;
    do decrUpprThs ( Link , OA , OA . bwthr = OA . bwthr - 0.1 * Link . dynamicRanges ) ;
}
inst oblig /ManagedObjects/Links/AllocIncrsdRel {
    on thrUpAlarmProcessing ( Link , lwr , uppr , OA ) ;
    subject s = /ManagedObjects/Links/Router/DMain ;
    target t = /ManagedObjects/Links/Router/DRsMPMA ;
    do incrAlloc ( Link , OA , OA . bw = OA . bw + 0.1 * Link . dynamicRanges ) ;
}
```

Figure 59. Selection of refined policies of our scenario

5.7 Results

Having gone through the refinement process for our application scenario, driven by the principles of our refinement framework, a total of 28 Ponder Obligation policies have resulted from its execution. These policies have been obtained and stored in the policy repository attached to the refinement framework. The population of the repository before and after the execution of this application scenario is shown in Figure 60 for which a general description is provided thereafter. In order to better describe these policies, we have sectioned this description taking as reference the high-level goals for QoS Management defined for the TEQUILA approach.



```
LDAP
cn=TEQUILA,o=upc.es
  cn=Private
  cn=Managers
    cn=Routing
    cn=Provisioning
  cn=ManagedObjects
    cn=Routers
    cn=Links
    cn=SubNets
    cn=SNConnections
    cn=genericObject
    cn=ClosedUserGroup
Ready
```

```
LDAP
cn=TEQUILA,o=upc.es
  cn=Private
  cn=Managers
    cn=Routing
    cn=SpareCapPropSplit
    cn=AvgDelayLoss
    cn=OverCapPropSplit
    cn=MinLinkOverLoad
    cn=conservativeSubsSatisfaction
    cn=fsConfigured
    cn=asConfigured_real
    cn=conservativeSubsAdmission
    cn=fsConfigured_real
    cn=asConfigured
  cn=ManagedObjects
    cn=Routers
      cn=congSolvdACAction
      cn=mildACProactiveAction
      cn=primeACCongProactiveAction
      cn=primeSRCongProactiveAction
      cn=tclCleardSRAction
      cn=minCriticalLevelPrecautions
      cn=mildSRProactiveAction
      cn=congSolvdSRAction
      cn=minInvocationPrecautions
      cn=tclCleardACAction
    cn=Links
      cn=AllocDecrsdRel
      cn=SplitSpareCapProp
      cn=IncUpprThRel
      cn=DecUpprThRel
      cn=DecOverCapProp
      cn=IncLwrThRel
      cn=DecLwrThRel
      cn=AllocIncrsdRel
    cn=SubNets
    cn=SNConnections
```

Figure 60. Policy repository before and after the execution of the scenario

5.7.1 Number of Subscriptions Controlled

The policies involved in realising this goal are "conservativeSubsSatisfaction" and "conservativeSubsAdmission". The Ponder representations of these policies are shown in Figure 61.

```
inst oblig /Managers/conservativeSubsSatisfaction {
                on SRFactrsSet ( TT , serviceType ) ;
                subject s = /Managers/SSM/SLSServSatisfMO ;
                target t = /Managers/SSM/SLSSPMA ;
                do setSatisfLevl ( serviceType . PHB , serviceType . SL = 1 ) ;
}
inst oblig /Managers/conservativeSubsAdmission {
                on ramRecvd ( TT , SRamin , SRwmin , SRmax , serviceType ) ;
                subject s = /Managers/SSM/BufferMO ;
                target t = /Managers/SSM/SLSSPMA ;
                do setMaxAccpt ( serviceType . PHB , serviceType . TDtmax < serviceType . Rwmin ) ;
}
```

Figure 61. Policies involved in the goal Number of Subscriptions Controlled

The policy "conservativeSubsSatisfaction" is triggered when there is a need to set the satisfaction level for potential subscriptions in the service subscription sub-system of TEQUILA. This occurs when the Satisfaction Rate factors have been set (event SRFactrsSet) in the Service Subscription Manager (subject /Managers/SSM/SLSServSatisfMO). The policy itself is used to define appropriate values on which the service admission mechanisms would rely; namely, the Satisfaction Level. The action enforced by this policy, and basically the value assigned to the Satisfaction Level (serviceType.SL = 1) ensures that the active SLSs under TEQUILA would enjoy their QoS at their almost satisfied rates even at congestion. Other policies described in Section 5.7.4 are used to define almost/fully satisfied rates. The subject and target objects of this policy belong to the service subscription manager components of the TEQUILA approach (/Managers/SSM/).

The policy "conservativeSubsAdmission" is triggered when the Resource Availability Matrix has been received from the Traffic Engineering function of the TEQUILA approach, namely when the event "ramRecvd(TT, SRamin, SRwmin, SRmax , serviceType)" occurs in the buffer control of the SSM (subject /Managers/SSM/BufferMO). The policy is used to specify an acceptable area in the calculated available resources per-TT. The action enforced by this policy defines a threshold to decide whether to accept an incoming request or not. More concretely, this policy ensures that the maximum estimated demand of the active subscriptions is lower than the resources previously calculated to guarantee the almost satisfied rates of the previously subscribed services, which is practically defined by the policy action "setMaxAccpt (serviceType.PHB, serviceType.TDtmax < serviceType.Rwmin)" of this policy.

5.7.2 Traffic Injection Controlled

The policies involved in realising this goal are "minCriticalLevelPrecautions" and "minInvocationPrecautions". The Ponder representations of these policies are shown in Figure 62.

```
inst oblig /ManagedObjects/Routers/minCriticalLevelPrecautions {
                on newConfigRecvd ( TT ) ;
                subject s = /Managers/Routers/SLSIMonitor ;
                target t = /Managers/Routers/SLSIPMA ;
                do setTCL ( TT , TT . tcl = s . Rmax ) ;}

inst oblig /ManagedObjects/Routers/minInvocationPrecautions {
                on newConfigRecvd ( TT ) ;
                subject s = /Managers/Routers/SLSIAdmission ;
                target t = /Managers/Routers/SLSIPMA ;
                do setAC ( TT , s . ACmin = 0 , s . ACmax = s . Rmax ) ;
}
```

Figure 62. Policies involved in the goal Traffic Injection Controlled

The policy "minCriticalLevelPrecautions" is triggered when there is a need to set the Target Critical Level (TCL) in the service invocation sub-systems of TEQUILA, namely when new Traffic Trunk (TT) configurations are received (event newConfigRecvd(TT)) in the Service Invocation Monitor (s=/Managers/Routers/SLSIMonitor) components. The action enforced by this policy is used to specify the level at which the likelihood of overwhelming the network is considered critical. In this particular case, the policy sets the higher value that can be assigned to the TCL by executing the policy action "setTCL(TT, TT.tcl = s.Rmax)". In this case, the precautions taken will be the lowest as the target critical level is equal to the maximum available resources. The subject and target objects of this policy belong to the service invocation components of the TEQUILA approach. This policy has some impact on the enforcement of the policies described in Section 5.7.3.

The policy “minInvocationPrecautions” is triggered when new Traffic Trunk (TT) configurations are received (event newConfigRecvd(TT)) in the Service Admission controls of the SLS-I modules (s=/Managers/Routers/SLSIAdmission). The policy is used to position thresholds between R^{w}_{min} and R_{max} in the Resource Allocation Buffer (RAB) for invocation purposes. While the R^{w}_{min} are the resources always available for the TT even at congestion, the R_{max} are the maximum resources available for the corresponding TT. For our application scenario, the threshold is set to the maximum available resources calculated per Traffic Trunk, given by the policy action “setAC(TT, s.ACmin=0, s.ACmax=s.Rmax)”.

5.7.3 QoS Degradation Prevented

This goal is realised by the policies “mildSRProactiveAction”, “mildACProactiveAction”, “tclCleardSRAction”, “tclCleardACAction”, “primeSRCongProactiveAction”, “primeACCongProactiveAction”, “congSolvdSRAction”, and “congSolvdACAction”. The Ponder representations of these policies are shown in Figure 63.

```
inst oblig /ManagedObjects/Routers/mildSRProactiveAction{
on tclAlarmRaised ( up , TT ) ;
subject s = /Managers/Routers/SLSIMain ;
target t = /Managers/Routers/SLSIPMA ;
do setSR ( TT , s . SR = s . SRas + ( s . SRfs - s . SRas ) / 2 ) ; }

inst oblig /ManagedObjects/Routers/mildACProactiveAction {
on tclAlarmRaised ( up , TT ) -> setSR ( ) ;
subject s = /Managers/Routers/SLSIMain ;
target t = /Managers/Routers/SLSIPMA ;
do setAC ( TT , s . ACmax = s . Rwmin ) ; }

inst oblig /ManagedObjects/Routers/tclCleardSRAction {
on tclAlarmRaised ( down , TT ) ;
subject s = /Managers/Routers/SLSIMain ;
target t = /Managers/Routers/SLSIPMA ;
do setSR ( TT , s . SR = s . SRfs  ) ;}

inst oblig /ManagedObjects/Routers/tclCleardACAction {
on tclAlarmRaised ( down , TT ) -> setSR ( ) ;
subject s = /Managers/Routers/SLSIMain ;
target t = /Managers/Routers/SLSIPMA ;
do setAC ( TT , s . ACmax = s . Rmax ) ;}
```

```
inst oblig /ManagedObjects/Routers/primeSRCongProactiveAction{
on congAlarmRaised ( TT ) ;
subject s = /Managers/Routers/SLSIMain ;
target t = /Managers/Routers/SLSIPMA ;
do setSR ( TT , s . SR = s . SRas ) ; }

inst oblig /ManagedObjects/Routers/primeACCongProactiveAction{
on congAlarmRaised ( TT ) -> setSR ( ) ;
subject s = /Managers/Routers/SLSIMain ;
target t = /Managers/Routers/SLSIPMA ;
do setAC ( TT , s . ACmax = 0 ) ; }

inst oblig /ManagedObjects/Routers/congSolvdSRAction {
on congResolvdAlarmRaised(TT) ;
subject s = /Managers/Routers/SLSIMain ;
target t = /Managers/Routers/SLSIPMA ;
do setSR ( TT , s . SR = s . SRfs ) ; }

inst oblig /ManagedObjects/Routers/congSolvdACAction {
on congResolvdAlarmRaised(TT) -> setSR ( ) ;
subject s = /Managers/Routers/SLSIMain ;
target t = /Managers/Routers/SLSIPMA ;
do setAC ( TT , s . ACmax = s . Rmax ) ; }
```

Figure 63. Policies involved in the goal QoS Degradation Prevented

The “mildSRProactiveAction” policy is triggered when the target critical level is crossed in the SLSI components, namely when the event “tclAlarmRaised(up, TT)” occurs in the SLSI Main component (s = /Managers/Routers/SLSIMain). The action enforced by this policy, namely “setSR (TT, s.SR=s.SRas + (s.SRfs - s.SRas) / 2)”, is used to reduce the service rates half-way between the calculated resources meant to provide fully satisfied rates (SRfs), and almost satisfied rates (SRas) to active services in the Resource Allocation Buffer (RAB) of the respective traffic trunks.

The “mildACProactiveAction” policy is triggered when the target critical level is crossed and new service rates have been assigned in the SLSI components, namely when the event “tclAlarmRaised (up, TT) -> setSR ( )” is registered in the SLSI Main components. This policy is used to set up a threshold for new invocations up to minimum resources always available, even at congestion state (R^{w}_{min}) in the Resource Allocation Buffer (RAB).

The “tclCleardSRAction” and “tclCleardACAction” policies assess the restoration of service rate and admission control adjustments once the target critical level alarm has been cleared. On the one hand, the former policy is committed to adjust the service rate to the fully satisfied rates for each TT by executing the action “setSR(TT, s.SR = s.SRfs)”, once the target critical level alarms have been cleared up in the corresponding traffic trunk in the SLSI components (event tclAlarmRaised(down, TT)). On the other hand, the “tclCleardACAction” policy adjusts the admission control mechanisms again, to the maximum available resources for each traffic trunk by executing the policy action “setAC (TT, s.ACmax = s.Rmax)”, once the target critical level alarm has been cleared up and the service rate adjustments have taken place in the SLSI components (event tclAlarmRaised(down, TT) -> setSR ()).

The “primeSRCongProactiveAction” and “primeACCongProactiveAction” policies drive the congestion resolution mechanisms in the traffic trunks of the network. The former policy commits with this task by limiting active users to enjoy rates aligned to their almost satisfied rates. This is achieved by enforcing the policy action setSR(TT, s.SR = s.SRas), under TT congestion events “congAlarmRaised(TT)” occurring in the SLSI components. In addition, admission control actions are enforced by the policy “primeACCongProactiveAction” in the sense of rejecting new invocations. This is achieved by setting up of the control thresholds to zero with the policy action “setAC(TT, s.ACmax = 0)”. This policy is triggered when the congestion alarm has been issued and the SLSI has taken service rate adjustments (event congAlarmRaised (TT) -> setSR ( )).

Finally, the “congSolvdSRAction” and “congSolvdACAction” policies drive the restoration of service rate and admission control mechanisms when the congestion states have been resolved in the SLSI components. On the one hand, the former policy is committed to restore the rates-to-enjoy by active users, to those aligned to their fully satisfied rates by enforcing the policy action “setSR (TT, s.SR = s.SRfs)”. On the other hand, the policy “congSolvdACAction” restores the admission control mechanisms to allow new invocations up to the maximum available resources in corresponding traffic trunk, by executing the policy action “setAC(TT, s.ACmax = s.Rmax)”.

5.7.4 Traffic Demand Estimated

The policies involved in realising this goal are “asConfigured”, “asConfigured_real”, “fsConfigured”, and “fsConfigured_real” whose Ponder representations are shown in Figure 64.

```
inst oblig /Managers/asConfigured {
on newRPC ( TT , serviceType , TD ) ;
subject s = /Managers/SSM/TF ;
target t = /Managers/SSM/TFPMA ;
do setAlmstSatisfFactr ( serviceType . PHB , serviceType .
FactrSRAS = 60 ) ; }

inst oblig /Managers/asConfigured_real {
on newRPC ( TT , serviceType , TD ) ;
subject s = /Managers/SSM/TF ;
target t = /Managers/SSM/TFPMA ;
do setAlmstSatisfFactr ( serviceType . PHB = "real" , serviceType .
FactrSRAS = 100 ) ; }
```

```
inst oblig /Managers/fsConfigured {
on newRPC ( TT , serviceType , TD ) -> setAlmstSatisfFactr ( ) ;
subject s = /Managers/SSM/TF ;
target t = /Managers/SSM/TFPMA ;
do setFulSatisfFactr ( serviceType . PHB , serviceType .
FactrSRFS = 85 ) ; }

inst oblig /Managers/fsConfigured_real {
on newRPC ( TT , serviceType , TD ) -> setAlmstSatisfFactr ( ) ;
subject s = /Managers/SSM/TF ;
target t = /Managers/SSM/TFPMA ;
do setFulSatisfFactr ( serviceType . PHB = "real" , serviceType .
FactrSRFS = 100 ) ; }
```

Figure 64. Policies involved in the goal Traffic Demand Estimated

The policies "asConfigured" and "asConfigured_real" are triggered when there is a new Resource for Provisioning Cycle (RPC) in the Traffic Forecast (TF) component of TEQUILA. This event is represented as "newRPC(TT, serviceType, TD)" in Fig. 64. These two policies are used to drive the percentage of the traffic rate, which if offered to an SLS, the SLS is thought to be "almost satisfied". The "asConfigured_real" policy takes care of real-time services for which the percentage rate is set to 100%. This is achieved by enforcing the policy action "setAlmstSatisfFactr (serviceType.PHB = "real", serviceType.FactrSRAS = 100)" in the TF module of TEQUILA. The "asConfigured" policy takes care of the rest of the services. For these the "asConfigured" policy sets the percentage rate to 60% by enforcing the action "setAlmstSatisfFactr (serviceType.PHB, serviceType.FactrSRAS = 60)".

The policies "fsConfigured" and "fsConfigured_real" are triggered when there is a new Resource for Provisioning Cycle (RPC), followed by a setting up of the almost satisfied percentages described above. The values enforced with the "fsConfigured_real" policy are 100% for real-time services, enforced with the action "setFulSatisfFactr (serviceType.PHB = "real", serviceType.FactrSRFS = 100)". On the other hand, the percentage rate enforced by the "fsConfigured" for the rest of the services is 85%. This situation is achieved by enforcing the action "setFulSatisfFactr (serviceType.PHB, serviceType.FactrSRFS = 85)".

5.7.5 Available Resources per-Traffic Trunk Calculated

The policies involved in realising this goal are "AvgDelayLoss", "MinLinkOverLoad", "SpareCapPropSplit" and "OverCapPropSplit". The Ponder representations of these policies are shown in Figure 65.

```
inst oblig /Managers/AvgDelayLoss {
on doRPC ( OA , bw , links ) ;
subject s = /Managers/ND/NDPMA ;;
target t = /Managers/ND/HopCount;
do calculateHopCount ( OA , links . hopConstraint = "avg" ) ;}
```

```
inst oblig /Managers/MinLinkOverLoad {
on startingOptimisation ( TEM , TTs , Links ) ;
subject s = /Managers/ND/NDPMA ;
target t = /Managers/ND/Optimisation ;
do setCostFunctionE ( s . OA , s . exp = "max" ) ;
}
```

```
inst oblig /Managers/SpareCapPropSplit {
on runPostProcessing ( TTs , TEM , OA ) ;
subject s = /Managers/ND/NDPMA ;
target t = /Managers/ND/PostProcessing ;
do allocSpareBW ( OA . bw = "prop" ) ;}
```

```
inst oblig /Managers/OverCapPropSplit {
on redistributeSpare ( TTs , TEM , OA ) ;
subject s = /Managers/ND/NDPMA ;
target t = /Managers/ND/PostProcessing ;
do reduceOverBW ( OA . bw = "prop" ) ;
}
```

Figure 65. Policies involved in the goal Available Resources per-TT Calculated

The above policies modify the behaviour of the internal components of the Network Dimensioning module for resources allocation tasks. The "AvgDelayLoss" policy is triggered when a new Resource Provisioning Cycle (RPC) occurs in the Network Dimensioning module (ND) of the TEQUILA system. This policy sets the strategy to follow for the expected traffic calculations in terms of delay and loss requirements. In other words, this policy enforces that the HopCount manager of the ND, to consider the average delay and packet loss introduced by the links of the underlying network as the default for the maximum hop-count constraints. This is achieved by enforcing the policy action calculateHopCount (OA, links.hopConstraint = "avg") as shown in Figure 65.

The policy "MinLinkOverLoad" is triggered when the network calculation process starts, namely when the event "startingOptimisation (TEM, TTs, Links)" is registered in the ND module of the TEQUILA system. This policy enforces that the network utilization shares the load throughout the links of the underlying network, hence avoiding that some links end-up overloaded after the calculation of resources. This is achieved by enforcing the policy action "setCostFunctionE (s.OA, s.exp = "max")" in the Optimisation module of the ND system (t = /Managers/ND/Optimisation).

The policy "SpareCapPropSplit" is triggered when the ND is forced to define how to handle the remaining resources of the network, once the calculation of the resources has been carried out. This situation occurs when the event "runPostProcessing (TTs, TEM, OA)" is registered in the ND module. This policy enforces to redistribute the usage of additional resources proportionally amongst the OAs sharing the corresponding traffic trunk. This is achieved when the action "allocSpareBW (OA.bw = "prop")" is executed in the PostProcessing module of the ND system (t = /Managers/ND/PostProcessing).

Finally, the policy "OverCapPropSplit" is triggered when the ND defines the way it will handle resources reductions in case of over-provisioning BW. This situation occurs when the event "redistributeSpare (TTs, TEM, OA)" is registered in the ND module. This policy enforces that in all cases, the reductions will be "proportional" to the resources reserved for each OA in the TEQUILA system. This is, by enforcing the policy action "reduce-OverBW (OA. bw="prop")" in the ND PostProcessing module (t = /Managers/ND/PostProcessing).

5.7.6 Dynamic Fluctuations Managed

This goal is realised by the policies "IncLwrThRel", "DecLwrThRel", "IncUpprThRel", "DecUpprThRel", "AllocIncrsdRel", "AllocDecrsdRel", "SplitSpareCapProp", and "DecOverCapProp". The Ponder representations of these policies are shown in Figure 66 for which a brief description is provided thereafter.

```
inst oblig /ManagedObjects/Links/IncLwrThRel {
on thrCrossingAlarm ( up , utilValue , Link , OA ) ;
subject s = /ManagedObjects/Links/Router/Monitor ;
target t = /ManagedObjects/Links/Router/DRsMPMA ;
do incrLwrThs ( Link , OA , OA . bwthr = OA . bwthr + 0.1 ∗ Link
. dynamicRanges ) ;
}
inst oblig /ManagedObjects/Links/DecLwrThRel {
on thrCrossingAlarm ( down , utilValue , Link , OA ) ;
subject s = /ManagedObjects/Links/Router/Monitor ;
target t = /ManagedObjects/Links/Router/DRsMPMA ;
do decrLwrThs ( Link , OA , OA . bwthr = OA . bwthr − 0.1 ∗ Link
. dynamicRanges ) ;
}
inst oblig /ManagedObjects/Links/IncUpprThRel {
on thrCrossingAlarm ( up , utilValue , Link , OA ) ;
subject s = /ManagedObjects/Links/Router/Monitor ;
target t = /ManagedObjects/Links/Router/DRsMPMA ;
do incrUpprThs ( Link , OA , OA . bwthr = OA . bwthr + 0.1 ∗
Link . dynamicRanges ) ;
}
inst oblig /ManagedObjects/Links/DecUpprThRel {
on thrCrossingAlarm ( down , utilValue , Link , OA ) ;
subject s = /ManagedObjects/Links/Router/Monitor ;
target t = /ManagedObjects/Links/Router/DRsMPMA ;
do decrUpprThs ( Link , OA , OA . bwthr = OA . bwthr − 0.1 ∗
Link . dynamicRanges ) ;
}
inst oblig /ManagedObjects/Links/AllocIncrsdRel {
on thrUpAlarmProcessing (utilValue , Link , OA ) ;
subject s = /ManagedObjects/Links/Router/DMain ;
target t = /ManagedObjects/Links/Router/DRsMPMA ;
do incrAlloc ( Link , OA , OA . bw = OA . bw + 0.1 ∗ Link .
dynamicRanges ) ;
}
inst oblig /ManagedObjects/Links/AllocDecrsdRel {
on thrDwnAlarmProcessing (utilValue , Link , OA ) ;
subject s = /ManagedObjects/Links/Router/DMain ;
target t = /ManagedObjects/Links/Router/DRsMPMA ;
do decrAlloc ( Link , OA , OA . bw = OA . bw − 0.1 ∗ Link .
dynamicRanges ) ;
}
inst oblig /ManagedObjects/Links/SplitSpareCapProp {
on postProcessingRequest ( Link , OA , BW ) ;
subject s = /ManagedObjects/Links/Router/DMain ;
target t = /ManagedObjects/Links/Router/DRsMPMA ;
do splitSpareCap ( Link , OA , s . LinkBWalloc = "proportionally" ) ;
}
inst oblig /ManagedObjects/Links/DecOverCapProp {
on postProcessingRequest ( Link , OA , BW ) ;
subject s = /ManagedObjects/Links/Router/DMain ;
target t = /ManagedObjects/Links/Router/DRsMPMA ;
do decrOverBW ( Link , OA , s . LinkBWred = "proportionally" ) ;
}
```

Figure 66. Policies involved in the goal Dynamic Fluctuations Managed

The policies shown on the left part of the Figure 66 handle the management of threshold crossings in the Monitoring components of the DRsM modules of the TEQUILA system. DRsM modules are attached to every link interface of the core routers pertaining to the underlying network, which are in turn defined as Monitor in the subject field of these policies. These policies are triggered when threshold crossing alarm (thrCrossingAlarm) events are registered in the Monitor components (s= /ManagedObjects/Links/Router/Monitor). The enforcement of the policies "IncLwrThRel", and "IncUpprThRel" results in threshold adjustments covering, respectively, lower and upper threshold settings for upwards threshold crossings.

Similarly, downwards threshold crossings are handled by the policies "DecLwrThRel", and "DecUpprThRel", both for lower and upper threshold crossings respectively. In all these cases, the threshold adjustments are 10 percent the dynamic range provided by the ND module, this is 10% of the difference between the maximum and minimum values that the DRsM can drive for dynamic traffic fluctuations. A concrete example is the action enforced by the policy "DecLwrThRel", namely "decrLwrThs (Link, OA, OA. bwthr = OA.bwthr - 0.1 * Link.dynamicRanges)" which decreases 10% of the dynamic range, the lower threshold in the DRsM module. Similar actions are enforced for the other policies shown in the left part of Fig. 66 to cope with the corresponding threshold crossings.

The policies "AllocIncrsdRel" and "AllocDecrsdRel" shown in the upper-right part of Figure 66 handle the increment of resources allocation. These policies are triggered by the upper and lower threshold crossing handlings respectively. These triggering events, namely "thrUpAlarmProcessing (utilValue, Link, OA)", and "thrDwnAlarmProcessing (utilValue, Link, OA)" are registered in the DMain components of the DRsM modules of the TEQUILA system. The policy actions in these two policies are enforced to increase/decrease the allocated resources for the corresponding OA for which the threshold crossing was issued. For example, the policy "AllocDecrsdRel" enforces the action "decrAlloc (Link, OA, OA.bw = OA. bw - 0.1 * Link.dynamicRanges)" to reduce the allocated bandwidth by a relative value equivalent to the 10% of the dynamic range provided by the ND component.

The policy "SplitSpareCapProp" is triggered by events signalling resources re-allocations in the DRsM, namely by means of post processing events occurring in the Main components (see postProcessingRequest(Link, OA, BW) event in Fig. 66). This policy enforces the action "splitSpareCap (Link, OA, s.LinkBWRed = "proportionally")" which ensures that the spare resources of the link are used proportionally amongst all the OAs sharing the link to which the DRsM module is attached.

Finally, the policy "DecOverCapProp" is also triggered after resource re-allocations. Nevertheless, it enforces the policy action "decrOverBW(Link, OA, s.LinkBWRed = "proportionally")" to make sure over-provisioned resources are decreased proportionally amongst the OAs sharing the link to which the corresponding DRsM belong to.

5.8 Analysis of the managed system behaviour

In this Section we provide information regarding the policy-guided behaviour of the TEQUILA approach taking as reference the refined policies of our application scenario. It is not our intention to validate the TEQUILA approach or the enforcement of the refined policies, as this may lead to statistical results that lay out of the scope of this Thesis. Our objective is to point out the constrained behaviour obtained through the execution of available refined policies. The results acquired through this validation scenario have been verified with the ones obtained as part of the final TEQUILA dissemination results [Damil02]. The interested reader may use this reference for a detailed description of the validation results produced for the execution of scenarios involving quantitative data regarding service subscriptions, traffic injection, and network topologies.

5.8.1 System behaviour based on traffic input

The policies refined in our application scenario enable the functional operation of the TEQUILA system as a result of the holistic view of policy refinement for QoS Management [Rub06c]. The following is the behaviour of the traffic input points that is controlled by policies refined in this application scenario. To better describe this aspect, we make use of the summarised representation of system behaviour shown in Figure 67.

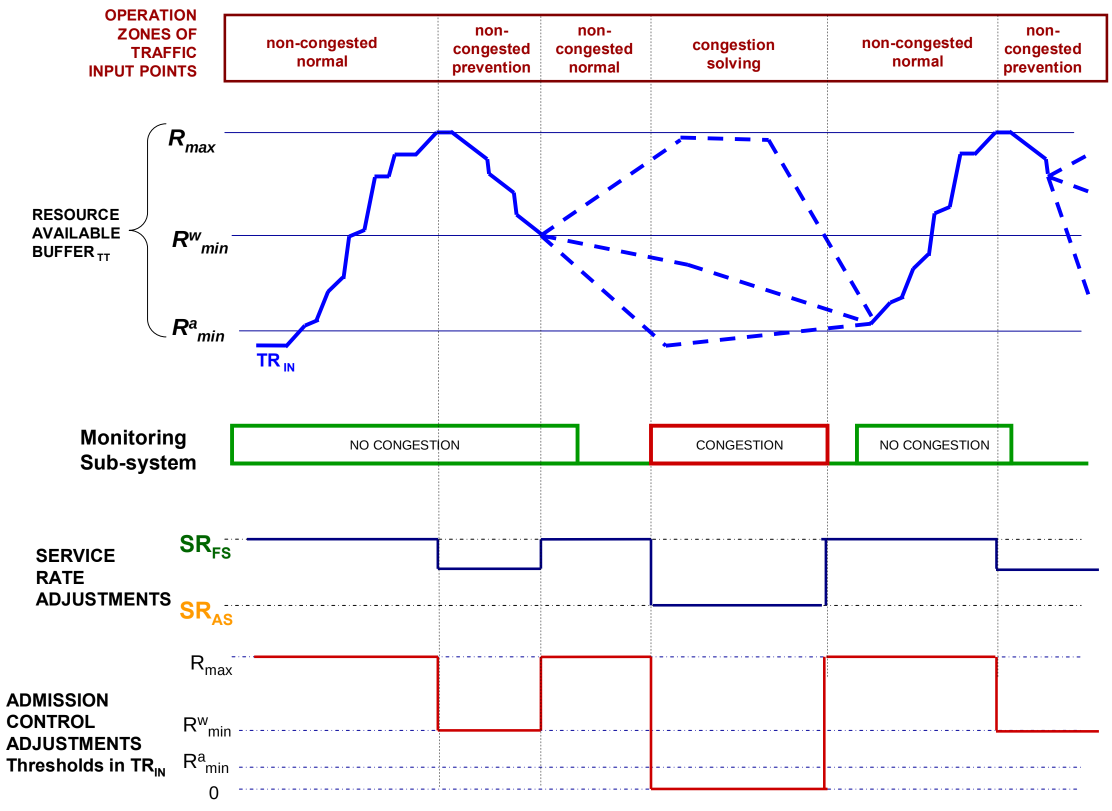


Figure 67. Summarised representation of system behaviour due to traffic input

As we have described earlier, for every traffic input point, the TEQUILA system calculates three key values that define the Resource Allocation Buffer per-TT (RAB_{TT}); maximum resources (*Rmax*), minimum always available even at congestion ($R^w min$) and minimum always available ($R^a min$). The traffic input (TR_{IN}) in the system is referred to the RAB_{TT} and its evolution is depicted in Fig. 67.

The refined policies constrain the relevant TEQUILA components and subsequently define three operational zones for the traffic input points. For better explanation these are referred to as “non-congested normal”, “non-congested prevention”, and “congestion solving”. These zones are graphically presented in the upper-part of Figure 67.

The system works in “non-congested normal” mode when the traffic injected in the network ranges from zero to the maximum resources for a specific TT and no congestion alarms are reported from the monitoring sub-system. In this mode, the service rate adjustments allow active users to send traffic up to their fully satisfied rates (SR_{FS}) and the admission control threshold is set to Rmax.

As depicted in Figure 67, the system works in “non-congested prevention” mode when the traffic injected in the network has reached the maximum resources of a TT (Rmax), and no congestion alarms are reported from the monitoring sub-system. In this case the service rate adjustments reduce the traffic injection of active users, half-way between fully satisfied rates (SR_{FS}) and almost satisfied rates (SR_{AS}). Also, in this mode the admission control adjustments reduce the threshold up to R^wmin. Eventually, the traffic input is reduced until its value crosses downwards R^wmin in the RAB_{TT}. At this point, the system is considered to shift to “non-congested normal” mode.

Similarly, the system works in “congestion solving” mode when, irrespective of the traffic input, the monitoring subsystem reports congestion in the system. By irrespective, we mean that the traffic input (TR_{IN}) in the system can be located in any point of the RAB_{TT} as we represent with the three dotted lines of TR_{IN} in the central part of Figure 67. More concretely, note that in the non-congested area, TR_{IN} can be in any part of the RAB_{TT}, same as within the congestion-solving area, see the evolution of the three dotted lines in Figure 67. This means that the “congestion solving” area can occur irrespective of the traffic input. In the “congestion solving” operational mode the admission control rejects any potential invocation (threshold set to 0) and the service rate of active users is reduced up to almost satisfied rates (SR_{AS}). The traffic input is reduced as a consequence of the former adjustments. Eventually, the congestion alarms are cleared thus shifting the traffic input point to “non-congested normal” mode.

5.8.2 System behaviour based on inner observed load

This section illustrates the behaviour of the system with respect to the inner observed load, this is, we show the internal rearrangements of resources to the different Per-Hop-Behaviours (PHBs) of the network as a result of the policies refined in this application scenario. To better describe this aspect, we make use of the summarised representation of system behaviour shown in Figure 68.

As we have described earlier, the TEQUILA system calculates, on a Per-Hop-Behaviour (PHB) basis, three key values; *maximum*, *minimum* and *congestion* required resources at each network interface represented as PHB_{max}, PHB_{min} and PHB_{cong} in Figure 68. In the text below, we consider a network interface shared by three PHBs.

The system is forced to rearrange resources based on observed load fluctuations based on policy rules. For instance, should PHB1 be under-utilised as we show in Figure 68, the system reduces its BW allocation so that spare BW can be allocated to other PHBs, for example PHB2 and PHB3 (see PHB1 under-utilised circle in Figure 68). Should PHB1 be over-utilised as we show in Figure 68, the system increases its BW allocation if sufficient link capacity is available, for example, from PHB2 and PHB3 in Figure 68 (see PHB1 over- utilised circle in Figure 68).

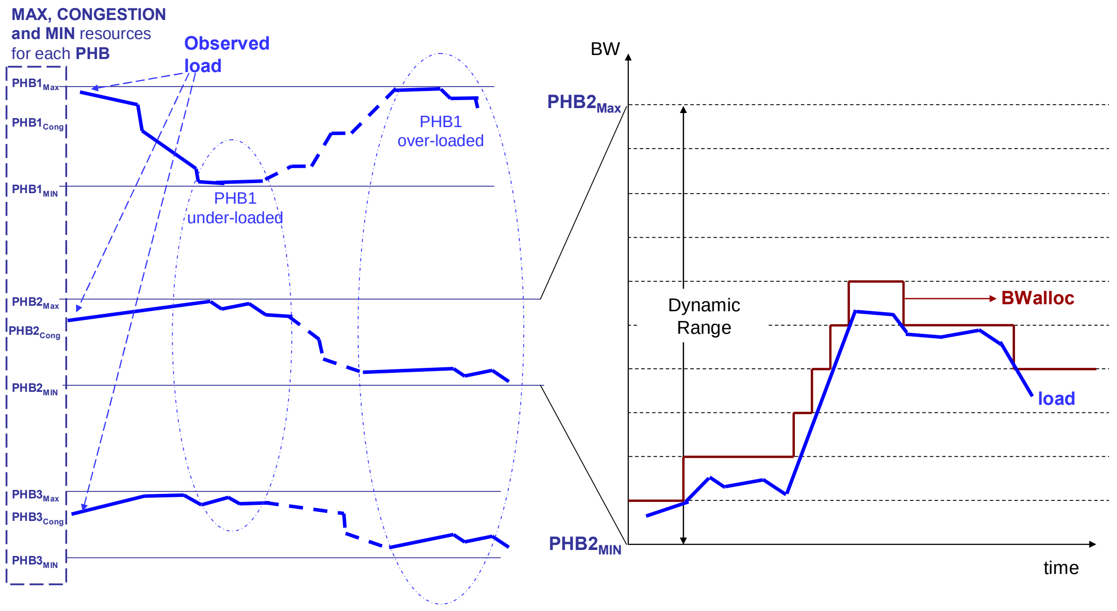


Figure 68. Summarised system behaviour due to inner observed load

The network interfaces react to observed load fluctuations within the maximum and minimum resources estimated for each PHB. The later is identified as dynamic range in Figure 68. The reactions are carried out in a discrete manner, this is, they take place when the load reaches specific points within the dynamic range. In this case, the system detects load fluctuations in ten symmetrical points within the dynamic range. Similarly, the system is forced to vary BW allocations in steps given by ten symmetrical points within the dynamic range for the corresponding PHB as we graphically show on the right part of Figure 68.

5.9 Conclusions

This Chapter has presented the applicability of a holistic refinement approach in the quality of service (QoS) management domain. Namely, we have presented how the policy refinement framework and the methodological approach to goal refinement described in Chapter 3 and Chapter 4 respectively, can be used to refine enforceable policies from QoS-oriented administrative views in a systematic manner. We have presented a holistic and realistic application scenario of policy refinement, capturing the requirements, processes, actors and phases involved in such a critical process. As far as we know, no other work has explicitly addressed the policy refinement problem in any application domain in such a complete view.

The contribution of this Chapter is itself the assessment of a policy refinement process in a holistic view, for a network management domain [Rub06c], namely for the QoS Management domain. This achievement has not represented significant administrative efforts as we have handled information that has been used during the design, implementation, and the operation of the TEQUILA approach. The following are some conclusions drawn after executing the refinement process in this concrete application domain.

We can conclude that the high-level goals definition is one of the most important steps for the assessment of the QoS-oriented refinement process. This step, which we suggest to be carried out during the design of the system, pre-establishes the way the administrator consultant will define operative views of QoS Management at runtime as these are generated through the high-level goals. In this sense, a remarkable issue of the methodological approach is that all the refinement process relies on administrative decisions. Nevertheless, policy authoring environments and policy-based management itself are driven by administrative decisions. Hence, we can conclude that the methodological approach does not substitute the administrative parties to be the most important sources for decisions of how to control QoS provisioning. Moreover, we have provided the administrative parties with an affordable approach to control QoS provisioning. We acknowledge that additional analysis techniques should be integrated to reduce or to avoid if possible, potential mistakes that the administrative parties could make during the refinement process.

Regarding the system composition hierarchy, the Service Management and the Traffic Engineering functions of the TEQUILA approach do not work isolated. Hence, a remarkable issue here is that the system composition hierarchy should integrate the details of these two functionalities which in turn should be used to drive the QoS-oriented goal refinement process. On the other hand, the administrator developer should also provide a complete and correct System Model for the managed system involved in the TEQUILA approach. It is mandatory that the administrator developer should provide complete and correct specifications for the former two issues. Again, the administrative parties are responsible for the definition and for checking the completeness of the information provided for these purposes. Additional analysis techniques should be figured out to reduce or to avoid if possible, potential mistakes that the administrative developer could make at these stages.

The policies refined in our application scenario enable the functional operation of the TEQUILA system to control QoS provisioning aligned to a QoS-oriented operative view. Regarding this issue, the methodological approach worked out in this Thesis has provided the means to define operative views for QoS management and/or modify current ones in a systematic manner. Moreover the application scenario has not evaluated under which conditions of user demands this operative view should be changed to allow a better performance of the network controlled with the TEQUILA approach. This may involve carrying out dense statistical analyses, with diverse network topologies, user demands, etc, which is out of the scope of the Thesis. Contrary, our methodological approach enables the administrator consultant to define operative views for QoS management and/or modify current ones in a systematic manner. We can conclude then, that our methodological approach can contribute to the development of statistical and feedback analysis techniques in favour of QoS provisioning.

Linked to the above issue and inspired in situations where a QoS-oriented operative view is no longer valid or that it should be changed, we could relate these situations to the fact that goals should evolve. Most probably, this *goal evolution* should be influenced by statistical changes of user load, topology changes, etc. So far, we have considered and provided a framework that considers static high-level goals and consequently static QoS-oriented operative views. Moreover, it is highly desirable to consider mechanisms that enable *goal evolution*. In this sense, we can conclude that our approach could be used as a starting point to address this challenging issue that may probably enable the policy refinement process to shift into a cyclic and continuous process in which contextual information could be used to drive the refinement process.

Another conclusion drawn from the execution of the scenario presented in this Chapter is that policy refinement demands a deep knowledge of the application domain at every stage of the refinement process. In this case, a deep knowledge of the QoS Management domain has been needed to carry out the validation scenario presented in this Chapter. We could conclude then that it is impossible to think of a simplistic solution to the policy refinement problem since it is a complex issue that certainly deserves more attention. Significant efforts are still necessary toward the solution of the refinement problem, most probably targeting specific application domains to identify its peculiarities and implications. In the QoS Management domain, future work could be directed to explore the implications of inter-domain QoS Management for which the research community has envisaged necessary to solve still open issues like that of the policy refinement problem. Application-wise we are currently exploring the implications of the refinement problem applied to pricing environments [Gut07].

Chapter 6 Related Work on Policy Refinement

6.1 Introduction

Policy refinement is a sub-field of policy-based management that has been rather dismissed, most probably due to its inherent complexity. In this sense, related work explicitly committing this open problem is scarce. In this Thesis, we have dealt with the critical nature of providing a complete approach to the policy refinement process, proposing all the necessary elements involved in this critical aspect of policy-based management. The following are the elements addressed in this Thesis, which are in turn represented in Figure 69 together with the techniques/principles on which such key elements rely:

- Formalising goal-oriented high-level requirements and goal refinement
- Link between goal fulfillment and system behaviour finding
- Abstraction of enforceable policies from system behaviour findings
- Functional prototype that provides support during the goal-oriented refinement process
- Application of the methodology to a concrete management application domain

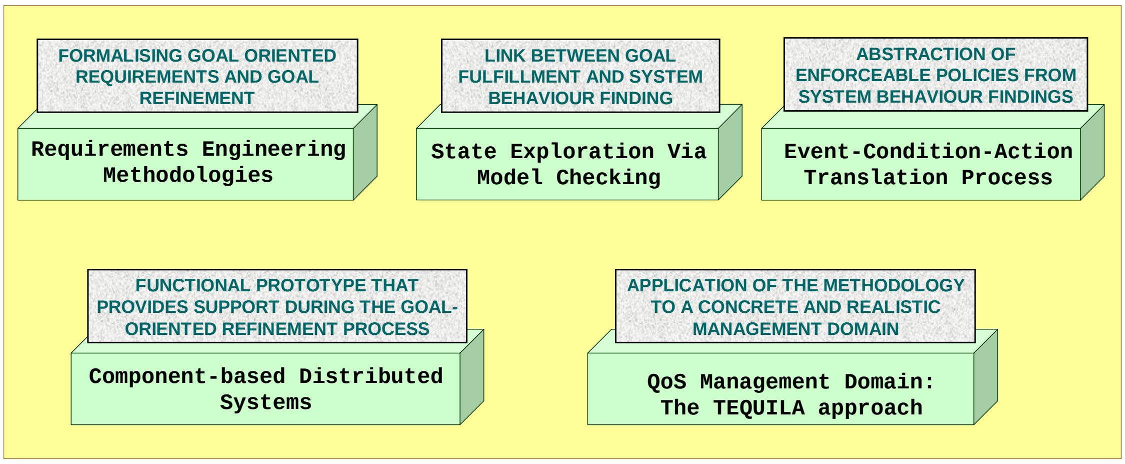

FORMALISING GOAL ORIENTED REQUIREMENTS AND GOAL REFINEMENT
Requirements Engineering Methodologies
LINK BETWEEN GOAL FULFILLMENT AND SYSTEM BEHAVIOUR FINDING
State Exploration Via Model Checking
ABSTRACTION OF ENFORCEABLE POLICIES FROM SYSTEM BEHAVIOUR FINDINGS
Event-Condition-Action Translation Process
FUNCTIONAL PROTOTYPE THAT PROVIDES SUPPORT DURING THE GOAL-ORIENTED REFINEMENT PROCESS
Component-based Distributed Systems
APPLICATION OF THE METHODOLOGY TO A CONCRETE AND REALISTIC MANAGEMENT DOMAIN
QoS Management Domain: The TEQUILA approach

Figure 69. Key elements of the Thesis

Although some relevant contributions have been provided in the refinement area, to the best of our knowledge, by the time of the publication of this Thesis, there is no evidence of any complete approach to goal-oriented policy refinement, addressing the above key elements in such a holistic manner.

This Section presents the related work on policy refinement. For this purpose we divide the related work in six sub-Sections. Section 6.2 presents related work on goal-oriented management. Section 6.3 presents an abduction and Event Calculus approach to policy refinement. Section 6.4 presents work done to generate policies from process specifications. Section 6.5 describes practical efforts carried out with refinement prototypes and finally, Section 6.6 describes other general efforts in the refinement area. We conclude this Chapter with Section 6.7.

6.2 Goal-oriented management

6.2.1 Specification of goal-oriented network management systems

The first approach addressing the need to design, implement and operate network management system by means of goals was work by Bean et. al [Bea93]. The authors proposed the first approach to turn the design of network management systems into a formal engineering discipline based on Requirements Engineering principles.

Starting from the view that Requirements Engineering is an activity of knowledge acquisition and formalization, the authors propose that such activity must not only describe the current and future system and its goals, but also its domain and range of influence. To this end, requirements engineering requires the cooperation of various members with expertise in the various aspects of network management and general system theory. Due to their very different backgrounds, these participants may have different perceptions of the goals of the future system and its environment. The challenge of the authors was to define a theoretically sound system and interface design framework, which would foster an orderly cooperation among the members participating in the requirements engineering activity and lead them to the production of a precise and complete set of end-user requirements. Team effort within the systems design framework is illustrated in Figure 70 as described in [Bea93].

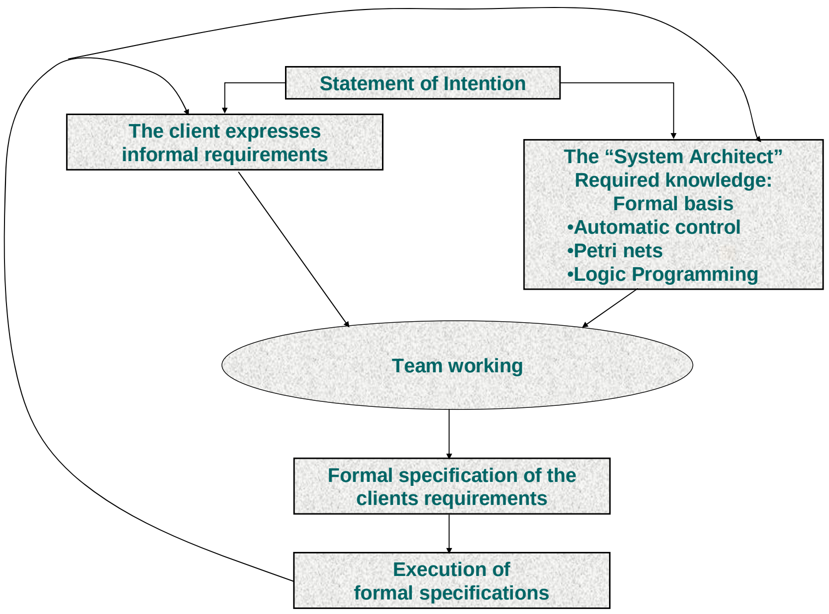

Statement of Intention
The client expresses informal requirements
The "System Architect" Required knowledge: Formal basis •Automatic control •Petri nets •Logic Programming
Team working
Formal specification of the clients requirements
Execution of formal specifications

Figure 70. System design team model

The starting point of this view is a “statement of intention,” which describes the overall goal of the system to be designed and the domain in which it must operate. There are some properties that user requirements must have. They should be executab1e. Although the end-user requirements would likely be described in textual format, the idea of the authors was to transpose these into executable code and to test them as part of the approval process. The executability of the requirements should facilitate the emergence of a common understanding among the team members, who will likely have very different backgrounds. User requirements should be verifiable. The execution of user requirements should be a simulation.

However, it was envisaged that more powerful methods should be used to ascertain that the requirements are reasonably complete and precise. In fact, the team members should be able to obtain answers to questions about the behaviour of the specified system. In general, these questions raised the concerns addressing the evaluation, the satisfiability, and the validity of the user requirements.

The authors considers that the Evaluation for example should find the state in which the system will be given a state of the system and a sequence of events. Satisfiability should deal with finding for example, which sequences of events change the system from the first state into the second, given two states of a system. At last but not least, validity should deal with situations where for example it was needed to verify that given two states of the system and a (possibly infinite) set of sequences of events, it was true that all the sequences of events in the set certainly change the system from the first state into the second.

The framework developed by Bean et al. considers that the tasks of writing and verifying user requirements should be carried out in parallel. For this, they considered that the structure of user requirements should be inspired by automatic control theory. Since real-time network management systems are control systems, the authors envisaged that philosophy and methods of automatic control theory should be used for their design.

According to this philosophy, user requirements should clearly specify two components: the controlled domain whose behaviour is being impacted by the future system (the controller), and the goal to be achieved by the controlled domain working in cooperation with the future system. This same concept applies when defining the interfaces between the cooperating systems and the information that must be defined specifically for this communication. The overall view of Bean et al., is graphically shown in Figure 71.

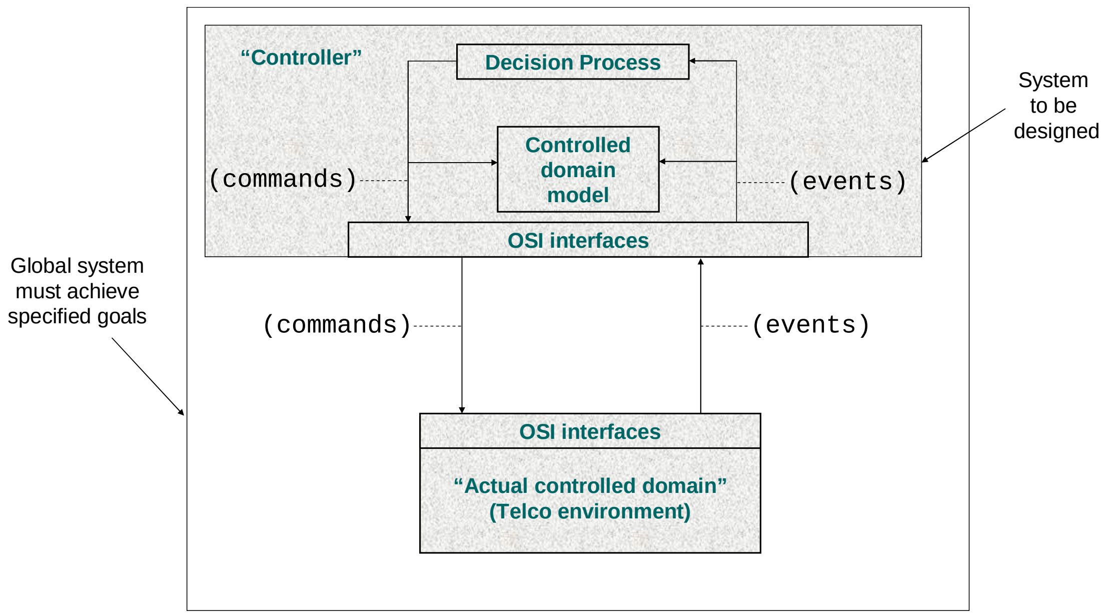


Figure 71. Overall view of a network management system

The team model proposed in this framework considers that the System Architect should manipulate an integrated design framework that integrates three mathematical techniques (see system design model in Fig. 70):

- automatic control theory
- Predicate-transition Petri net theory
- logic programming.

Automatic control theory for discrete-event systems [Ram89] provides the unifying theme that binds the various parts of the global system. It was envisaged that this theory would provide a formal framework for the design of large systems with feedback control. For instance, Figure 71 shows this overall view of a network management system structured according to the concepts of automatic control theory. This system is composed of three parts: (1) the controlled domain, which consists of network elements, operations support systems, and network operators equipped with computer terminals. (2) the controller that must be designed and implemented so that the global system, which is composed of the controller working in conjunction with the controlled domain, achieves a specified goal, and (3) the interfaces and interconnections that exist between the various components for which communication information must be defined.

According to the authors view, Predicate-Transition Petri Net Theory [Gen86] should be used to describe both the structure of processes (how processes are linked together) and their behaviour (how processes change their state). This is, Predicate-transition Petri enables the framework to introduce, in a formal manner, the concepts of individuals with changing properties and changing relations into the Petri net theory. Intuitively, predicate-transition Petri nets would provide a dynamic perspective of the common notion of a relational structure. A structure is a tuple of objects comprising a set of individuals, called the domain of the structure, together with functions and relations in that domain. Operators (function symbols) and predicates (relation symbols) should form the vocabulary of the language in which one talks formally about structures, i.e., about the properties and relations of the individuals. The language used is first-order predicate logic. A dynamic structure is characterized by the fact that some relations are variable in the sense that their extensions (i.e., the set of individuals for which the corresponding predicates are true) may vary from state to state due to the occurrence of processes in the modelled system. The changes in the relation extensions are defined by transition schemes.

Finally, Bean et al.'s framework integrated Logic Programming (Prolog) [Ster86] as a feasible formalism to provide the descriptive framework. Logic programming is based on the concept of using predicate logic as a programming language [Ster86]. Logic programming languages have more natural semantics than other programming languages. Due to their high-level declarative semantics, they are "almost" specification languages. Moreover, they also have an operational meaning. In addition, the logic programming language Prolog provides a suitable framework for the realization of meta-linguistic abstractions, i.e., the establishment of new descriptive languages that are particularly well suited to the problems at hand. For example, such a meta-linguistic abstraction can be achieved by constructing a specialized language in Prolog. This is done by writing a Prolog interpreter for the specialized language. This interpreter then, when applied to an expression of the specialized language, would perform the actions required to evaluate the expression. User requirements must contain the interpreter defining the language used for the description of the controlled domain and the goals. Since the interpreter is written in Prolog, this language plays the role of a *lingua franca*, i.e., a language that is assumed to be understood by everyone.

Although these efforts were aligned with the TMN (Telecommunications Management Network) standards, these initiatives do not provide any clue about how these goal-oriented requirements could be addressed in policy-based systems or the implications for its application to such systems, amongst other reasons because by the early nineties, the idea of policy-based management was not as mature as we could consider it by the time of the publication of this Thesis. In addition, there is no evidence of any implementation that brings all the concepts considered in the authors' framework, or a performance evaluation that makes their proposals trustworthy. In conclusion, there is not explicit evidence of the evolution of this initiative.

Our holistic view of policy refinement could be considered as a specialisation of the above initiative, narrowed down to address the refinement process in policy-based network management systems. We provide a methodological approach and a functional solution that makes a collaborative environment for shared work amongst the actors involved in the policy refinement process, from the design/implementation of the system, up to the operation of such system.

6.2.2 The KAOS methodology

Since the early stages of policy-based management, the research community envisaged the need to specify policies at an abstract level and progressively refine them into enforceable policies. The KAOS methodology approach was first brought into the policy community by van Lamsweerde [Lam99] as an alternative to ground the policy refinement problem. The KAOS approach [Dar93], [Dar95], [Dar98], [Lam95] used in this Thesis is one of the most extended approaches in the Requirements Engineering area as it provides support to formalise goal-oriented requirements specifications, and a rather complete and mature goal-based reasoning technique to refine goals (which are related to abstract policies) into lower-level ones.

We have described extensively the foundations and principles of the KAOS methodology throughout this Thesis, concretely in Section 2.2.3. The main limitation of this methodology is that it does not provide support to relate the target system behaviour to goal fulfillment finding. Consequently, the use of the KAOS methodology to refine policies as a stand-alone methodology has never been demonstrated in management contexts, more concretely in policy-based systems.

The holistic nature of the work presented in this Thesis overcomes the limitations of the KAOS approach to refine enforceable policies from abstract goal-oriented requirements in a systematic manner. In this sense, two key issues are particularly relevant:

- Methodology-wise, we have extended the KAOS goal refinement process to make it suitable and affordable in network management systems. More concretely, we have provided generic guidelines to address goal refinement for the later systems.
- Analysis-wise, we explicitly use the linear temporal logic foundations of the KAOS approach as the base for analysis techniques that enable automatic production of enforceable policies from goal graph structures elaborated through goal refinement. Hence, we make use of the KAOS foundations to overcome its limitations in favour of the policy refinement problem.

Other efforts targeting the limitations of the KAOS approach to address the policy refinement problem are described in the following Section.

6.3 Abduction and Event Calculus-based approach to Policy Refinement

An effort that has taken the KAOS approach to address the policy refinement problem is work by Bandara et al., [Ban04]. The authors claim to have proposed a goal-based approach to policy refinement, grounded in the KAOS methodology. This approach proposes a formal specialisation of the ideas presented in [Lam99] to use the KAOS approach [Dar98] to produce Ponder policies from KAOS goal graph structures. In this sense, the contribution of Bandara et al., [Ban04] has been acknowledged as a substantial contribution to the policy refinement area. In [Ban04], the Event Calculus is used to model both system behaviour and policy such that formal reasoning techniques can be used to analyse policy specifications. Event Calculus and Abduction are used to formalise the refinement analysis.

The Event Calculus [Efs02], [Mil99] is a logic-based formalism that provides a theoretical framework for representing, specifying and reasoning about dynamic systems. The Event Calculus description includes a collection of axioms (sentences) that describe general principles for deciding specific features of event-based systems. Each specification includes a collection of sentences, describing the particular effects of events or actions in an event-based system.

The authors use a variation of Event Calculus consisting of (i) a set of time points (that can be mapped to the non-negative integers); (ii) a set of properties that can vary over the lifetime of the system, called *fluents*; and (iii) a set of event types. In addition the language includes a number of base predicates, initiates, terminates, holdsAt, happens, which are used to define some auxiliary predicates and domain independent axioms.

These are summarised below [Ban04]:

Base predicates:

initiates(A,B,T)	event A initiates fluent B for all time > T
terminates(A,B,T)	event A terminates fluent B for all time > T
happens(A,T)	event A happens at time point T
holdsAt(B,T)	fluent B holds at time point T. This predicate is useful for defining static rules (state constraints)
initiallyTrue(B)	fluent B is initially true.
initiallyFalse(B)	fluent B is initially false.

Auxillary predicates:

clipped(T1,B,T2)	fluent B is terminated sometime between time point T1 and T2
declipped(T1,B,T2)	fluent B is initiated sometime between time point T1 and T2

Domain independent axioms:

```
holdsAt(B, T) ∧ ¬ clipped(T, B, T1) ∧ T<T1         → holdsAt(B, T1)
initiates(A, B, T) ∧ happens(A, T)
        ∧ ¬ clipped(T, B, T1) ∧ T<T1               → holdsAt(B, T1)
¬holdsAt(B, T)  ∧ ¬ declipped(T, B, T1)
        ∧ T<T1                                     → ¬holdsAt(B, T1)
terminates(A, B, T) ∧ happens(A, T)
        ∧ ¬ declipped(T, B, T1) ∧ T<T1             → ¬holdsAt(B,T1)
```

In the classical form of the Event Calculus used by Bandara et al., theories are written using Horn clauses and the frame problem is solved by circumscription, which according to the authors, allows the completion of the predicates initiates, terminates and happens, leaving open the predicates holdsAt, initiallyTrue and initiallyFalse. This approach allows the representation of partial domain knowledge (e.g. the initial state of the system). Formulae derived from Event Calculus derived from the circumscription of the EC representation. To provide an implementation of such a Calculus in Prolog, the Bandara et al.'s approach uses pos and neg functors. The semantics of the Prolog implementation assumes the Close Word Assumption (CWA) and Herbrand models where predicates are appropriately completed. The use of pos and neg functions on the *fluents* allows the authors to keep open their interpretation of being true/false, in the same way as circumscription does in the classical representation. In this way it is guaranteed that the implementation of the Event Calculus is sound and complete with respect to the classical EC formalisation. The interested reader can find the correspondence between the classical EC with circumscription and the logic program implementation in [Mil99], [Mil02]. Moreover a brief description of how this classical EC is used in the context of refinement [Ban03] is described bellow.

The reasoning used in the author's approach is abduction. It is used to determine the sequence of events that need to occur such that a given set of *fluents* will hold at a specified point in time. The overall claim for refinement is that at a given level of abstraction, there will be some description of the system (SD) and the goals (G) to be achieved by the system. The relationship between the system description and the goals is the Strategy (S), i.e. the Strategy describes the mechanism by which the system represented by SD achieves the goals denoted by G. Formally this would be stated as [Ban04]:

(1) - $SD_X, S_X \quad G_X$

X is a label denoting the abstraction level.

As the authors certainly claim, in this approach it is expected that the user should provide a representation of the system description, in terms of the properties and behaviour of the components, together with a definition of the goals that the system must satisfy. While the behaviour of the system is defined in terms of the pre- and post-conditions of the operations supported by the components, the goals to be satisfied can be defined in terms of desired system states.

Once the user has provided this information, a transformation step is needed into a formal representation that supports automated analysis. Given the relationship between the system description, strategy and goal defined in (1) above, the authors propose abduction to programmatically infer the strategies that will achieve a particular goal as graphically shown in Figure 72 [Ban04]. Additionally, the properties of the goal decomposition approach described previously are used to decompose the system description and strategies as follows: (2) - $G_{X1}, G_{X2}, \ldots , G_{XN} \models G_X$ Goal Decomposition

(3) - $SD_{X1}, S_{X1} \models G_{X1}$

$SD_{X2}, S_{X2} \models G_{X2} \ldots$

$SD_{XN}, S_{XN} \models G_{XN}$ (from 1)

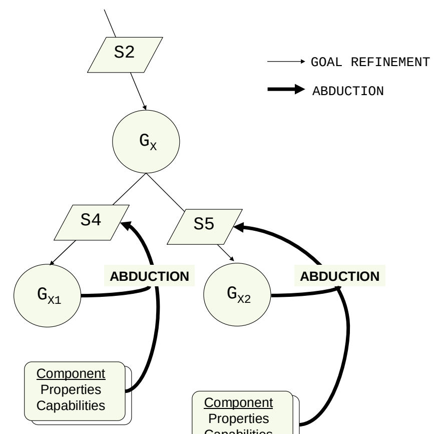


Figure 72. Derivation of strategies from goals and system description

The authors claim that if there is some combination of lower level goals from which the original goal can be inferred, then for each of these sub-goals there must be a corresponding strategy and system description combination which will achieve it. Therefore, provided the goal decomposition is correct, intuitively the combination of the lower level system descriptions should allow inference of the abstract system description and similarly the combination of the lower level strategies should allow inference of the abstract strategy. The derived strategy can be represented using the following syntax [Ban05a]:

```
Strategy AchievedGoal
     OnEvent         Events derived from transitions with system events.
     DerivedActions  Actions derived from transitions with operation invocations.
     Constraints     Constraints derived from guards.
```

As the authors claim, the exact circumstances, in which a strategy should be encoded as a policy, rather than system functionality, will depend on the particular application domain. Once a strategy is identified, the authors use it in the action clause of the final policy. The domain hierarchy is used to identify the exact objects in the system that correspond to those entities mentioned in the high-level policy which are used in the subject and target clauses of the final policy. Finally the event and constraints of the high-level policy are mapped, by the user, into the final policy.

Clearly, the approach reported in [Ban03], [Ban04] and the work reported in this Thesis, both are committed to overcome the limitations of the KAOS methodology in favour of the policy refinement process. Moreover both approaches have remarkable differences.

In terms of the goal refinement methodology, we consider that [Ban04] does not make clear how the goal refinement process could be assessed in network management systems. In this Thesis we have proposed the guidelines to make use of inherent features of network management systems to drive, together with the KAOS refinement patterns, the goal refinement process.

[Ban04] mainly concentrates on the analysis techniques aimed at acquiring the so-called strategies. Nevertheless, it is unclear how the linear temporal logic foundations of the KAOS goal graph structures are used in their analysis techniques to drive the acquisition of policies from high-level abstract requirements. Namely, we find it unclear how the ad-hoc machinery developed to abduce strategies based on EC-based specifications work as the approach does not address explicit temporal execution of goals [Ban05a]. This makes it difficult to think of the scalability for large refinements whilst explicit execution of goals is pivotal in the work presented in this Thesis.

Our framework has been conceived to define the behaviour of the system as a reactive system, namely by finite state machines, collaboration diagrams, etc, compiled in standard UML standard notations, hence making it easier and friendly to use for network administrators and developers. More important, the analysis techniques used to abstract the system behaviour that should fulfill high-level goals, are different in nature. While Event Calculus and abduction is used in [Ban03],[Ban04] to infer the sequences of actions that achieve particular goals, our approach goes through pure LTL-based state exploration to obtain the necessary system behaviour (actions, events and states) that the underlying managed entities should exhibit as to fulfill lower-lever goals elaborated through temporal refinement patterns.

Finally, in terms of functionality, [Ban05b] acknowledges that the tool implementation developed on purpose to verify their analysis techniques, is not considered a fully functional system, whilst we have made evident the functionality of the prototype implementation to refine, enforceable policies from abstract KAOS goal graph specifications. To the time of the publication of this Thesis, there is no evidence of performance evaluation and consequently we cannot provide qualitative evaluations between the EC-based approach and the work carried out in this Thesis.

6.4 Automated generation of policies from process specifications

In our work we introduced an ad-hoc process aimed at abstracting enforceable policies from system behaviour findings. This key element of the refinement process has no precedent in the policy refinement area as it has been developed exclusively for our methodological approach. The closer concept related to the above is the automated generation of policies from process specifications [Dan04]. The following is work by Danciu et al with this regard.

6.4.1 Process and Policy Characteristics

Danciu et al., propose a set of patterns for the automated generation of policies from process specifications. To achieve the translation between process and policy specification, common characteristics of processes and policies should be leveraged.

As the authors describe in [Dan04], processes are described as sets of entities, events and actions. Entities can be either human personnel or managed objects (MOs). Entities may generate events or execute actions. Examples for entities are: ’change manager’, ’user’ and ’application server’. Events communicate a change of state in an entity that is relevant to management. Examples for events are: ’document changed’, ’user requests service’ and ’application server restarted’. Actions are performed by entities in response to events. They can perform a change of state in entities. A special form of action is the generation of an event. Examples for actions are: ’update document’, ‘authenticate user’ and ’restart application server’.

In addition, policies are also derived considering Subject, Target, Event, Action and Condition. Subject designates the entity to which the policy applies. Target identifies an entity that is affected by the policy (i. e. the target of the action of the policy). Event describes the event which causes the policy to be evaluated. Action is the operation performed on a target to enforce the policy. Condition is the constraint under which the policy is enforced. While only the event, action and condition fields are ubiquitous, the introduction of subjects and targets is necessary for referencing entities.

6.4.2 Mapping Management Process to Policies

Having defined common characteristics of process and policies, mappings that serve as generic translation patterns should be defined.

The characteristics of management processes are used as a common base for the targeted mapping. Figure 73 shows an overview of the mapping process to policy characteristics as illustrated in [Dan04]. Entities are mapped to subjects and targets. Thus, persons or MOs can have policies specified for them (subject). They can also be affected by policies (target). Actions in process semantics correspond to actions of policies. An action can be anything between one single function call and entire scripts executed on a target MO. Events in processes match the event-constraint-pair in policies. While it is common for process specifications to contain conditions, these can be represented using a sufficiently detailed typing of events. Such a typing will yield a very large number of event types and is, in consequence unsuited for implementation. An event is considered to have happened, when an event notification (e.g. a signal) has been received, and the conditions (if any) yield hue.

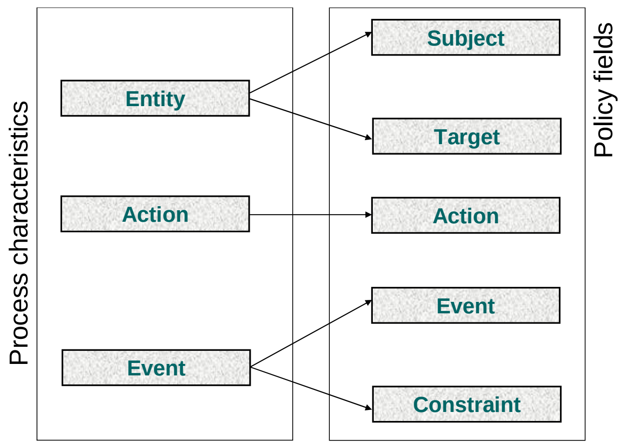
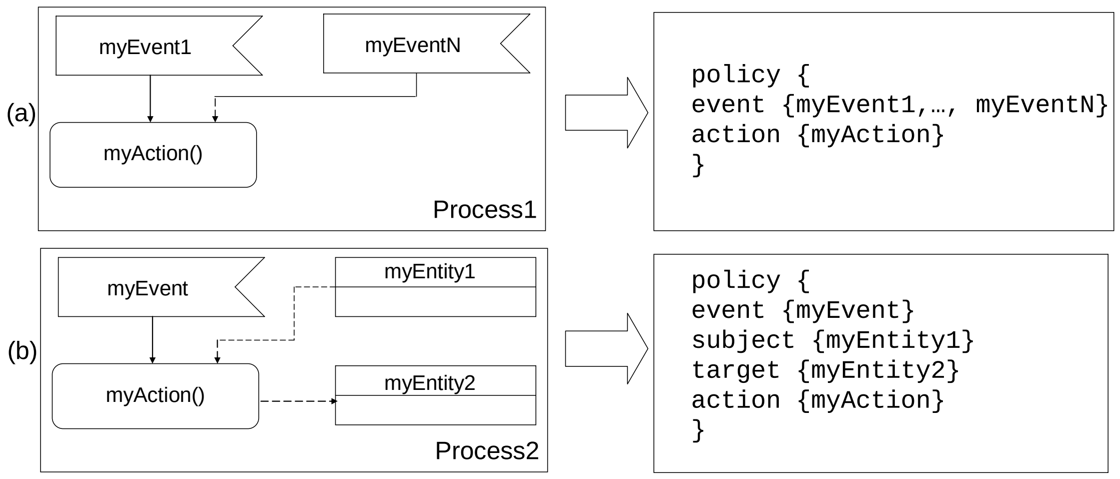


Figure 73. Mapping process to policy characteristics

Relevant of this work are the translation patterns to generate policies from process. The most fundamental structure in an event-driven process is the unconditional execution of an action in response to an event. As shown in Figure 74a, several events can be specified to trigger the execution of the action (the comma in the event list implies or). In most cases, the entity responsible for the action and the target of that action need to be specified as well (Figure 74b). The first case results in a single policy with only the *event* and *action* fields present. In the second case, *subject* and *target* of the policy are determined by examining the direction of the arrows that connect the entities to the action. An entity having an arrow pointing *towards* it (UML output object) results in a policy target, while entities having arrows pointing *away* (input object) from them (and towards the action) result in policy subjects. The policies resulting from the patterns are shown in order on the right part of Figure 74.

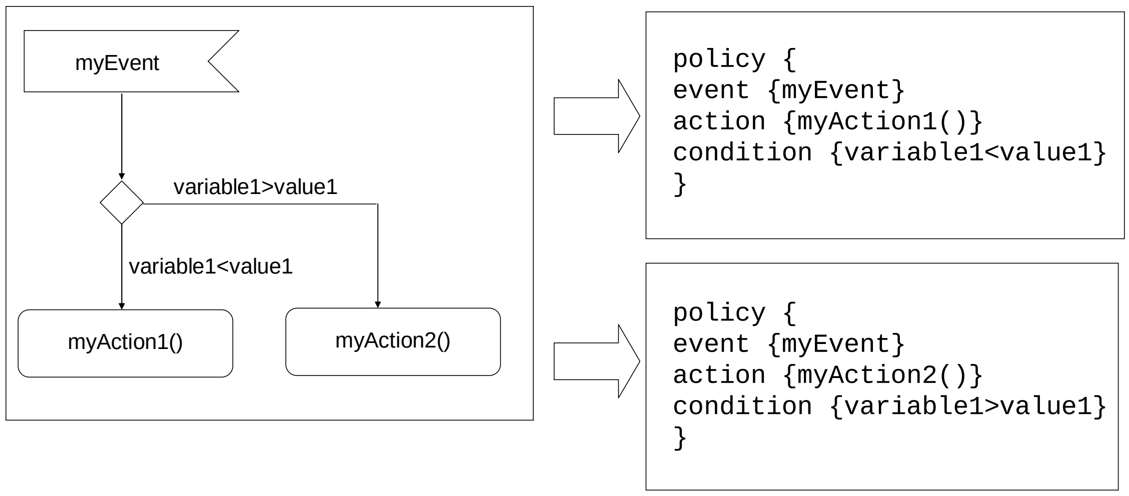


```
policy {
event {myEvent1,…, myEventN}
action {myAction}
}
```

```
policy {
event {myEvent}
subject {myEntity1}
target {myEntity2}
action {myAction}
}
```

Figure 74. Basic patterns

The process in Figure 75 chooses between two different actions depending on the value of a predicate. Two policies are generated from this process, differing at least in the condition field. The general case has several decision branches, each carrying its guard condition. In this case, a number of policies equal to the number of branches needs to be generated in such a way that every policy includes the condition of the decision branch it corresponds to. Any condition can be formulated for a decision branch; for clarity, Figure 75 is constrained to an `<expression> <operator> <expression>` pattern as described in [Dan04].



```
policy {
event {myEvent}
action {myAction1()}
condition {variable1<value1}
}
```

```
policy {
event {myEvent}
action {myAction2()}
condition {variable1>value1}
}
```

Figure 75. Decision pattern

As the authors certainly claim, this approach is committed to generate policies from process specifications. With this regard, the approach looks like a classic template-based policy generation mechanism. This is, the attributes of the policy fields are restricted to pre-established values. Contrary, the policies produced in the work described in this Thesis do not obey to any pre-established template. We encode policies systematically using a translation process applied directly over policy-restricted system behaviour. The authors in [Dan04] recognise their approach consists of mapping management process into operational policies although it has been acknowledged as a substantial contribution to the policy-based management area since it enables the production of large amounts of policies. Another concern with this approach is that it is yet unclear how policies are committing to which or what administrative objectives. More concretely, it is not clear how the administrative parties should align process definition with business objectives or administrative views. By the time of the publication of this Thesis, there is no evidence of performance data or dissemination results for further analysis to carry out comparisons between this approach and the work carried out in this Thesis.

6.5 POWER Prototype

Functional prototypes tailored explicitly to address the policy refinement problem are rather scarce and still lacking in the policy research community. Work by HP Laboratories [Cas00] is one of the few policy refinement approaches hitherto implemented for this purpose.

6.5.1 Refinement Philosophy

POWER is developed from the idea that a policy should be looked upon as "the *constraints* and *preferences* on the state, or the state transition, of a system, and is a guide on the way to achieving the overall objective which itself is also represented by a desirable system state" [Goh97]. The authors use the concept of constraint for the components in policy description; the context in which a policy operates; the triggering event which kicks policy into consideration in the context; and the policy statement. In the POWER approach, the refinement of policy consists of two aspects: the refinement of policy context by making constraints more specific, and refinement (constraining) of objects used in the policy.

Similar as work by Bean et. al [Bea93], POWER separates responsibilities of policy making persons whom are respectively called expert and consultant. The "expert" is the person with deep domain knowledge, such as that in the field of security, or network QoS related mechanisms. A "consultant" is the person who has deep knowledge of the business for which policies are to be established. The expert deals mainly with policy function to mechanisms mapping and the consultant mainly deals with business to policy function mapping. With this separation, the expert creates policy templates, which will be used by the consultant to create policy according to the business needs.

6.5.2 Prototype Architecture

The architecture of POWER [Cass00] is graphically shown in Figure 76 for which a brief description is provided thereafter.

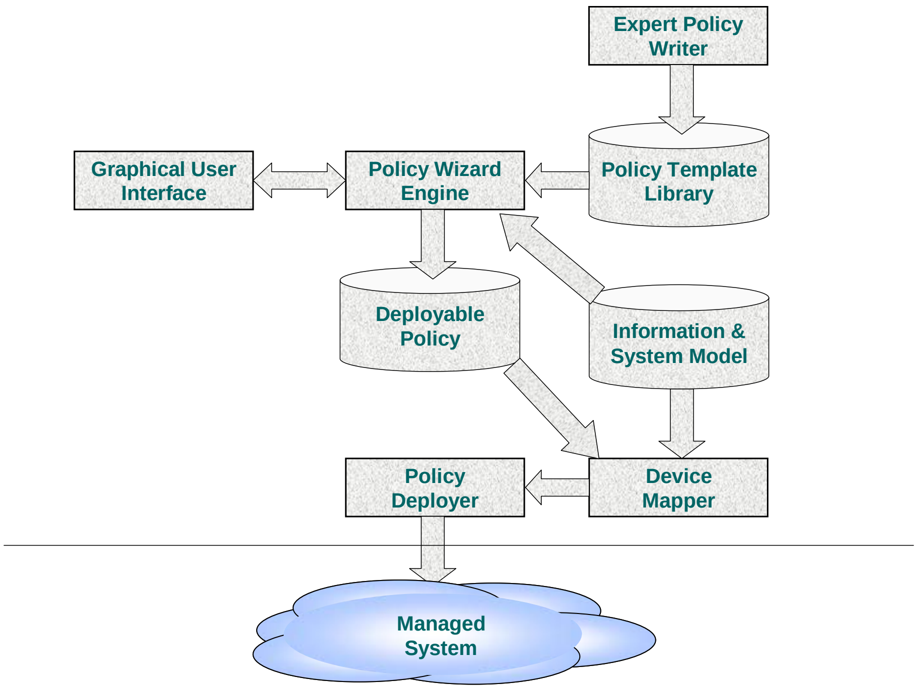


Figure 76. POWER Policy Authoring Environment Architecture

A key component is the Policy Template Library. It is a collection of *policy templates* which have been created by the expert of the domain that the policies are meant for. Each template is a "package" that describes the policy according to certain principle, and the way a consultant, using the authoring environment, can refine it. The authors propose to achieve this through embedding the refinement steps and instructions in the template as components. The policy template is implemented with Prolog and can be manipulated by the Policy Wizard Engine. Its components can be classified accordingly to their usage [Cass00]:

- Policy Statement: The description of the policy. These are predicate logic statements with several views, one of which is "natural-language like" and is exposed to the policy user.
- Policy Context: The description of contextual constraints within which the policy will operate. The contextual information allows one to arbitrarily define a domain *dynamically* within which the policy statement is valid.
- Informational components: they provide extra information to the policy user. For example the "abstract" and the "description" contain descriptive text about the meaning of the policy.
- Procedural components: they have embedded process instructions used to drive the "refinement flow". For example the "sequence" component defines the steps the Policy Wizard Engine will lead the consultant through.

Another key component is the Information and System Model ISM. It models the information in the underlying policy-controlled environment. It is implemented using the Common Information Model from the Distributed Management Task Force CIM [DMTF-CIM], with extensions in the area of business and organisation model. Included with the implementation will be the low-level linkage of object classes to information sources that creates the mapping to managed objects. The authors implement this as a set of Prolog statements that can be easily accessed by the Policy Wizard Engine.

The Policy Wizard Engine (PWE) is the heart of POWER. The authors acknowledge it as the combination of: (i) A Prolog inference engine; (ii) An interpreter that manipulates a policy template according to the embedded information, and provide support to the graphical user interface; (iii) A module that interacts with the Information and System Model using a defined API; (iv) A module that deals with "deployable policies": (v) Procedures that interact with the "Policy Deployer" using a defined API.

The PWE loads policy templates from the library. Through the use of a GUI, a relevant template is selected, and by interpreting the embedded information in the template, the PWE guides the consultant in the refinement process to ensure that:

- Within an abstract policy, objects, which can be made more specific through the selection of its sub-class
- Legitimate additional constraint can be included as contextual information.

At the end of the refinement process the PWE saves the policy either for further refinement or for it to be used in deployment.

The Graphical User Interface hides the low-level policy details, such as the policy template infrastructure and Prolog programming language, in order to present an easy and simplified way to access the system functionality.

Regarding the Deployable Policies Database, a policy is deployable only when a set of real world system objects can be found and for which configuration is specified. The system stores those policies in order to be uploaded by the "Policy Deployer" and be deployed, and/or to be available to the consultant or other system modules for further manipulations.

The authors propose a device mapper that in turn uses the Information and System Model ISM to convert from a policy description in the form of a policy statement and context containing variables into a series of system specific function calls.

The prototype implementation presented in this Thesis differs to POWER in the sense that the latter is an environment in which the user is guided to choose policies from pre-designed policy templates tailored for specific use. Instead, we use a pure goal-oriented approach in which the administrator developer documents both, the capabilities of the target system and the options with which the administrator consultant can choose the administrative views of how to handle the system at operation time. Hence, we do not limit the options of the administrative parties to the use of templates of any kind.

The major limitation of the POWER toolkit is its inherent template-like functionality which constrains it to work under limited scopes. In addition, there is no evidence of the implications of its use in a concrete application domain, or a fully implemented prototype including all the architecture details described earlier. Consequently, to the time of the publication of this Thesis, there is no evidence of performance data that allow assess qualitative comparisons between POWER with our refinement approach. Apart from POWER, there is no evidence of any other functional solution to policy refinement.

6.6 Other refinement efforts

By the time of the publication of this Thesis, we are being aware of a rather increasing interest of analysis techniques tailored to tackle policy refinement. This is at some point needed for the success of policy-based management. The remaining of this section provides some recent advances on policy refinement.

6.6.1 Automated Policy Refinement for Managing Composite Services

Carey et al. [Car04], [Car05] propose to manage composite services automatically with the use of a policy refinement approach. In this context, policy refinement is applied to mapping high level goals of a composite service automatically down to low level policies which interact with the service and previously existing constituent services. The authors propose an architecture that works as along side a workflow-style composite service execution engine. It also includes a policy refinement engine, a policy execution engine and a state machine. The policy refinement model described in [Car04] is graphically shown in Figure 77.

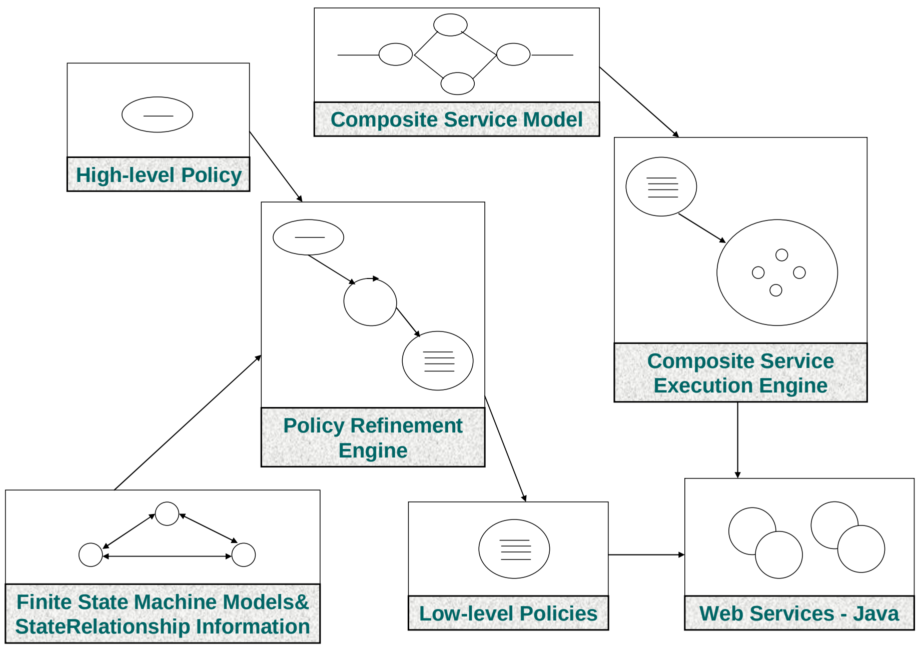


Figure 77. Policy refinement model for managing composite services

The architecture of this work assumes that the execution of the service composition is performed by a workflow engine. This engine uses a sequence description for a composite service to create scripts with rules that activate the services according to the sequence description given.

The authors propose that service implementers use Finite State Machinces (FSMs) to expose the, largely non-functional, elements of the implementation's behaviour that are intended to be managed via policy rules. The states of a FSM represent the non-functional aspect configurations of the service it describes. A transition between two states in a FSM represents an action needed to be performed in the service to move from one state to the other. For composite services, the service's non-functional aspects are described in terms the non-functional aspects of its constituent services. The relationship information between the composite service's states and the constituent service's states is referred to as *state relationship information*.

Policy is proposed in this work for managing the services by the use of the FSMs, where it can monitor for a specific state and when this state is reached, it can perform the necessary action to reach the desired state.

Prior to the execution of the composite service, the state machine should load the FSM models for each of the constituent services together with a default state for each of the FSMs. During the composite service execution, the state machine changes the current state of a FSM when an action for that state transition occurs on an active service.

The architecture of this proposal has a policy refinement engine. The authors acknowledge a goal as a high level policy with event, condition and action, expressed only with the states defined in the FSM for the composite service. When applied to the framework along with a composite service, a goal is refined into intermediate policies and subsequently into service policies. This occurs prior to the execution of the composite service.

In this refinement view, the refinement of a high level policy to an intermediate policy is achieved with the aid of the composite service map, which describes how the states in the composite service's FSM are mapped to states of its constituent services. The high level policy is therefore refined into an intermediate policy, an aggregate policy expressed in terms of states from constituent services. The aggregate intermediate policy can have several events, conditions and action.

The transformation of an aggregate intermediate policy into service policy is more complex, it entails the creation of a service policy, a low level policy, for each of the intermediate's event, condition and action. The refinement of an aggregate intermediate policy also generates a service meta-policy. The service meta-policy is the backbone of the service policies as it ties them together.

Regarding the conditions of the intermediate policy, for each of these conditions a service policy is generated and is triggered when the state specified by the intermediate condition is encountered. As its action, it sends a notification to a service meta-policy that this condition has been reached. The same applies to the intermediate policy's events. For each action of the intermediate policy, a service policy is generated and is triggered when it receives a notification from a service meta-policy. As its action, it performs the action needed to attain the desired state specified by the intermediate policy action. The generated service meta-policy for the aggregate intermediate policy expects notifications from all the 'condition' service policy and 'event' service policy as its condition. As its action, it sends notifications to all the 'action' service policies.

By the time of the publication of this Thesis, to the best of our knowledge, there is no evidence of performance evaluations of the refinement approach nor details of the implications of the approach towards solving the refinement problem in this application domain.

6.6.2 Multi-layer modeling refinement for network security management

Albuquerque et al. [Alb05] propose a refinement mechanism based also on a multi-layer system modeling approach addressing the network security management domain. The authors address the refinement problem by modeling security systems, based on the concepts of policy-based management and model-based management.

Model-Based Management (MBM) is proposed to support policy-based management by the use of an object-oriented model of the system to be managed. Based upon this model, policy refinement is accomplished such that configuration parameters for security mechanisms can be automatically derived. The structure of the model as provided in Alb05 is shown in Figure 78, where three abstraction levels can be distinguished: *Roles & Objects* (RO), *Subjects & Resources* (SR), and *Processes & Hosts* (PH). Each level is a refinement of the superior level in the sense of a "policy hierarchy".

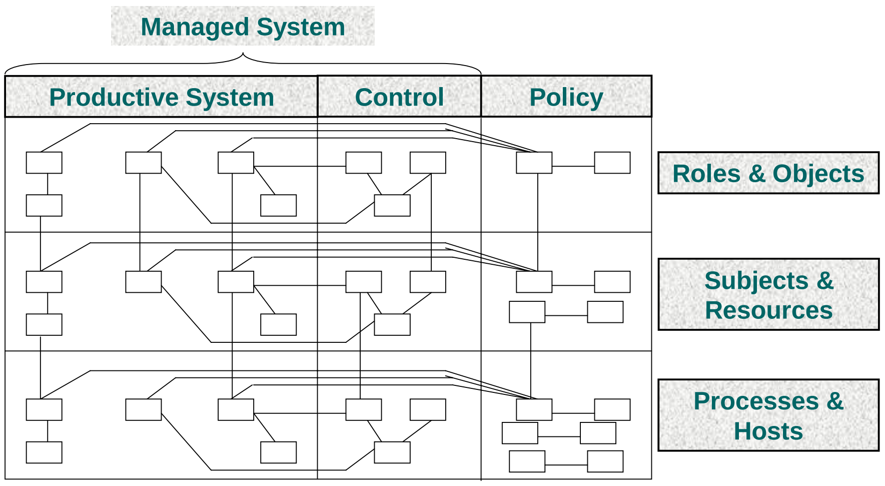


Figure 78. Overview of Model Based Management

The higher level Roles & Objects (RO) offers a business-oriented view of the network whereas the lowest level is related to a technical view. The vertical subdivisions differentiate between the model of the actual managed system (with productive and control elements) and the policies that regulate this system. This last category encompasses requirement and permission objects, each of which refers to the model components of the same level and expresses security policies.

The second level Subjects & Resources (SR) offers a system view defined from the standpoint of the services that the system will provide, and it thus consists of a more complex set of classes. In this view, objects of these classes represent: (a) people working in the modelled environment; (b) subjects acting on the user's behalf; (c) services in the network that are used to access resources; (d) the dependency of a service on other services; and lastly (e) resources in the network.

The lowest level Processes & Hosts (PH) is responsible for modelling the mechanisms that will be used to implement the security services defined in SR. The PH level will have even more classes than before, representing for instance the hosts, with their respective network interfaces and processes.

The claim is that Model-Based Management (MBM) also provides a support tool, which, at first, assists the user in the modeling of the system by means of a graphical editor with additional functions for the checking model-dependent constraints. Once the system modelling is finished, the tool performs an automatic refinement of the abstract security policies (in the RO level), through the intermediary levels (SR and PH), until achieving configuration files for the supported security mechanisms.

The claim is that the above modeling technique makes it possible an automation of the building of a policy hierarchy on the basis of a system's model that is structured in different abstraction levels. In this process, the analysis of the system's objects, relationships and policies at a certain abstraction level enables the generation of lower level policies, based also on the system's model in the lower level and on the relations between entities of the two layers. As such, the model entities of a certain level and their relationships supply the contextual information needed to automatically interpret and refine the policies of the same level.

A relevant issue of this approach is that the administrative parties must manually establish relationships between entities of the different layers of the model. This is at some point unavoidable in any automated approach. Nevertheless, the implications of this policy refinement approach in a concrete application domain are still unclear. In addition, as a template-like approach, it is limited to pre-established conditions of policy refinement.

6.6.3 Automated decomposition of access control policies

Access control policies are concerned with the definition of access control rules that define who is allowed to do what to which resources [SuL05]. In this domain, policy refinement is receiving some attention from the research community. For instance, Su et al. [SuL05] address resource management for distributed applications. In this domain Policy Decision Points (PDPs) can be used by each resource. From the view that each site can have its own PDP and that common policy can be distributed to each site based PDP, the claim is that a common high level policy for the application can be decomposed (refined) into site-specific policies, which are then distributed to each site and only contain policy information relevant for controlling access to that site.

The work introduce the concepts of abstract resource types, concrete resource types, resource type hierarchies, resource instances, policy hierarchies and policy decomposition rules. The multiple resources of a distributed application are seen as having a hierarchical structure. Policies are needed to control access at each of the levels. At the highest level, the policy is concerned about controlling access to a single abstract resource, for example a particular Grid application. At the lowest level, the policy addresses controlling access to specific concrete resources, such as servers and file stores which make up the abstract resource.

Intermediate levels are possible, for example, a computer cluster or a distributed database. The claim is that policies at any level for whatever abstract or concrete resources can be automatically produced from the high level policy based on policy decomposition rules and resource type hierarchies.

The authors propose resource type hierarchies that should describe how the high level abstract resources are constructed from their lower level concrete and abstract resources. Resource type hierarchies also say what actions (or methods) each of the resources support. A resource instance is an instance of a resource type hierarchy. Multiple instances of the same type hierarchy can occur.

Refinement-wise, Policy decomposition rules define how high level policies transform into their low level ones so that the policy decomposition (refinement) is realized based on these rules and a resource type hierarchy. By means of policy decomposition, see Figure 79 [SuL05], existing access control decision systems such as PERMIS and Akenti [Jho98] are able to be applied to multiple distributed resource applications. Based on an access control policy P for an abstract resource R, which is a distributed resource, a group of low level policies p1-p6 for each component of R can be produced. To do so, relationships between the resources are established. In the example shown in Figure 79, the resource R contains six sub resources, which may be decomposed further into other sub-resources (not shown). This process is carried out recursively until all sub-resources become site specific concrete resources, where the low level policies are then able to be used to control access to them. This resource decomposition and low level policy production defines a simple policy refinement process. This policy refinement process is claimed to produce a policy for any resource at any level and ensures that each stage of the decomposition is correct and consistent.

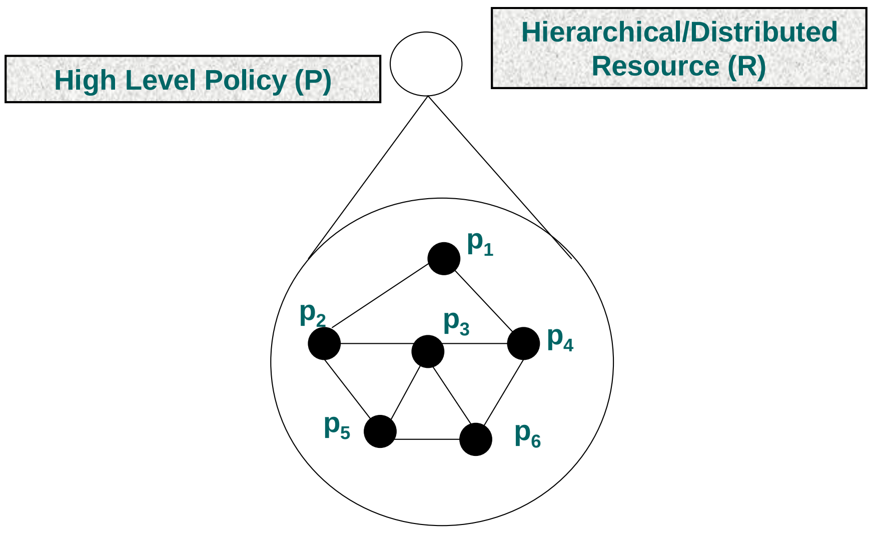


Figure 79. Producing Low-level Policies (p1 to p6) from a high level policy p

As for the description of the refinement rules, this approach is apparently inspired in classical template-based approaches although the authors have extended it to a hierarchy-like approach. The authors consider multiple resource types of distributed applications to build up a resource type hierarchy in distributed applications. From these hierarchies, policies belonging to a resource type are refined into policies applied to specific resource instances. The authors claim that low-level policies are simpler in terms of the number of policy components contained in them. The work reported in [SuL05] is an ongoing work and there is not evidence of a complete approach to policy refinement applied to this domain although the ideas presented above may pioneer future studies in this field.

6.6.4 Synergy between conflict analysis and policy refinement

More recently, the synergy between policy refinement and conflict analysis has received some attention from the research community. Work by Davy et al.[Dav06] describe an architecture for enabling application specific conflict prevention via model driven policy refinement. A refinement algorithm can retrieve relevant information from the information model based on defined policy so that it links loosely defined high level policy to behavioural constraints contained in the information model. This process enables policy enforcement to be more aware of application specific constraints, leading to a more trustworthy and dependable policy based management system.

In the above context, policy refinement involves the specification of additional condition clauses within the policy, which subsequently allows the detection of conflicts with other policies that would otherwise have gone undetected by standard policy conflict detection algorithms. In cases where system information models describe constraints relating to the operation of managed entities, relevant policies are augmented with conditions reflecting these constraints, so that they would not be enforced in a manner that results in these constraints being violated. System constraints in the information model are defined by the system architect who has expert knowledge in the functionality of the system being modelled. These system constraints may come in the form of action pre-conditions, invariants, or post-conditions. System constraints defined within the information model supply implicit knowledge not usually available to the policy authoring process. The work introduce an automated policy refinement process which obviates the need for policy authors to be cognisant of the detailed constraints on system operation, but which outputs policies that are sufficiently well specified that policy conflict detection processes can be effective and efficient. Figure 80 illustrates a policy-based system incorporating model-driven conflict prevention as illustrated in [Dav06].

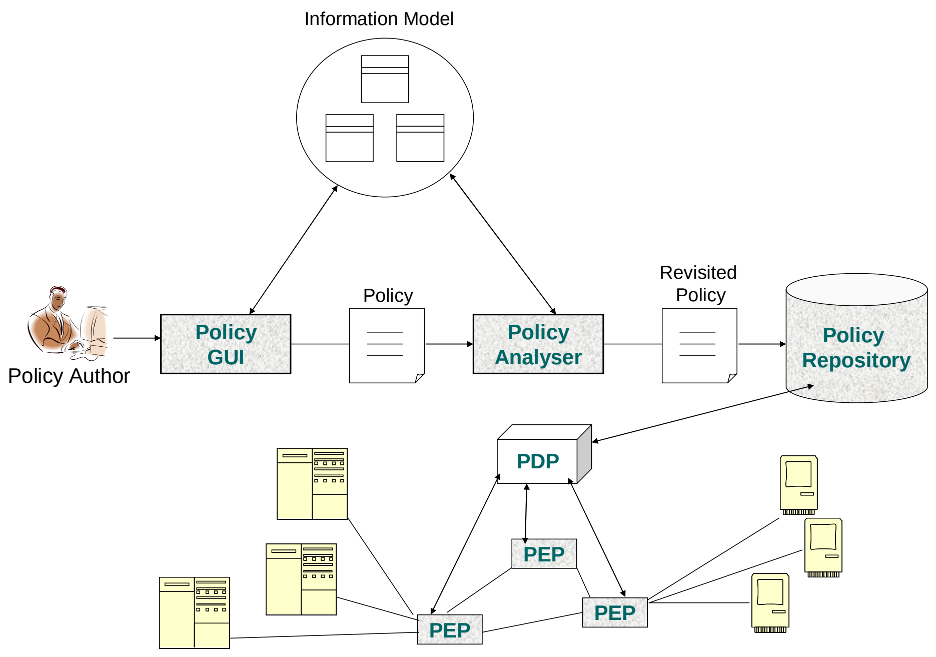


Figure 80. Policy system introducing model-driven conflict prevention

In this view of conflict prevention during the refinement process, policies created or modified by policy authors are expressed in strict accordance with the terms used in the information model, since the policy GUI is tightly coupled to the information model. Once created/modified policies are passed to the Policy Analyser, which takes their subjects and/or targets and queries the information model for relationships (and constraints on these relationships) for these subjects/targets. Using relationship and constraint information the authors claim that it is possible to assess more precisely those circumstances in which the policy actions should be invoked. The authors propose a Policy Analyser that employs an algorithm that retrieves the relevant relationships and constraints from the information model given an arbitrary policy defined in accordance with that information model.

Once the associated relationships and constraints have been retrieved, the original policy should be refined. As there may be multiple action constraints to be added into the policy, they must first be checked against each other so that the resulting policy action constraints do not logically contradict [Dav06]. An example of this would be if two constraints were added to a policy specifying that the action may only be performed during daytime hours, and another constraint specifying that the action may only be performed during night time hours. This type of rule contradiction will cause the policy not be enforced at anytime, and so the policy can not be refined and is invalid against the referenced information model. According to the authors view, the constraints should be checked against existing policy conditions for completeness.

A more detailed description of this approach can be found in [Dav06]. The refinement process of the work described above is closer to policy conflict analysis. The work reported in this Thesis does not include conflict analysis although some complementary work in this area, related to the work presented in this Thesis can be found n

[Cha05], [Cha06]. We also envisage that policy refinement should be coupled to conflict analysis techniques.

6.6.5 A Classification-Based Approach to Policy Refinement

Classification of statistical data is used to ground the policy refinement problem by Udupi et al [Udu07]. As systems are typically designed based on high-level goals, this approach proposes deriving policy bounds on low-level metrics such that high-level goals are met. Low level metrics are measured at operation time, from which system administrators or experts use domain knowledge to implicitly map bounds of these lower-level metrics such that the high-level performance goals are met. Refinement is carried out using a combination of data classification and test-and-development approaches. A system is deployed within a test-and-development environment and a data set containing values of low-level metrics is collected by placing appropriate workloads on the system. Policy bounds are derived by applying classification techniques on datasets. The classification rules are further refined using statistical distributions to arrive at certain low level rules that are useful for system monitoring and to check the system health when it is deployed and running.

The main steps for policy refinement in this view are: (i) Test and Development Phase, (ii) Classification Phase, (iii) Policy Derivation and Refinement Phase. These steps are summarised in Figure 81 as graphically illustrated in [Udu07].

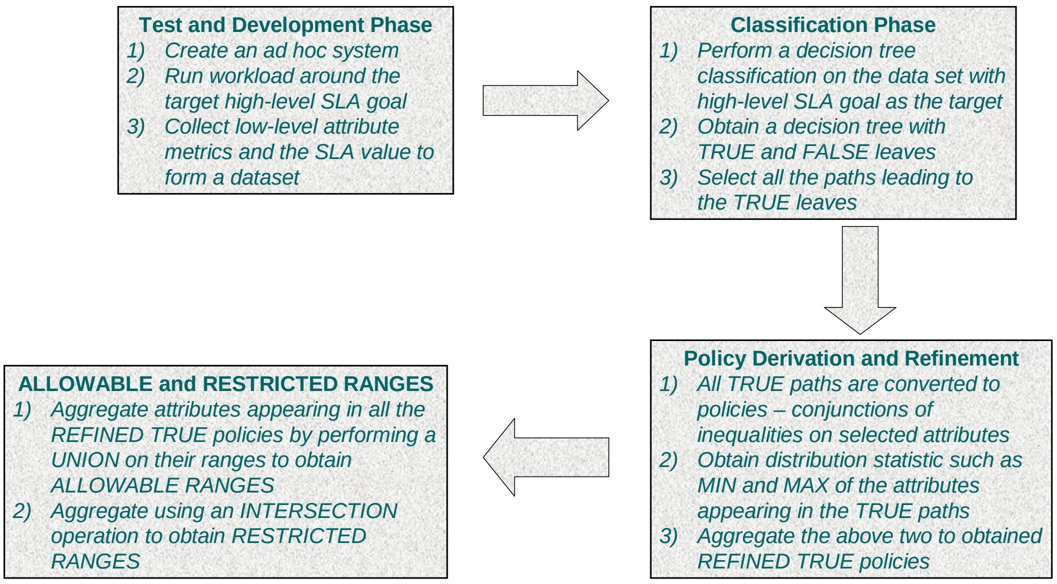

Test and Development Phase
1) Create an ad hoc system
2) Run workload around the target high-level SLA goal
3) Collect low-level attribute metrics and the SLA value to form a dataset
Classification Phase
1) Perform a decision tree classification on the data set with high-level SLA goal as the target
2) Obtain a decision tree with TRUE and FALSE leaves
3) Select all the paths leading to the TRUE leaves
Policy Derivation and Refinement
1) All TRUE paths are converted to policies – conjunctions of inequalities on selected attributes
2) Obtain distribution statistic such as MIN and MAX of the attributes appearing in the TRUE paths
3) Aggregate the above two to obtained REFINED TRUE policies
ALLOWABLE and RESTRICTED RANGES
1) Aggregate attributes appearing in all the REFINED TRUE policies by performing a UNION on their ranges to obtain ALLOWABLE RANGES
2) Aggregate using an INTERSECTION operation to obtain RESTRICTED RANGES

Figure 81. Classification-based policy refinement approach

For the fist step of the refinement process [Udu07], an ad hoc system configuration should be created on the basis of the given high-level SLA goals. A Test and Development environment records values for the low level system attributes by placing workloads on the system that are spread around the ranges of the target workload [Udu07].

Next, the data is pre-processed for classification. This phase involves applying a classification algorithm on the dataset, and deriving the policies that are useful for these purposes. In the dataset collected, for classification purposes the target variable computed should be included from the evaluation of the high-level SLA metrics. The target variable is a Boolean value taking either TRUE or FALSE as the possible values depending on whether the high-level SLA goals are satisfied or not. Classification techniques such as decision tree classification methods should be applied on the dataset for the given target. All the TRUE rules are collected and passed to the next phase.

The Policy derivation and refinement phase derives policies from the output of the classification phase. Considering a decision tree approach for classification, policies should be derived from all the paths leading to TRUE leaves. These TRUE policies are a conjunction of inequalities on various attributes that are picked by the classification phase. A refinement strategy that is applied at this level uses the distribution statistics of the attributes on these TRUE paths. For all attributes in the TRUE tuples (tuples in the dataset that correspond to a TRUE high-level SLA value), distributions statistics such MINIMUM and MAXIMUM values are computed. The inequalities of attributes that appear on the TRUE policies are further refined by appending the MINIMUM and MAXIMUM values giving definite bounds for the attributes, resulting in the required TRUE REFINED policies. This process results in a set of TRUE REFINED policies, which can be used for monitoring system health as for designing subsequent configurations. Further refined categories of policies are provided by aggregating the different policies generated to arrive at rules on individual attributes. Allowable and Restricted ranges are considered for these purposes.

To our understanding, this classification approach could be considered as a template-like approach in the sense that the models and algorithms developed for these purposes are restricted to pre-defined situations i.e. classifications. Another issue that deserves more attention is the relationship between the refined policies and the goals that are linked to the formers. The authors claim to address SLA goal fulfillment and refine policies accordingly. To the best of our knowledge this is a very difficult and challenging issue that implies continuous monitoring and evaluation tasks (feedback mechanisms and analysis). Such mechanisms and their interaction with the refinement approach are unclear although the authors recognise the lack of a verification process for the low-level policies to be consistent.

6.7 Conclusions

In this Chapter we have reviewed related work on policy refinement. We must acknowledge that the refinement area is at its initial stage and explicit efforts to tackle it are rather scarce.

Section 6.2 reviewed the related work concerning goal management. The efforts presented in this Section have been derived from the Requirements Engineering techniques. Initially, by the early nineties, work by Bean et al. [Bea93] introduced the goal-oriented view into network management systems although this work has not been widespread in the research community, most probably due to its lack of continuity in the sense of implementation or dissemination of the concepts developed in such an initiatives. The work presented in this Thesis could be considered as a specialisation of Bean et al.'s initiative, narrowed down to address the refinement process in policy-based network management in a goal oriented fashion.

Regarding goal-oriented efforts, the KAOS approach has apparently been established as one of the default methodologies to ground policy refinement, at least in the work reported by the research community so far and also in our work. The KAOS methodology makes it a powerful support due to the goal-based reasoning techniques that it provides. Moreover, its main limitation is that that it does not provide support to relate the target system behaviour to goal fulfillment finding. Consequently, it does not allow refine enforceable policies from KAOS-like requirements. The methodology has never been demonstrated in management contexts i.e. policy-based systems.

Section 6.3 reviewed the Abduction and Event Calculus (EC)-based approach to goal oriented refinement [Ban03], [Ban04]. The latter approach is committed to overcome the limitations of the KAOS methodology in favour of the policy refinement process. Moreover, it does not make clear how the goal refinement process could be assessed in network management systems. Nevertheless, it is considered a major contribution as it provides powerful analysis techniques aimed at acquiring strategies tailored to fulfilling previously identified goal states. Nevertheless, it is unclear how the linear temporal logic foundations of the KAOS goal graph structures are used in the analysis techniques to drive the acquisition of policies from high-level abstract requirements. To our understanding, it is unclear how the ad-hoc machinery developed to abduce strategies based on EC-based specifications work as the approach does not address explicit temporal execution of goals [Ban05a]. This makes it difficult to think of the scalability for large refinements whilst explicit execution of goals is pivotal in the work presented in this Thesis. Finally, in terms of functionality, [Ban05b] acknowledges that the tool implementation developed on purpose to verify their analysis techniques, is not considered a fully functional system.

Section 6.4 reviewed a pioneer approach to generate policies from process specifications [Dan04]. Contrary to our approach that generates policies from system behaviour, this template-like approach is restricted to pre-established settings as the authors recognise that it relies on a mapping process. More important, it is yet unclear how policies are aligned to administrative objectives.

Section 6.5 reviewed the fist approach explicitly conceived to refine policies in a policy authoring environment. POWER [Cas00] is an environment in which the user is guided to choose policies from pre-designed policy templates tailored for specific use. The major limitation of POWER during the operation of policy systems is the restriction to the administrative parties to choose from pre-defined conditions (policy templates). Additionally, there is no evidence of performance data that allow assess qualitative comparisons between POWER with our refinement approach. Apart from POWER, there is no evidence of any other functional solution to policy refinement hence it is acknowledged as a major contribution to the policy refinement area.

Finally, Section 6.6 reviewed some recent advances in the policy refinement area. With this regard, we could conclude that the policy refinement area is starting to receive an increasing interest from the research community. We could conclude that most of the work is still at the preliminary stage and that substantial efforts are still necessary to overcome all the implications confronting the policy refinement process in different application domains. In addition, template-like approaches are still mostly used in the current approaches and this is one of our major constraints with such solutions. In general, template-like refinements limit the refinement process to pre-established conditions of operation. More dynamic approaches are needed for consideration in future works. Another crucial issue for further considerations is the synergy between refinement and conflict prevention that in principle are the two major issues confronting the utilisation of policy-based management as a widespread technology.

Chapter 7 Summary Conclusions

The following are the summary conclusions of the work presented in this Thesis.

7.1 Review of Contributions

In this Thesis we have addressed one of the most challenging and complex issues of policy-based management, the policy refinement process. The following are the contributions we claim to have made with this regard.

7.1.1 A Policy Refinement Framework and Prototype

Following the idea that goals can represent the achievements of policies at different levels of abstraction [Lam99], [Ban04], we have proposed a holistic goal-oriented refinement framework [Rub05] and prototype [Rub06a],[Rub06b] that provide support to address the policy refinement process.

We have proposed a framework that, having adopted the KAOS goal elaboration method, overcomes its limitations to produce enforceable policies from KAOS goal graph structures. To accomplish this we have made use of inherent logical relationships of goal refinements which to the best of our knowledge, is a genuine advantage of the framework as it enables the automated acquisition of enforceable policies aligned to KAOS-defined goals. This general contribution yields to a number of novel contributions that we claim to have made:

- We have defined appropriate mechanisms aimed at establishing temporal relationships amongst lowest-level goals, and consequently to high-level goal fulfillments. We have exploited the logical foundations of the KAOS methodology, namely the Linear Temporal Logic formalism. These novel mechanisms exploit the potentiality of these foundations which have been unexploited or unconsidered to systematise the policy refinement process.
- We have introduced the use of patterns for behavioural properties specification as the means to represent the fulfillment of lowest-level goals. This novel approach enables the representation of the fulfillment of lowest-level goals, with formal notations and more important, to use searching engines to acquire restricted system behaviour aligned to lowest-level goals fulfillment and consequently with the fulfillment of KAOS high-level goals.
- We have introduced a scalable translation process that produces enforceable policies from restricted system behaviour.

- We have designed and implemented a prototype that enables the execution of the refinement process along the life cycle of a policy-based system; design, implementation, start-up, and operation. To the best of our knowledge, this is the first fully functional approach to goal-oriented policy refinement presented in the literature. With this regard, there is no evidence of any other solution of this kind to provide comparative results of performance.

From the above, we draw the general conclusion that in order to address the policy refinement problems, it is mandatory to be able to represent policies in hierarchical levels in an affordable manner, establish formal relationships amongst them for further analysis tailored at acquiring enforceable policies aligned to KAOS-defined goals. Another general conclusion is that it is feasible and practically achievable to produce enforceable policies aligned to high-level goals in a systematic manner.

7.1.2 A Systematic Approach

One of the most relevant steps of the policy refinement process is that of goal refinement. The KAOS methodology provides application-independent refinement patterns to assess goal refinement. A novel contribution of our work is a systematic approach to drive goal refinement in management contexts [Rub06c]. As far as we know, this is the first work that defines generic guidelines to exploit hierarchical relationships of policy systems in favour of the policy refinement problem. One of the advantages of this approach is that the guidelines can be used in management contexts irrespective of their application domain as long as it is feasible to identify hierarchical relationships within the target system.

With this regard, central to the goal refinement process are the high-level goals that the system designer should define. It is unavoidable and at some point, it would be impossible to systematise the goal refinement problem unless high-level goals are specified in advance. These high-level goals represent the most valuable information to start up the goal refinement process and also represent valuable information that should be used for system design, development, implementation, and operation. Nevertheless, the information handled in this crucial step is anyway used for system design purposes. The incremental effort for a system designer shouldn't be significant.

The result of the systematic approach can be seen as goal-graphs that integrate goal refinement patterns and that mimic the management system composition hierarchy. Moreover, we must acknowledge that it would be very difficult, if not impossible, to assess goal refinement unless the administrative parties are able to establish relationships between the high-level goals and the levels of the system compositional hierarchy. Again, this is an affordable price that should be paid to systematise the goal refinement process.

From all the above, a general conclusion is that as an application-domain independent approach, the goal refinement process requires anyway a deep knowledge of the target system and hence of the application domain. We are still far away from providing a simplistic solution to goal refinement and consequently to policy refinement because most probably, such simplistic solution may not exist. Also, the control of administrative decisions has not been shifted to any generic method but contrary, we have pointed out how the administrators can take advantage of information that system designers handle, in favour of the policy refinement problem. Consequently, each step of the goal refinement process is yet driven by administrative decisions.

7.1.3 Application Scenario

We have executed a complete policy refinement process in the quality of service (QoS) management domain [Rub06c], considering all the requirements, processes, actors and phases involved in such a critical process. To the best of our knowledge, this is the first complete approach to policy refinement carried out in any concrete application domain. The following are some conclusions yielded from this overall achievement.

The execution of the policy refinement process for the QoS Management domain does not represented significant work load as the information used for this achievement has been used for the design, implementation, and the operation of the TEQUILA architecture on which the refinement process has been executed.

A crucial step is the definition of QoS-oriented high-level goals as it has a direct impact on the goal refinement process at design time. This step also pre-establishes the way the administrator consultant will define operative views of QoS Management at runtime. The high-level goals should be representative of the most relevant aspects with which the administrator should control the underlying managed systems in favour of QoS provisioning. The administrative actors should agree on these high-level goals. It would be practically impossible to systematise the refinement process without a complete and rather concrete set of QoS-oriented high-level goals.

Linked to the above remark, refinement-wise we have provided the administrative parties with an affordable systematic approach to define QoS-oriented high-level goals and assess their refinement. At the end of the refinement process we have successfully refined a complete set of policies that enable the functional operation of the TEQUILA system to control QoS provisioning aligned to a QoS-oriented operative view. The refined policies have successfully addressed the synergy of the Traffic Engineering and Service Management functionalities of the TEQUILA approach since these two functions do not work isolated for QoS delivery. For this assessment, it is mandatory to having compiled complete and correct data concerning the hierarchical system composition and the system models for every sub-system involved in the TEQUILA system. Again, this is an affordable price that should be paid in favour of systematising the policy refinement process.

A remarkable issue is that the QoS-oriented refinement process has demanded a deep knowledge of the QoS Management domain although the systematic approach and prototype described in this Thesis, makes policy refinement an affordable and feasible process.

7.2 Discussion and Future work

This section provides a discussion and possible future work on relevant topics addressed in this Thesis.

7.2.1 Framework and Prototype

Regarding the framework and prototype described in this Thesis, a number of issues deserve additional discussion.

Regarding the prescription of the goals addressed in this Thesis, we have restricted ourselves to study Achieve goals [Lam01], this is, we have forced the target managed system to behave to commit with some target behaviour. We have accomplished this by considering unconditional state transitions of the system model in response to events, taking for granted that all states and transitions are permissible in the system model. This approach has enabled us to relate obligation policies with Achieve goals in a systematic manner. Further work could be directed to explore the implications of considering non-permissible states and their potential relationships with Maintain, Avoid and Cease goals [Lam01]. Moreover, we envisage that the principles of the framework and the prototype should be preserved since these goal prescriptions, the framework and prototype itself, all rely on Linear Temporal Logic formalisms. In addition, we envisage that the consideration of these types of goals may enable to carry out analysis for application domains in which authorisation, access control and security issues have special relevance.

One of the principles of the framework and prototype is the acquisition of system behaviour aligned to goal fulfillment, from which enforceable/deployable policies are eventually produced. This acquisition process is carried out through plain state space search. The main drawback of this approach is undoubtedly the state explosion problem for very-large scale System Model specifications. Although we have not experienced this situation in any of the scenarios that we have worked through this Thesis, we recognize that further work should be directed to review the implications of this approach. Most probably, a revisited framework and prototype may include additional System Model specification management procedures and/or revisited heuristic searching mechanisms [Ede01] intended to avoid the state explosion problem for ultra-large scale System Model specifications.

Linked to the acquisition of system behaviour, we have initially considered situations where the searching engine provides only one report of system behaviour fulfilling the high-level goals. Moreover, we acknowledge that a more robust approach should consider handling multiple behaviour traces (if existed also in more robust System Model specifications) which in turn would produce policies applicable to different conditions of system execution. Moreover, we envisage that the principles of the refinement approach should be preserved for each of the multiple restricted-system behaviour traces. The implications of this could be part of future work.

Regarding the administrative tasks considered in the framework and prototype, namely documenting the System Model, and the assessment of goal refinement and goal selection. All these tasks are unavoidably assessed by the administrative parties. Future work could be directed to prevent potential mistakes that the administrators could make during the assessment of the aforementioned tasks.

Another limitation of our current framework and prototype is that we have not considered the implications of run-time changes of the System Model. Future work could be directed to implement tracking mechanisms to police changes on this information and update the refined/deployed policies accordingly.

Finally, we have provided the means to produce enforceable/deployable policies aligned to high-level goals. Moreover, we are still at the very initial stage of the policy refinement problem. Future work should be directed to explore the implications or the relationship between SLA fulfillment and the formulation of high-level goals. This is a critical and challenging issue that may possibly imply to adapt feedback mechanisms to our refinement framework as to involve service management, system performance, goal specialisation, etc, for particular application domains. Consequently, substantial efforts are still needed to solve the policy refinement problem.

7.2.2 Applicability of the Methodological Approach

We claim that a refinement framework could be considered useless unless we provide some guidelines to address the policy refinement problem in management contexts. The methodological approach that we have provided in this Thesis certainly helps in assessing goal refinement, and eventually enables the automated acquisition of enforceable/deployable policies aligned to high-level goals. Nevertheless some issues deserve additional discussion.

The main drawback of the methodological approach is that applying it to specific refinement problem solving requires a deep knowledge of the target system or the application domain. For example, the QoS Management domain in which we have applied our methodology has demanded a deep knowledge of the capabilities, design, implementation details, etc, of the TEQUILA approach. There is no solution that overcomes this drawback, the refinement problem demands a deep knowledge of the application domain.

Our methodological approach prescribes that administrator developer establishes high-level goals from which that the administrator consultant could define QoS-oriented operative views in accordance with the former high-level goals. This issue deserves some additional discussion. For instance, the operative views can be seen as fixed declarations of the administrative view for controlling QoS delivery. It has been out of the Thesis scope to study under which conditions of user demands, these operative views should be changed to allow a better performance of the network controlled with the TEQUILA approach. This is a critical issue that may possible involve statistical analysis, several conditions of user demands, and most probably, different scenarios applied to different network topologies. Moreover, we must acknowledge that our methodological approach enables the consultant to define new operative views and/or modify current ones in runtime.

Linked to the above issue, further work could be directed to integrate statistical and feedback analysis techniques in favour of QoS provisioning, and to enhance the framework itself. One way to accomplish this could be the consideration of evolving goals, i.e. *goal evolution*. We believe that *goal evolution* should be influenced by statistical changes of user load, topology changes, etc. This is a challenging issue, not only for the QoS Management domain, but for the policy refinement problem in general. We strongly believe that our framework could be revisited to address this challenging issue that may probably enable the policy refinement process to shift into a cyclic and continuous process in which contextual information could be used to drive the refinement process.

A general conclusion and future work of this Thesis is that significant efforts are still necessary toward the solution of the refinement problem. We hope that the ideas developed in this Thesis may encourage policy designers and researchers to address the policy refinement problem and/or to revisit the policy refinement framework and methodological approach presented in this Thesis. Policy refinement is still at its initial stage; hence, substantial efforts should be made to solve it.

Appendix A. PROMELA code examples

PROMELA SPECIFICATION EXAMPLE

This part of the Appendix shows the PROMELA specification for the two components, Managed Object 1 and Managed Object 2 exemplified in Section 2.3.3., and graphically shown in Figure 82 for better convenience.

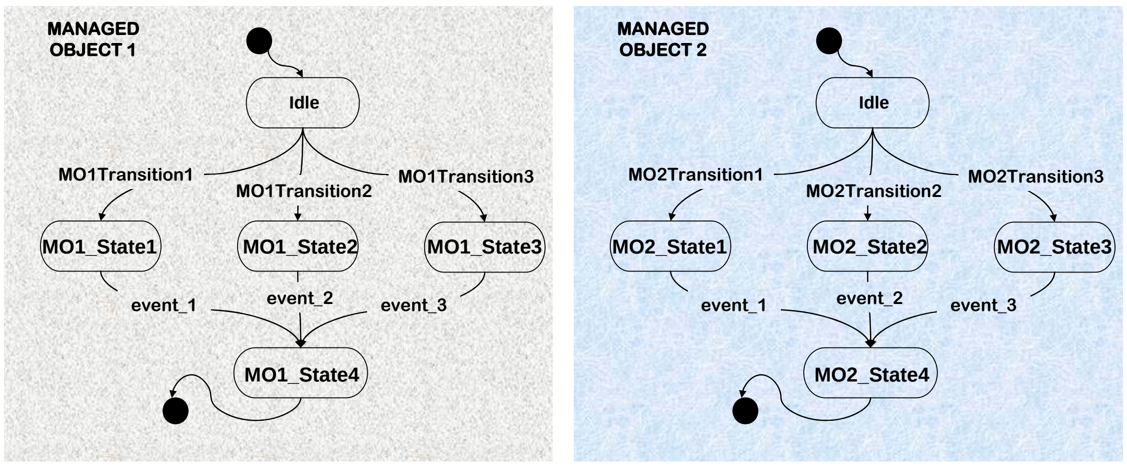


Figure 82. Example of collaborative behaviour

PROLEMA specification:
```
#define top_initial0 1
#define top_initial02Idle 1
#define empty 0
#define completion_queue_size 2
#define Idle 2
#define completion_Idle 0
#define cmplIdle 0
#define MO2_State3 3
#define cmplMO2_State3 0
#define completion_MO2_State3 1
#define MO2_State1 4
#define cmplMO2_State1 0
#define completion_MO2_State1 2
#define MO2_State2 5
#define cmplMO2_State2 0
#define completion_MO2_State2 3
#define number_objects 2
#define queue_size 5
#define send_event_1 0
#define MO2_State4 6
#define cmplMO2_State4 0
#define completion_MO2_State4 4
#define send_event_2 1
#define send_event_3 2
#define MO2_State42top_final0 2
```

```
#define top_final0 7
#define success 0
#define top_initial0_G0 1
#define top_initial02Idle_G1 1
#define Idle_G2 2
#define completion_Idle_G3 3
#define cmplIdle_G4 0
#define MO1_State2 3
#define MO1_State1 4
#define MO1_State3 5
#define MO1_State4 6
#define cmplMO1_State4 0
#define completion_MO1_State4 4
#define MO1_State42top_final0 2
#define top_final0_G5 7
#define obj_MO1 1
#define obj_MO2 2

chan event_queues[number_objects] = [queue_size] of {byte, chan};

pid processIds[number_objects];

proctype ManagedObject2(chan event_queue; byte this; byte initialiser_MO10) {
  chan ack_in = [0] of {bit};
  chan ack_out;
  chan internal_queue = [completion_queue_size] of {byte};

  byte state0;
  byte state0_transition;
  byte MO1;
  bool completed[1];
  byte current_event;

  xr event_queue;
  atomic {
    MO1 = initialiser_MO10;
    state0 = top_initial0;
    state0_transition = top_initial02Idle;
    goto transitionFiring
  };
  main:
  current_event = empty;
  if
  :: internal_queue?[current_event] ->
     internal_queue?current_event
  :: else ->
     if
     :: event_queue?[current_event, ack_out] ->
        event_queue?current_event,ack_out
     :: else ->
        event_queue?current_event,ack_out
     fi
  fi;
  atomic {
    if
    :: state0 == Idle && current_event == completion_Idle && completed[cmplIdle] ==
true && true && true ->
       completed[cmplIdle] = false;
       state0 = empty;
       state0 = MO2_State3;
       completed[cmplMO2_State3] = true;
       internal_queue!completion_MO2_State3;
       goto main
    :: state0 == Idle && current_event == completion_Idle && completed[cmplIdle] ==
true && true && true ->
       completed[cmplIdle] = false;
       state0 = empty;
       state0 = MO2_State1;
       completed[cmplMO2_State1] = true;
       internal_queue!completion_MO2_State1;
       goto main
    :: state0 == Idle && current_event == completion_Idle && completed[cmplIdle] ==
true && true && true ->
       completed[cmplIdle] = false;
       state0 = empty;
       state0 = MO2_State2;
       completed[cmplMO2_State2] = true;
       internal_queue!completion_MO2_State2;
       goto main
    :: state0 == MO2_State1 && current_event == completion_MO2_State1 && com-
pleted[cmplMO2_State1] == true && true && true ->
       completed[cmplMO2_State1] = false;
       state0 = empty;
       event_queues[MO1-1]!send_event_1,ack_in;
       state0 = MO2_State4;
       completed[cmplMO2_State4] = true;
       internal_queue!completion_MO2_State4;
       goto main
    :: state0 == MO2_State2 && current_event == completion_MO2_State2 && com-
pleted[cmplMO2_State2] == true && true && true ->
       completed[cmplMO2_State2] = false;
       state0 = empty;
       event_queues[MO1-1]!send_event_2,ack_in;
       state0 = MO2_State4;
       completed[cmplMO2_State4] = true;
       internal_queue!completion_MO2_State4;
       goto main
    :: state0 == MO2_State3 && current_event == completion_MO2_State3 && com-
pleted[cmplMO2_State3] == true && true && true ->
       completed[cmplMO2_State3] = false;
       state0 = empty;
       event_queues[MO1-1]!send_event_3,ack_in;
       state0 = MO2_State4;
       completed[cmplMO2_State4] = true;
       internal_queue!completion_MO2_State4;
       goto main
    :: state0 == MO2_State4 && current_event == completion_MO2_State4 && com-
pleted[cmplMO2_State4] == true && true && true ->
       state0_transition = MO2_State42top_final0;
       goto top_label
    :: else
    fi;
    top_label:
    skip;
    transitionFiring:
    do
    :: state0_transition == top_initial02Idle ->
       state0_transition = empty;
       state0 = empty;
       state0 = Idle;
       completed[cmplIdle] = true;
       internal_queue!completion_Idle
    :: state0_transition == MO2_State42top_final0 ->
       state0_transition = empty;
       completed[cmplMO2_State4] = false;
       state0 = empty;
       state0 = top_final0
    :: else ->
       break
    od;
    if
    :: state0 != top_final0 ->
       goto main
    :: else ->
       goto end_machine
    fi
  };
  end_machine:
  success
}

proctype ManagedObject1(chan event_queue; byte this; byte initialiser_MO20) {
  chan ack_out;
  chan internal_queue = [completion_queue_size] of {byte};

  byte state0;
  byte state0_transition;
  bool completed[1];
  byte current_event;
  byte MO2;

  xr event_queue;
  atomic {
    MO2 = initialiser_MO20;
    state0 = top_initial0_G0;
    state0_transition = top_initial02Idle_G1;
    goto transitionFiring
  };
  main:
  current_event = empty;
  if
  :: internal_queue?[current_event] ->
     internal_queue?current_event
  :: else ->
     if
     :: event_queue?[current_event, ack_out] ->
        event_queue?current_event,ack_out
     :: else ->
        event_queue?current_event,ack_out
     fi
  fi;
  atomic {
    if
    :: state0 == Idle_G2 && current_event == completion_Idle_G3 && com-
pleted[cmplIdle_G4] == true && true && true ->
       completed[cmplIdle_G4] = false;
       state0 = empty;
       state0 = MO1_State2;
       goto main
    :: state0 == Idle_G2 && current_event == completion_Idle_G3 && com-
pleted[cmplIdle_G4] == true && true && true ->
       completed[cmplIdle_G4] = false;
       state0 = empty;
       state0 = MO1_State1;
       goto main
    :: state0 == Idle_G2 && current_event == completion_Idle_G3 && com-
pleted[cmplIdle_G4] == true && true && true ->
       completed[cmplIdle_G4] = false;
       state0 = empty;
```

```
        state0 = MO1_State3;
        goto main
     :: state0 == MO1_State1 && current_event == send_event_1 && true && true ->
        state0 = empty;
        state0 = MO1_State4;
        completed[cmplMO1_State4] = true;
        internal_queue!completion_MO1_State4;
        goto main
     :: state0 == MO1_State2 && current_event == send_event_2 && true && true ->
        state0 = empty;
        state0 = MO1_State4;
        completed[cmplMO1_State4] = true;
        internal_queue!completion_MO1_State4;
        goto main
     :: state0 == MO1_State3 && current_event == send_event_3 && true && true ->
        state0 = empty;
        state0 = MO1_State4;
        completed[cmplMO1_State4] = true;
        internal_queue!completion_MO1_State4;
        goto main
     :: state0 == MO1_State4 && current_event == completion_MO1_State4 && com-
pleted[cmplMO1_State4] == true && true && true ->
        state0_transition = MO1_State42top_final0;
        goto top_label
     :: else
     fi;
     top_label:
     skip;
     transitionFiring:
     do
     :: state0_transition == top_initial02Idle_G1 ->
        state0_transition = empty;
        state0 = empty;
        state0 = Idle_G2;
        completed[cmplIdle_G4] = true;
        internal_queue!completion_Idle_G3
     :: state0_transition == MO1_State42top_final0 ->
        state0_transition = empty;
        completed[cmplMO1_State4] = false;
        state0 = empty;
        state0 = top_final0_G5
     :: else ->
        break
     od;
     if
     :: state0 != top_final0_G5 ->
        goto main
     :: else ->
        goto end_machine
     fi
  };
  end_machine:
  success
}

init {
  atomic {
    processIds[obj_MO1-1] = run ManagedObject1(event_queues[obj_MO1-1], obj_MO1,
obj_MO2);
    processIds[obj_MO2-1] = run ManagedObject2(event_queues[obj_MO2-1], obj_MO2,
obj_MO1)
  }
}
```

EXECUTION REPORT PROVIDED BY THE PROMELA INTERPRETER

This part of the Appendix shows the complete execution report reported by the PROMELA Interpreter SPIN for the example provided in Section 2.3.3., and graphically shown in Figure 83 for better convenience.

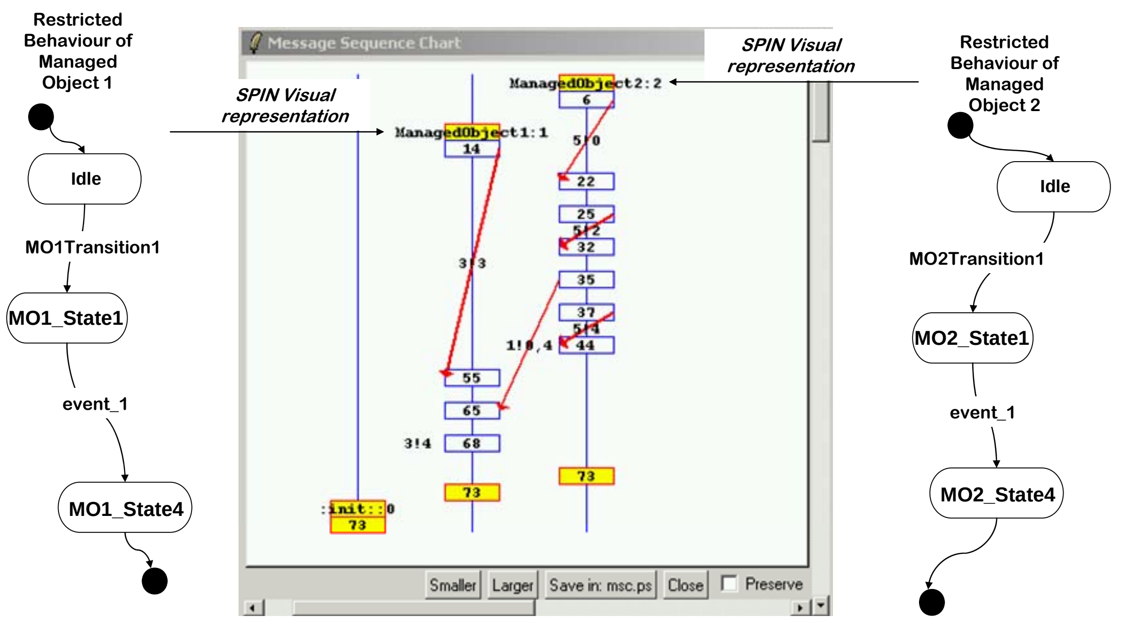


Figure 83. Visual representation of execution report

SPIN execution report:

```
Starting :init: with pid 0
spin: warning, "pan_in", global, 'byte processIds' variable is never used
spin: warning, "pan_in", proctype ManagedObject2, 'byte this' variable is never used
spin: warning, "pan_in", proctype ManagedObject1, 'byte this' variable is never used
spin: warning, "pan_in", proctype ManagedObject1, 'byte MO2' variable is never used
spin: couldn't find claim (ignored)
Starting ManagedObject1 with pid 2
  2:proc  0 (:init:) line 261 "pan_in" (state 1)   [processIds[(1-1)] = run
ManagedObject1(event_queues[(1-1)],1,2)]   <merge 4 now @2>
        queue 1 (event_queues[0]):
        queue 2 (event_queues[1]):
        processIds[0] = 2
        processIds[1] = 0

Starting ManagedObject2 with pid 3
  2:proc  0 (:init:) line 262 "pan_in" (state 2)   [processIds[(2-1)] = run
ManagedObject2(event_queues[(2-1)],2,1)]   <merge 4 now @4>
        queue 1 (event_queues[0]):
        queue 2 (event_queues[1]):
        processIds[0] = 2
        processIds[1] = 3
4:  proc  2 (ManagedObject2) line  61 "pan_in" (state 1)   [MO1 = initial-
iser_MO10]  <merge 83 now @2>
4:  proc  2 (ManagedObject2) line  62 "pan_in" (state 2)   [state0 = 1]   <merge
```

```
83 now @3>
4:  proc  2 (ManagedObject2) line  63 "pan_in" (state 3)  [state0_transition = 1]
    <merge 83 now @83>
5:  proc  2 (ManagedObject2) line 135 "pan_in" (state 70)
    [((state0_transition==1))] <merge 0 now @71>
5:  proc  2 (ManagedObject2) line 136 "pan_in" (state 71) [state0_transition = 0]
    <merge 75 now @72>
5:  proc  2 (ManagedObject2) line 137 "pan_in" (state 72)  [state0 = 0]   <merge
75 now @73>
5:  proc  2 (ManagedObject2) line 138 "pan_in" (state 73)  [state0 = 2]   <merge
75 now @74>
5:  proc  2 (ManagedObject2) line 139 "pan_in" (state 74)  [completed = 1] <merge
75 now @75>
6:  proc  2 (ManagedObject2) line 140 "pan_in" (state -)   [values: 5!0]
6:  proc  2 (ManagedObject2) line 140 "pan_in" (state 75)  [internal_queue!0]
7:  proc  2 (ManagedObject2) line 146 "pan_in" (state 81)  [else]
8:  proc  2 (ManagedObject2) line 150 "pan_in" (state 86)  [((state0!=7))] <merge
0 now @6>
10: proc  2 (ManagedObject2) line  67 "pan_in" (state 6)   [current_event = 0]
12: proc  1 (ManagedObject1) line 172 "pan_in" (state 1)   [MO2 = initial-
iser_MO20]  <merge 71 now @2>
12: proc  1 (ManagedObject1) line 173 "pan_in" (state 2)   [state0 = 1]   <merge
71 now @3>
12: proc  1 (ManagedObject1) line 174 "pan_in" (state 3)  [state0_transition = 1]
    <merge 71 now @71>
13: proc  1 (ManagedObject1) line 234 "pan_in" (state 58)
    [((state0_transition==1))] <merge 0 now @59>
13: proc  1 (ManagedObject1) line 235 "pan_in" (state 59) [state0_transition = 0]
    <merge 63 now @60>
13: proc  1 (ManagedObject1) line 236 "pan_in" (state 60)  [state0 = 0]   <merge
63 now @61>
13: proc  1 (ManagedObject1) line 237 "pan_in" (state 61)  [state0 = 2]   <merge
63 now @62>
13: proc  1 (ManagedObject1) line 238 "pan_in" (state 62)  [completed = 1] <merge
63 now @63>
14: proc  1 (ManagedObject1) line 239 "pan_in" (state -)   [values: 3!3]
14: proc  1 (ManagedObject1) line 239 "pan_in" (state 63)  [internal_queue!3]
15: proc  1 (ManagedObject1) line 245 "pan_in" (state 69)  [else]
16: proc  1 (ManagedObject1) line 249 "pan_in" (state 74)  [((state0!=7))] <merge
0 now @6>
18: proc  1 (ManagedObject1) line 178 "pan_in" (state 6)   [current_event = 0]
20: proc  2 (ManagedObject2) line  69 "pan_in" (state -)   [5?0]
20: proc  2 (ManagedObject2) line  69 "pan_in" (state 7)   [(inter-
nal_queue?[current_event])]
22: proc  2 (ManagedObject2) line  70 "pan_in" (state -)   [values: 5?0]
22: proc  2 (ManagedObject2) line  70 "pan_in" (state 8)   [inter-
nal_queue?current_event]
24: proc  2 (ManagedObject2) line  88 "pan_in" (state 25)
    [((((((state0==2)&&(current_event==0))&&(completed==1))&&1)&&1))] <merge 0
now @26>
24: proc  2 (ManagedObject2) line  89 "pan_in" (state 26)  [completed = 0] <merge
30 now @27>
24: proc  2 (ManagedObject2) line  90 "pan_in" (state 27)  [state0 = 0]   <merge
30 now @28>
24: proc  2 (ManagedObject2) line  91 "pan_in" (state 28)  [state0 = 4]   <merge
30 now @29>
24: proc  2 (ManagedObject2) line  92 "pan_in" (state 29)  [completed = 1] <merge
30 now @30>
25: proc  2 (ManagedObject2) line  93 "pan_in" (state -)   [values: 5!2]
25: proc  2 (ManagedObject2) line  93 "pan_in" (state 30)  [internal_queue!2]
26: proc  2 (ManagedObject2) line  94 "pan_in" (state 31)  [goto main]
28: proc  2 (ManagedObject2) line  67 "pan_in" (state 6)   [current_event = 0]
30: proc  2 (ManagedObject2) line  69 "pan_in" (state -)   [5?2]
```

```
30: proc  2 (ManagedObject2) line  69 "pan_in" (state 7)   [(inter-
nal_queue?[current_event])]
32: proc  2 (ManagedObject2) line  70 "pan_in" (state -)   [values: 5?2]
32: proc  2 (ManagedObject2) line  70 "pan_in" (state 8)   [inter-
nal_queue?current_event]
34: proc  2 (ManagedObject2) line 102 "pan_in" (state 39)
    [((((((state0==4)&&(current_event==2))&&(completed==1))&&1)&&1))] <merge 0
now @40>
34: proc  2 (ManagedObject2) line 103 "pan_in" (state 40)  [completed = 0] <merge
42 now @41>
34: proc  2 (ManagedObject2) line 104 "pan_in" (state 41)  [state0 = 0]    <merge
42 now @42>
35: proc  2 (ManagedObject2) line 105 "pan_in" (state -)   [values: 1!0,4]
35: proc  2 (ManagedObject2) line 105 "pan_in" (state 42)
    [event_queues[(MO1-1)]!0,ack_in]
        queue 1 (event_queues[0]): [0,4]
36: proc  2 (ManagedObject2) line 106 "pan_in" (state 43)  [state0 = 6]    <merge
45 now @44>
36: proc  2 (ManagedObject2) line 107 "pan_in" (state 44)  [completed = 1] <merge
45 now @45>
37: proc  2 (ManagedObject2) line 108 "pan_in" (state -)   [values: 5!4]
37: proc  2 (ManagedObject2) line 108 "pan_in" (state 45)  [internal_queue!4]
38: proc  2 (ManagedObject2) line 109 "pan_in" (state 46)  [goto main]
40: proc  2 (ManagedObject2) line  67 "pan_in" (state 6)   [current_event = 0]
42: proc  2 (ManagedObject2) line  69 "pan_in" (state -)   [5?4]
42: proc  2 (ManagedObject2) line  69 "pan_in" (state 7)   [(inter-
nal_queue?[current_event])]
44: proc  2 (ManagedObject2) line  70 "pan_in" (state -)   [values: 5?4]
44: proc  2 (ManagedObject2) line  70 "pan_in" (state 8)   [inter-
nal_queue?current_event]
46: proc  2 (ManagedObject2) line 126 "pan_in" (state 63)
    [((((((state0==6)&&(current_event==4))&&(completed==1))&&1)&&1))] <merge 0
now @64>
46: proc  2 (ManagedObject2) line 127 "pan_in" (state 64)  [state0_transition =
2]
47: proc  2 (ManagedObject2) line 132 "pan_in" (state 69)  [(1)]
48: proc  2 (ManagedObject2) line 141 "pan_in" (state 76)
    [((state0_transition==2))] <merge 0 now @77>

48: proc  2 (ManagedObject2) line 142 "pan_in" (state 77) [state0_transition = 0]
    <merge 83 now @78>

48: proc  2 (ManagedObject2) line 143 "pan_in" (state 78)  [completed = 0] <merge
83 now @79>

48: proc  2 (ManagedObject2) line 144 "pan_in" (state 79)  [state0 = 0]    <merge
83 now @80>

48: proc  2 (ManagedObject2) line 145 "pan_in" (state 80)  [state0 = 7]    <merge
83 now @83>

49: proc  2 (ManagedObject2) line 146 "pan_in" (state 81)  [else]

50: proc  2 (ManagedObject2) line 152 "pan_in" (state 88)  [else]

51: proc  2 (ManagedObject2) line 153 "pan_in" (state 89)  [goto end_machine]

53: proc  1 (ManagedObject1) line 180 "pan_in" (state -)   [3?3]
53: proc  1 (ManagedObject1) line 180 "pan_in" (state 7)   [(inter-
nal_queue?[current_event])]

55: proc  1 (ManagedObject1) line 181 "pan_in" (state -)   [values: 3?3]
55: proc  1 (ManagedObject1) line 181 "pan_in" (state 8)   [inter-
nal_queue?current_event]

57: proc  1 (ManagedObject1) line 197 "pan_in" (state 23)
    [((((((state0==2)&&(current_event==3))&&(completed==1))&&1)&&1))] <merge 0
now @24>
57: proc  1 (ManagedObject1) line 198 "pan_in" (state 24)  [completed = 0] <merge
6 now @25>
57: proc  1 (ManagedObject1) line 199 "pan_in" (state 25)  [state0 = 0]   <merge
6 now @26>
57: proc  1 (ManagedObject1) line 200 "pan_in" (state 26)  [state0 = 4]   <merge
6 now @6>
59: proc  1 (ManagedObject1) line 178 "pan_in" (state 6)   [current_event = 0]

61: proc  1 (ManagedObject1) line 182 "pan_in" (state 9)   [else]
63: proc  1 (ManagedObject1) line 184 "pan_in" (state -)   [1?0,4]
63: proc  1 (ManagedObject1) line 184 "pan_in" (state 10)
    [(event_queue?[current_event,ack_out])]
        queue 1 (event_queues[0]): [0,4]
65: proc  1 (ManagedObject1) line 185 "pan_in" (state -)   [values: 1?0,4]
65: proc  1 (ManagedObject1) line 185 "pan_in" (state 11)
    [event_queue?current_event,ack_out]
        queue 1 (event_queues[0]):
67: proc  1 (ManagedObject1) line 207 "pan_in" (state 33)
    [(((((state0==4)&&(current_event==0))&&1)&&1))]   <merge 0 now @34>
67: proc  1 (ManagedObject1) line 208 "pan_in" (state 34)  [state0 = 0]   <merge
37 now @35>
67: proc  1 (ManagedObject1) line 209 "pan_in" (state 35)  [state0 = 6]   <merge
37 now @36>
67: proc  1 (ManagedObject1) line 210 "pan_in" (state 36)  [completed = 1] <merge
37 now @37>
68: proc  1 (ManagedObject1) line 211 "pan_in" (state -)   [values: 3!4]
68: proc  1 (ManagedObject1) line 211 "pan_in" (state 37)  [internal_queue!4]
69: proc  1 (ManagedObject1) line 212 "pan_in" (state 38)  [goto main]
71: proc  1 (ManagedObject1) line 178 "pan_in" (state 6)   [current_event = 0]

spin: trail ends after 73 steps
#processes: 3
        queue 1 (event_queues[0]):
        queue 2 (event_queues[1]):
        processIds[0] = 2
        processIds[1] = 3
73: proc  2 (ManagedObject2) line 157 "pan_in" (state 93)
73: proc  1 (ManagedObject1) line 179 "pan_in" (state 16)
73: proc  0 (:init:) line 264 "pan_in" (state 4)
3 processes created
Exit-Status 0
```

Appendix B. DSC Platform Highlights

This Appendix is aimed at providing the highlights of the DSC platform [Mee00] on which the refinement prototype was implemented. DSC stands for Distributed Software Component and is a general purpose framework to build distributed software based on a component design paradigm. We present a simple example to implement a very simple client server component application using DSC. The example has been chosen to be as simple as possible since the methodology is similar for developing large and complex component systems.

In this example a client component will send a text message to a server component, which will print the message “Hello World” to the console. Although this example is about as simple as it can get, it will be a fully distribution transparent application where the client and server part may be on different machines, on the same machine in separate containers, or on the same machine in a shared container. The component implementation and development process is independent of the chosen run time configuration. The overall methodology consists of the following steps:

- **System design.**
- **Component implementation.**
- **Component utilisation.**

SYSTEM DESIGN

The design steps are as follows:

1. Decomposition of the application in components and describe the function of each component
2. Define the interfaces and interface operations that each component will provide. The interfaces may consist of (or inherit from) existing IDL interface, or may require new IDL specifications. IDL stands for Interface Definition Language.
3. Define the component specification for each component.

Design step 1: Decomposition

For the Hello World example this results in the design as shown in Figure 84. The server component is named `HelloWorld`, the client component is named `HelloActor`. They both provide a component interface with the same name. This interface will be generated automatically and will provide standard operations. The `HelloWorld` component offers one additional facet named `hello` which provides one operation named `print`. The functionality of the `HelloWorld` component is to print any message it receives to the console. The functionality of the `HelloActor` component is to send a message to the `HelloWorld` component.

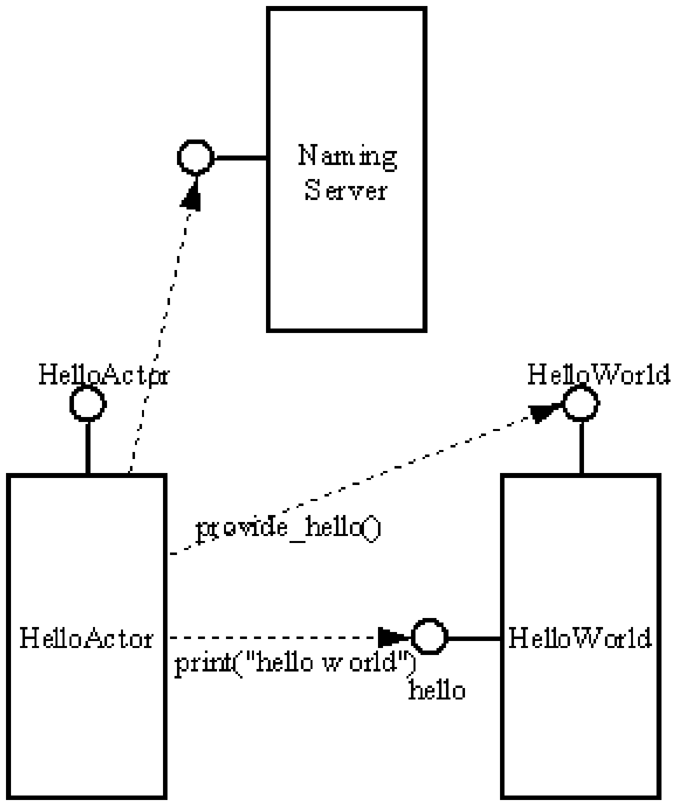


Figure 84. Hello World design with DSC framework

Design step 2: IDL Specification

The next design step is to write the interface specifications in IDL. The IDL language allows related interfaces to be grouped in modules. For the HelloWorld example the interface is placed in module "hello" as we show bellow:

```
#ifndef _EXAMPLES_HELLO_IDL_
#define _EXAMPLES_HELLO_IDL_
/**
 * Thisis the component specification for a simple component.
 * @author Harold Batteram
*/
module hello {
    /**
     * The i_Hello interface only has the print operation.
     */
    interface i_Hello {
        void print(in string message);
    };
};
#endif
```

The IDL file declares one interface of type `hello::i_Hello` which has one operation with signature `void print(in string message)`.

Design step 3: CSL Specification

Each component must have a component specification written in the Component Specification Language (CSL). The CSL file will be used as input for the DSC toolkit to generate the necessary source files of the component. The CSL file declares the type of a component and lists the facets offered by the component. The CSL file for the HelloWorld component shown below declares the HelloWorld component.

```
#include <examples/Hello.idl>
/**
 * This is the component specification for the HelloWorld component.
*/
component examples::Hello::HelloWorld {
    /**
     * There is just one facet.
     */
    interface hello::i_Hello hello {}

}
```

The CSL for the HelloActor is even simpler since this component does not offer any facets itself but only uses the facets of the HelloWorld component. To make this dependency explicit, the `requires` clause is used in the CSL file, shown below.

```
#include <examples/Hello.idl>
/**
 * This is the component specification for the HelloActor component.
*/
component examples::Hello::HelloActor {
    /**
     * The HelloActor component needs the HelloWorld component to obtain
     * a reference to the hello facet.
     *
     * The use clause is used to declare dependencies on other components.
     */
    uses {
        examples::Hello::HelloWorld
    }
}
```

Both CSL files start by including the Hello.idl file.

COMPONENT IMPLEMENTATION

Implementation step 1: Skeleton generation

The DSC toolkit can be used to generate implementation skeletons. The implementation skeletons are generated as templates that can be copied for further development. The DSC toolkit generates the component skeletons based on information obtained from parsing the CSL and IDL files. For each component a component implementation skeleton and facet implementation skeletons for each supported skeletons are generated. These implementation skeletons need to be completed by the component developer. To complete the implementation, code for initialisations, lifecycle management and operation bodies need to be filled in. All generated skeleton coding have default implementations so that the component developer only needs to focus on the essentials.

Implementation step 2: Skeleton completion

Once the implementation skeletons have been generated we can modify the contents. For example, for the HelloActor component we only have to modify the implementation file of the HelloActor component, namely the HelloActorImpl.java. In the component implementation we have to accomplish three things:

1. Obtain a reference to the HelloWorld server component.
2. Use this reference to obtain a reference to the hello facet of the HelloWorld component.
3. Invoke the print operation on the hello facet.

The pattern to obtain component references, facet references and invoking operations is very common and is a key pattern in the DSC Framework. The implementation of this pattern is shown in the content of the HelloActor.java file shown below.

```
package examples.Hello.HelloActorPackage;
public class HelloActorImpl extends examples.Hello.HelloActorPackage.HelloActorGen
{
private dscitfs.hello.i_Hello helloFacet = null;
/**
* The following methods will be called by the component framework.
* They may be overloaded and filled with your own lifecycle code.
*/
// Called once when the component is constructed.
public void init() {
try {
// First obtain a reference to the HelloWorld component.
examples.Hello.HelloWorld helloWorldComponent =
examples.Hello.HelloWorldHelper.narrow(m_container.findComponent(
"examples.context/Hello.context/HelloWorld.component"));
// We can now ask the HelloWorld component for a reference to the facet named hello.
helloFacet = helloWorldComponent.provide_hello();
} catch (dsc.UnknownItem e) {
trace("init", e);
}
};
// Called whenever the component is started.
public void start() {
helloFacet.print("Hello World!");
}
//Called whenever the component is stopped.
public void stop() {
}
//Called once when the component is destroyed.
public void shutdown() {
}
/**
* Constructor.
*/
public HelloActorImpl(String name, org.omg.PortableServer.POA poa) {
super(name, poa);}
}
```

A private variable `helloFacet` is declared which will be used to hold the hello facet reference. The generated implementation skeleton contains four life cycle method: `init()`, `start()` `stop()` and `shutdown()`. `Init()` is called once during initialisation of the component and is commonly used to initialise global member variables that will be used during component operations. The `start()` method is called by the framework after initialisation of the component and its facets. Components can be stopped to suspend their operation and restarted by the framework. When a component is destroyed by the framework the stop method will be called for the component and all its facets followed by the shutdown method. In this example we will implement the `init()` method to initialize the `helloFacet` variable and implement the `start()` method to call the `print` operation on the `helloFacet`.

The container implementation provides the `findComponent()` method which can be used to locate components that have been registered by the naming server using their component name. Component and facet references are ordinary CORBA object references. The reference returned by `findComponent()` must be converted to a reference to the CORBA stub object using the standard CORBA narrow method provided by the Helper class for the specific CORBA object type.

Once we have the reference to component `HelloWorld` we can request the facet reference using the standard `provide_hello()` operation. When the facet reference is assigned to variable `helloFacet`, the initialization of the component is finished and we are ready to use the facet. We implement the start method in which we call the `print()` operation on the `helloFacet`. This invocation will be sent to the server component through the CORBA ORB wherever the server is located. This completes the implementation of the client part.

In order to complete the implementation of the HelloWorld server component we only have to make a few modifications in the facet implementation for the hello facet. This implementation is generated in the file produced for such a purpose,namely the helloImpl.java file. The most relevant is the operation delegated for the print operation which must be modified as follows.

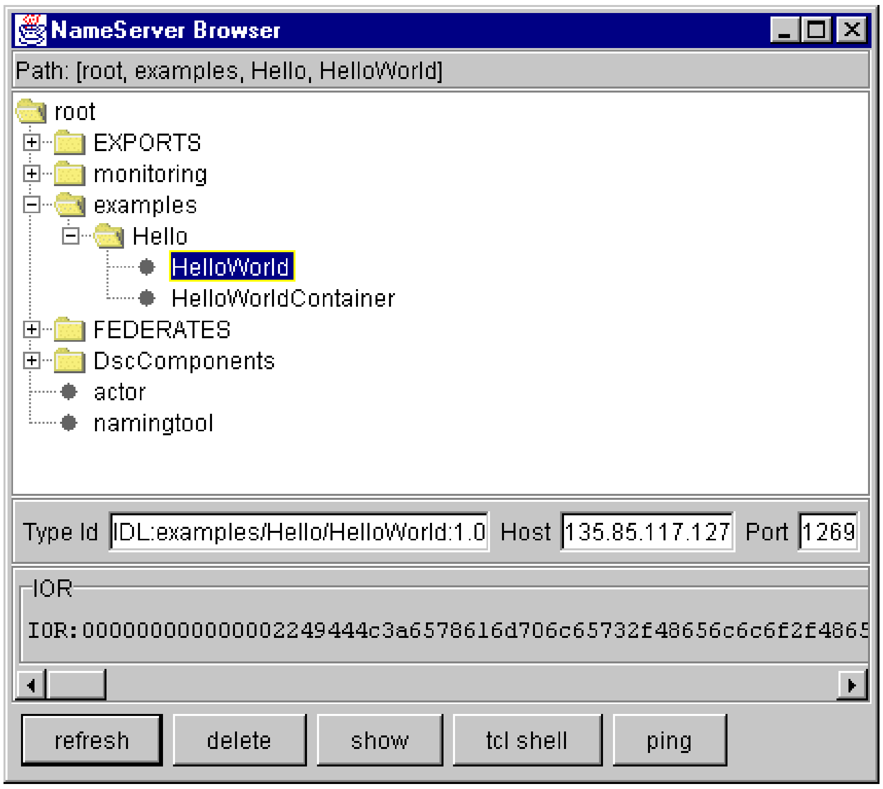
FROM:
```
public void print(java.lang.String message) {
// TODO: replace with your method code
reactor.print(message);
}
```

TO:
```
public void print(java.lang.String message) {
System.out.println(message);
}
```

This way, the two components are ready for compilation and use.

COMPONENT UTILISATION

In order to utilise the HelloWorld component we need to invoke the print method on the hello facet. The DSC framework enables the utilisation of components as stand-alone for development purposes by the use of an actor tool. The actor component is a general purpose testing tool which can be used to invoke operation on selected component facets and observe the result of the invocation. To test the HelloWorld component we must first start the component and then start the actor. This will show a window with the contents of the naming server. During its initialisation the HelloWorld component will have registered its component reference with the naming server. We can find this reference by browsing the actor window and selecting the HelloWorld component as shown in Figure 85.


NameServer Browser

Path: [root, examples, Hello, HelloWorld]

```
root
  EXPORTS
  monitoring
  examples
    Hello
      HelloWorld
      HelloWorldContainer
  FEDERATES
  DscComponents
  actor
  namingtool
```

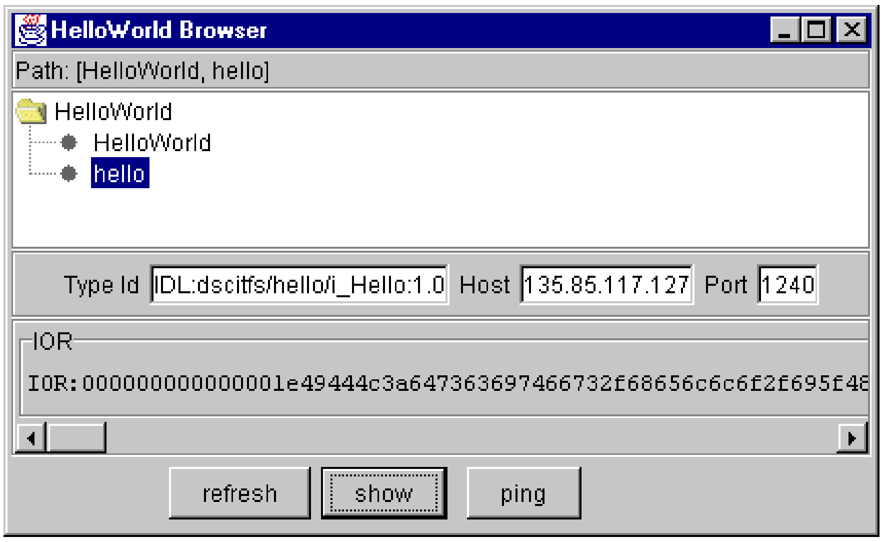
Type Id IDL:examples/Hello/HelloWorld:1.0 Host 135.85.117.127 Port 1269

IOR

IOR:000000000000002249444c3a6578616d706c65732f48656c6c6f2f4865

Figure 85. Actor object browser

When the HelloWorld component is selected its contents can be seen by pressing the show button. This will present a window showing all the facets that the component has as we show in Figure 86. A target facet can now be selected and its operations can be seen by pressing show again. This will now present a window showing the selected operation and allows its parameter values to be filled in a similar way to the reactor window discussed earlier. When the print method is selected on the hello facet we can fill in the message parameter and invoke the operation as shown in Figure 87. The HelloWorld server will receive this invocation and print the contents of the message to the console. We now show that the server part works too.


HelloWorld Browser

Path: [HelloWorld, hello]

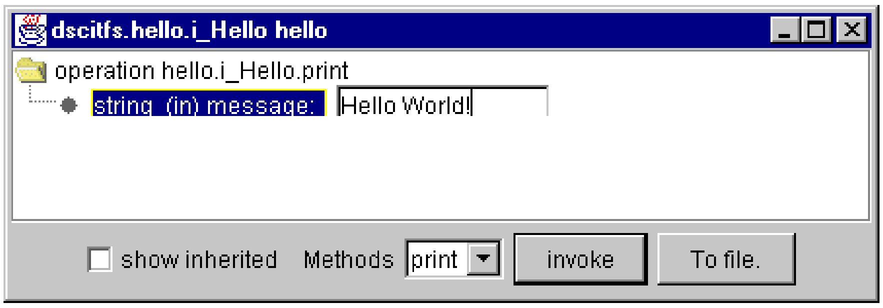
HelloWorld
- HelloWorld
- hello

Type Id IDL:dscitfs/hello/i_Hello:1.0 Host 135.85.117.127 Port 1240

IOR

```
IOR:000000000000001e49444c3a647363697466732f68656c6c6f2f695f48
```

Figure 86. Component browser


dscitfs.hello.i_Hello hello

operation hello.i_Hello.print
- string (in) message: Hello World!

show inherited Methods print

Figure 87. Operation browser

In order to test both components together, we first start the HelloWorld server and then the HelloActor client component. The server should print the "Hello world!" message again on the console. We must acknowledge that the principles of this simple example are preserved for more complex distribution component scenarios.

Appendix C. Prototype Detailed Design

The next sub-Sections provide the implementation issues of the DSC components that integrate our prototype.

IMPLEMENTATION OF THE GOAL MANAGEMENT COMPONENTS

Objectiver Interoperability Issues

The Objectiver package and the Goal Manger are used to achieve the Goal Management tasks. The goal graph structures and the data handled by Objectiver represent the source of information for the analysis intended to accomplish with the administrative criteria defined through this facility. In practice, it will be necessary to exchange information between the Objectiver package and the Goal Manager. To address this interoperability issue we have made use of the DSC framework facilities.

The DSC development framework provides means to specify data structures that help in formalising object-oriented interoperability between distributed components. In our solution we have used the MTNM (Multi-Technology Network Management) globaldefs::NamingAttributes_T structure [Mtnm] as the base of our naming schemes throughout our design and implementation. This structure is used to define identifiers for managed entities that are not instantiated as first class CORBA objects and thus do not have object identifiers. Regarding the Goal Management tasks, the above facilities have been used to define an object-oriented data structure that helps in representing the attributes of every goal within our solution, namely the `Goal` entity.

Every `Goal` entity of our above structure is integrated by five NamingAttributes_T attributes. These attributes are the `goalName, refinedSons, tempRelationship, goalDetails`, and `temporalPrescription`. Each `Goal` entity has a unique `goalName` and may have many `refinedSons`. At the same time, each `Goal` has a specific `temporalPrescription` that can be one of the four available types: `Achieve, Cease, Maintain` and `Avoid`. Each goal also has specific `goalDetails` as attributes. Available to these, for instance are the type `GoalHierarchy` that provides the level of a given goal within the hierarchy of a goal graph (i.e. between highest and lowest levels). Other type for this is the `AdminOption` that identifies the action selected by the consultant (e.g. “refine” and “selected” are the most relevant type instances). As in plan-based techniques [Lam01], lowest-level goals are identified by state predicates. For the later purposes, the `BehaviorSpec` and `SpecStateId` types are used to formalise these data. These latter are only used with lowest-level goals. Finally the `temporalRelationship` attribute can be either `Milestone, MultipleMilestone` or `CaseDriven` types, and are associated to the refinement pattern applied to parent goals.

The above five NamingAttributes_T attributes are integrated into a **`GoalMetaData_T`** object. With regard to sets of `Goal` entities like Goal Selections, goal graph structures or goal sub-graphs, these are aggregated into **`GoalMetaDataList_T`** objects which are basically lists of **`GoalMetaData_T`** objects. The globaldefs::NamingAttributes_T structure has been used as the base to design and implement the `Goal` entity and the rest of our naming schemes throughout the prototype as IDL files.

The Figure 88 shows a fragment of the meta data types IDL file (`GoremochMetaDataTypes.idl`) that defines the data structures of our distributed environment. The fragment shows the **`GoalMetaData_T`** structure that specialises the `Goal` entity described above. The **`GoalMetaDataList_T`** shown in the lower part of Figure 88 defines a sequence of **`GoalMetaData_T`** structures, namely to specialise Goal Selections, goal sub-graphs, etc.

```
#ifndef       GoremochMetaDataTypes_idl
#define       GoremochMetaDataTypes_idl

#include <Mtnm/globaldefs.idl>

module Goremoch{

/**
* Used to specify the Goal entity to bring the gap between
* Objectiver concepts and our policy refinement framework.
*
**/
struct GoalMetaData_T {

globaldefs::NamingAttributes_T goalName;
globaldefs::NamingAttributes_T refinedSons;
globaldefs::NamingAttributes_T tempRelationship;
globaldefs::NamingAttributes_T goalDetails;
globaldefs::NamingAttributes_T temporalPrescription;

};

/**
* Sequence of GoalMetaData_T.
*/
typedef sequence<GoalMetaData_T> GoalMetaDataList_T;
```

Figure 88. Definition of `Goal` entity with the DSC framework

Goal Management Components

The Figure 89 presents the context of our Goal Management components. The Goal Manager component's main class is identified as GoalManager. It provides the i_GoalManager interface and implements an internal class identified as ObjectiverCoordinator. The Objectiver toolkit is identified as the ObjectiverService class and the interface provided by this service is identified as i_Objectiver.

When a Request for Policy Refinement (RPR) is submitted through the i_GoalManager interface, the Goal Manager is in charge of finding all the goals that are influenced or affected by the Goal Selection. The latter goals are formalised into `Goal` entities specialised as **`GoalMetaData_T`** structures. In order for the Goal Manager to assess this, it executes iterative queries to Objectiver through the i_Objectiver interface. The resulting `Goal` entities are compiled in a **`GoalMetaDataList_T`**. The following sub-Sections provide a brief description of the main methods illustrated in Figure 89.

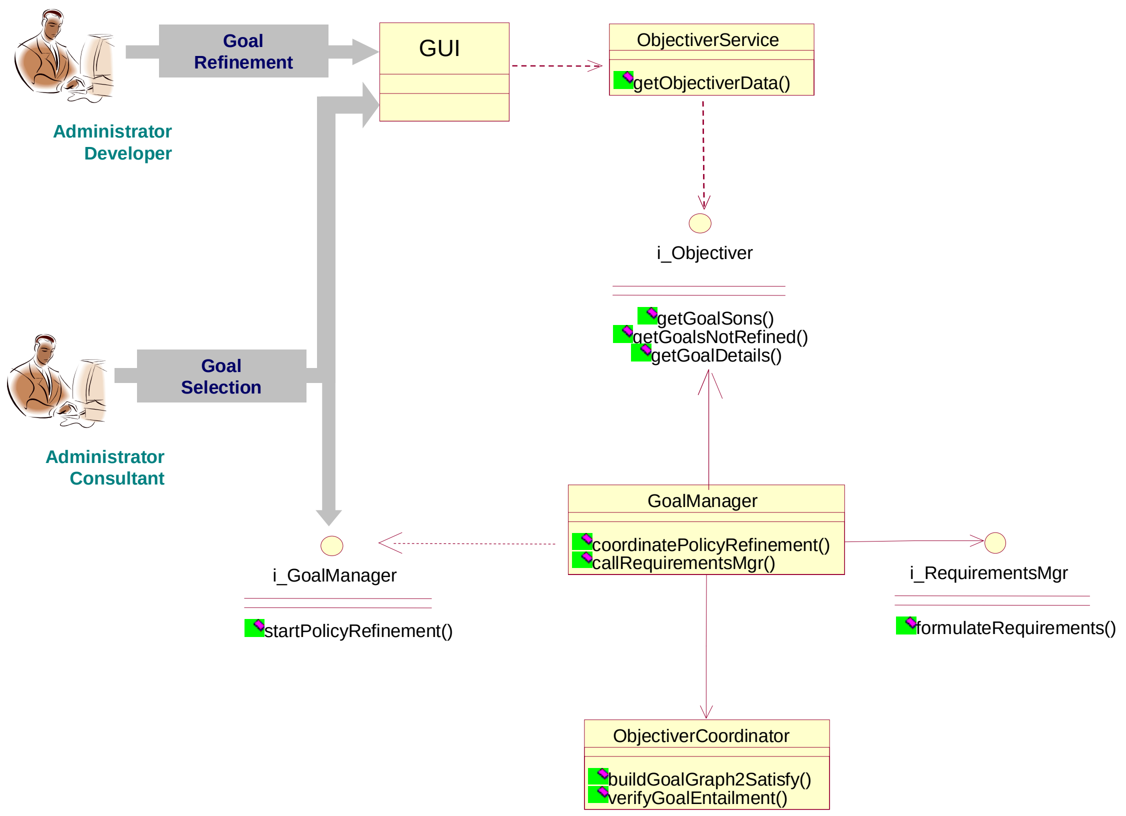


Figure 89. Goal Management Components

Goal Manager

CLASS/INTERFACE NAME	**i_GoalManager**
Description	The public interface of the Goal Manager. This is used to receive the Request for Policy Refinement (RPR) submissions

Operation Name	startPolicyRefinement	
Description	This is used by the Administrator Consultant to submit the RPR and then to initiate the automatic acquisition of policies that may fulfill the Goal Selection.	
Argument Name	**Argument Type**	**Description**
goalToSatisfy	String	When a RPR is submitted, the submission may include the High-level Goal that the consultant desires to fulfill. The Goal Selection has been documented in the Objectiver GUI. This argument is used to drive the querying process with the Objectiver package. If null, the RPR is processed with the informa-

		tion provided by the objectiverData argument (explained bel-low)
objectiverData	ObjectiverMeta-DataList_T	The consultant may provide Objectiver generic objects repre-senting complete goal graph structures. The Goal Selection has been formulated through the Objectiver GUI and the goal graph structure of the selection is submitted through this ar-gument. This is used to drive the querying process as well. If null, the RPR is processed with the information provided by the goalToSatisfy argument (explained above)
adminConstraints	Administrative-ConstraintsList_T	The applicability of the policies resulting from the refinement process may be restricted to administrative decisions. This ar-gument is a list of administrative constraints that express the conditions on which the High-level Goals should be fulfilled.
Returns	Boolean	

CLASS/INTERFACE NAME	**GoalManager**
Description	DSC Goal Manager Component. This class coordinates the Request for Policy Re-finement (RPR) processing, the execution of queries to the Objectiver package and the submission of the verified Goal Selection to the Requirements Manager

Operation Name	coordinatePolicyRefinement	
Description	Instantiates an ObjectiverCoordinator object which in turn processes the RPR.	
Argument Name	**Argument Type**	**Description**
goalToSatisfy	String	This is used to drive the queries to the Objectiver package as the means to acquire the Goal entities/attributes involved in the Goal Selection. If null, the querying process is driven by the objectiverData argument (explained bellow)
objectiverData	ObjectiverMeta-DataList_T	This is used to drive the queries to the Objectiver package as the means to acquire the Goal entities/attributes involved in the Goal Selection. If null, the querying process is driven by the goalToSatisfy argument (explained above)
adminConstraints	Administrative-ConstraintsList_T	List that express the constraints of the applicability of the to-be-refined policies
Returns	Boolean	

Operation Name	callRequirementsMgr	
Description	This method is in charge of providing the verified goal sub-graph to the Require-ments Manager. It also provides the administrative constraints that the consultant may have introduced during the RPR submission	
Argument Name	**Argument Type**	**Description**
goalData	GoalMetaDataL-ist_T	This is a list of logically-correct Goal Selection
adminConstraints	Administrative-ConstraintsList_T	A list of administrative constraints to express the conditions on which the verified Goal Selection should be fulfilled

Returns	Boolean

CLASS/INTERFACE NAME	**ObjectiverCoordinator**
Description	This class is in charge of processing the Request for Policy Refinement (RPR). It builds a goal sub-graph according to the Goal Selection. It also verifies that the Goal Selection logically entails High-level Goal fulfillment.

Operation Name	buildGoalGraph2Satisfy	
Description	Returns a GoalMetaDataList_T object that compiles the Goal entities that are affected or that influence the Goal Selection. It coordinates and drives the interaction with Objectiver. It uses the getGoalsNotRefined, getGoalSons and getGoalDetails methods from the i_Objectiver interface	
Argument Name	**Argument Type**	**Description**
goalToSatisfy	String	The Goal Selection has been documented in the Objectiver package. This argument is the highest-level goal that will be satisfied with the to-be-refined policies. Under these circumstances, the querying process is intended to acquire the goal sub-graph involved in the Goal Selection, namely the `Goal` entities that integrate the selection. If null, the querying process is driven by the objectiverData argument (described bellow)
objectiverData	ObjectiverMeta-DataList_T	The Goal Selection is provided in this object from the Objectiver GUI. Under this circumstance, the querying process is intended to map the submitted ObjectiverMedatata object (Objectiver goal sub-graph) into the goal-oriented information model of our prototype. If null, the querying process is driven by the goalToSatisfy argument (described above)
Returns	GoalMetaDataList_T	

Operation Name	verifyGoalEntailment	
Description	This method coordinates the verification of the correctness and consistency of the compiled Goal Selection. It uses the temporal prescription and the logical relationships between goals to make sure that the Goal Selection logically entails the fulfillment of High-level Goals	
Argument Name	**Argument Type**	**Description**
goalSelection	GoalMetaDataL-ist_T	Acquired Goal Selection compiled into a `Goal` entity list.
Returns	Boolean	

Objectiver Package

CLASS/INTERFACE NAME	**i_Objectiver**
Description	This is an interface intended to make systematic queries to the Objectiver toolkit. It implements the Objectiver Open API to access the Objectiver database.

Operation Name	getGoalSons	
Description	Returns the list of refinements of a parent goal	
Argument Name	**Argument Type**	**Description**
parentGoal	String	The parent goal for which the goal refinements are required
Returns	String []	

Operation Name	getGoalsNotRefined	
Description	Returns the list of lowest-level goals from a goal graph	
Argument Name	**Argument Type**	**Description**
goalGraph	String	The goal graph for which the lowest-level goals are required
Returns	String []	

Operation Name	getGoalDetails	
Description	Returns the list of details of a specific goal. The details are compiled into a GoalMetaData_T object as `Goal` entities	
Argument Name	**Argument Type**	**Description**
goal2Detail	String	The goal for which the details are required
Returns	GoalMetaData_T	

COMPONENTS OF THE POLICY REFINEMENT MECHANISMS

This sub-Section describes the overall implementation of the Requirements Manager, Search Manager and Policy encoder. In addition, we describe the supporting components that make it possible to automate the process, namely the Inventory Manager and Behaviour Manager.

Classes of the Policy Refinement Mechanisms components

The Figure 90 illustrates the overall classes of the components specialising the Policy Refinement Mechanisms for which a brief description is provided in the following sub-Sections.

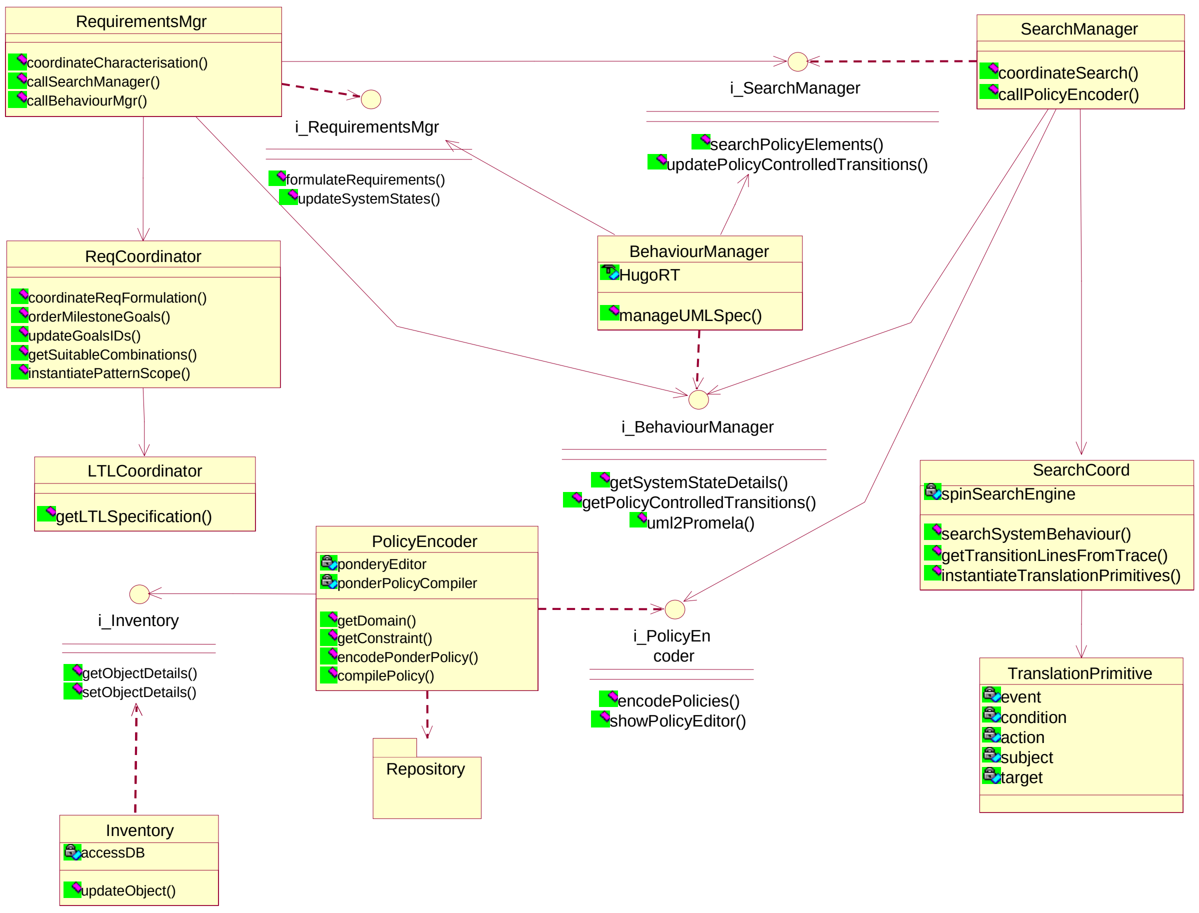


Figure 90. Class diagram of the Policy Refinement Mechanisms

Refinement Manager Implementation

The Requirements Manager is in charge of deducing goal fulfillment characterisations represented in Linear Temporal Logic (LTL) formulae. The component provides the i_RequirementsMgr interface to receive verified Goal Selections from the Goal Manager. It uses the i_SearchManager provided by the Search Manager to submit the LTL goal characterisations.

The management tasks of this DSC component are executed by the ReqCoordinator class. It implements a database of behavioural properties classified by pattern/scopes. The latter is identified as LTLCoordinator and allows producing LTL formulae from behavioural properties in runtime. In order to produce meaningful and useful LTL goal characterisations, this component should provide the goal fulfillment characterisations in terms of the actual PROMELA specification (input language of SPIN). In other words, the goal characterisation must make reference to states that can be interpreted by the SPIN searching engine in order to allow further analysis.

CLASS/INTERFACE NAME	**i_RequirementsMgr**
Description	This is the public interface of the Requirements Manager component. It is the interface between the Goal Management components and the components specialising the automated policy refinement mechanisms.

Operation Name	formulateRequirements	
Description	Method provided to receive the verified Goal Selection from the Goal Manager	
Argument Name	**Argument Type**	**Description**
goalMetaDataList	GoalMetaDataList_T	Verified Goal Selection with the meaningful information abstracted from the Objectiver package
adminConstraints	AdministrativeConstraintsList_T	Administrative constraints to express the conditions on which the verified Goal Selection should be fulfilled
Returns	Boolean	

Operation Name	updateSystemStates	
Description	Receives the details of PROMELA specification that corresponds to the specification of the lowest-level goals (used by the Behaviour Manager).	
Argument Name	**Argument Type**	**Description**
goalMetaDataList	GoalMetaDataList_T	List of goals for which the update is provided
systemDataList	SystemDataList_T	List of state IDs corresponding to the goals provided in the goalMetaDataList argument. The length of this argument should be the same as for the goalMetaDataList argument
Returns	Boolean	

CLASS/INTERFACE NAME	**RequirementsMgr**
Description	DSC Requirements Manager Component. This class coordinates the deduction process of meaningful LTL goal fulfillment characterisations and their submission to the Search Manager. It also executes queries to the service provided by the Behaviour Manager to acquire the PROMELA state identifications for lowest-level goals.

Operation Name	coordinateCharacterisation	
Description	Instantiates a ReqCoordinator object which in turn processes the verified Goal Selection and coordinates the requirements characterisation step	
Argument Name	**Argument Type**	**Description**
goalMetaDataList	GoalMetaDataList_T	The list of `Goal` entities for which the LTL goal characterisation will be coordinated
adminConstraints	AdministrativeConstraintsList_T	Administrative constraints on which the LTL goal characterisation should be fulfilled
Returns	Boolean	

Operation Name	callSearchManager

Description	Submits the LTL goal fulfillment characterisation and the constraints on their applicability to the Search Manager	
Argument Name	**Argument Type**	**Description**
ltlPropertyList	LTLPropertyList_T	LTL goal characterisation to the Search Manager
adminConstraints	Administrative-ConstraintsList_T	Administrative constraints on which the LTL goal characterisation should be fulfilled
goal2Fulfill	GoalMetaData_T	The goal characterisation corresponds to a the fulfillment of High-level Goal which is provided by this argument
Returns	Boolean	

Operation Name	callBehaviourMgr	
Description	Updates the PROMELA specification for the lowest-level goal state identifications	
Argument Name	**Argument Type**	**Description**
stateDetails	GoalMetaData_T	The lowest-level goals for which the PROMELA specification is required. The Behaviour Manager returns this data using the i_RequirementsMgr interface
Returns	Boolean	

CLASS/INTERFACE NAME	**ReqCoordinator**
Description	This class coordinates the LTL goal characterisation process. The fulfillment of High-level Goals entails the fulfillment of lowest-level goals. Consequently, the LTL goal characterisation should be a correlation of the fulfillment of lowest-level goals. In order for this characterisation to be meaningful for further analysis, it should be expressed in terms of the actual PROMELA specifications. The main method of this class is the coordinateReqFormulation which in turn makes use of internal private methods to coordinate the following general process: **Input**: A verified Goal Selection $[\mathtt{Goal}_1,..,\mathtt{Goal}_u]$specialised in a `Goal-MetaData_List` object **Procedure**: • Deduce the High-level Goal $\mathtt{Goal}_{HG}$ for which the characterisation is processed • Deduce the lowest-level goals $\mathtt{Goal}_{LL}$ that make $\mathtt{Goal}_{HG}$ be fulfilled • Order lowest-level goals by a temporal fulfilment: $\mathtt{Goal}_{LL} = [\mathtt{Goal}_{L1},...,\mathtt{Goal}_{Lv}]$ • Establishing temporal relationships between lowest level goals is imperative to characterise goal fulfilment. This step formalises suitable groups of temporal relationships between lowest-level goals. This grouping step is driven by the feasibility to map temporal relationships to available pattern/scope properties in the database: $\mathtt{TR}[\mathtt{Goal}_{LL}] = [\mathtt{TR}_1,...,\mathtt{TR}_w]$ • Instantiate the temporal relationship that characterises goal fulfilment with the corresponding pattern/scopes of the property database: $\mathtt{LTL}[\mathtt{TR}_1,...,\mathtt{TR}_w] = [\mathtt{ltl}_1,...,\mathtt{ltl}_y]$ • Update the specification of the states prescribing the lowest-level goals with the actual PROMELA specification **Output**: Meaningful LTL formulae characterising goal fulfilment

Operation Name	orderMilestoneGoals	
Description	Deduces the lowest-level goals that make a higher-level goal be fulfilled. The method returns a list of goals ordered by temporal fulfilment	
Argument Name	**Argument Type**	**Description**
goalMetaDataList	GoalMetaData_T	The list of goals from which the temporal order of fulfillment is required
goal2Fulfill	GoalMetaData_T	The High-level Goal that will drive the ordering process
Returns	GoalMetaDataList_T	

Operation Name	updateGoalsIDs	
Description	Returns the PROMELA naming specification of every lowest-level goal of the provided goal list	
Argument Name	**Argument Type**	**Description**
milestones2Fulfill	GoalMetaDataList_T	The list of goals for which the PROMELA specification is required
Returns	GoalMetaDataList_T	

Operation Name	getSuitableCombinations	
Description	Returns a list of suitable combinations to group the temporally-ordered goal list. For this task the method uses combinations of pattern/scope available in the LTL property database implemented by the LTLCoordinator class.	
Argument Name	**Argument Type**	**Description**
goals2Customise	GoalMetaDataList_T	List of temporally-ordered goals for which a suitable combination is required
Returns	String []	

Operation Name	instantiatePatternScope	
Description	Returns the LTL formula characterising the fulfillment of the pattern/scope argument. It uses the getLTLSpecification method of the database LTLCoordinator	
Argument Name	**Argument Type**	**Description**
patternScope	String[]	Pattern/scope combination
Returns	String	

Search Manager Implementation

The Search Manager is in charge of producing policy fields from goal fulfillment characterisations represented in Linear Temporal Logic (LTL). The component provides the i_SeachManager interface to receive the characterisations from the Requirements Manager and uses the i_PolicyEncoder to submit the policy fields for further tasks. The policy fields produced by this component should reflect the actual events, conditions and actions implemented in the system. For this, the Search Manager makes use of the managed system documentation handled by the Behaviour Manager.

The management tasks of this DSC component are executed by the SearchCoord class. It implements a SPIN search engine that allows finding the system behaviour necessary to commit with the LTL characterisation in runtime. Once the system behaviour is acquired, the next step is to abstract the necessary policy fields that should reproduce such behaviour. In order to automate this process, the Search Manager applies the Translation Process which are in turn implemented by the TranslationPrimitive class.

CLASS/INTERFACE NAME	**i_SearchManager**
Description	The public interface of the Search Manager. This is used to receive the LTL goal fulfillment characterisations and to receive notifications about the managed system state transitions that are controlled by policies

Operation Name	searchPolicyElements	
Description	Used to receive the goal fulfillment characterisations	
Argument Name	**Argument Type**	**Description**
ltlPropertyList	LTLPropertyList_T	A list of states and their occurrence in temporal ordering.
adminConstraints	Administrative-ConstraintsList_T	Administrative constraints on which the refined policies should apply
goals2Satisfy	GoalMetaDataL-ist_T	For informative purposes, the verified selection of goals
goal2Fulfill	GoalMetaData_T	For informative purposes, the High-level Goal to which the policy fields are correlated.
Returns	Boolean	

Operation Name	updatePolicyControlledTransitions	
Description	Receives the transitions that are controlled by policies in the managed system (used by the Behaviour Manager)	
Argument Name	**Argument Type**	**Description**
candidateTransi-tions	PolicyAttributes-List_T	Lst of transitions that are controlled by policies. This is used to drive the application of the Translation Process
Returns	Boolean	

CLASS/INTERFACE NAME	**SearchManager**
Description	DSC Search Manager Component. This class coordinates the abstraction of policy fields and their submission to the Policy Encoder.

Operation Name	coordinateSearch	
Description	Instantiates a SearchCoord object which in turn coordinates the searching process	
Argument Name	**Argument Type**	**Description**
ltlPropertyList	LTLPropertyList_T	List of states and their occurrence in temporal ordering. The searching process is coordinated to first search the system behaviour necessary to fulfil such ordering of states and eventually to abstract the policy fields that should reproduce that be-

		haviour in runtime
adminConstraints	Administrative-ConstraintsList_T	Administrative constraints on which the policy fields should apply
goals2Satisfy	GoalMetaDataL-ist_T	For informative purposes, the verified selection of goals
goal2Fulfill	GoalMetaData_T	For informative purposes, the High-level Goal to which the policy fields are correlated
Returns	Boolean	

Operation Name	callPolicyEncoder	
Description	Submits the policy fields and the constraints on their applicability to the Policy Encoder	
Argument Name	**Argument Type**	**Description**
policies	PolicyAttributes-List_T	List of ECA (Event-Condition-Action) policies that fulfill the temporal ordering of goals.
constraints	Administrative-Constraints_T	Administrative constraints on which the policy fields should apply
goals2Satisfy	GoalMetaDataL-ist_T	For informative purposes, the verified selection of goals
goal2Fulfill	GoalMetaData_T	For informative purposes, the High-level Goal to which the policy fields are correlated
Returns	Boolean	

CLASS/INTERFACE NAME	**SearchCoord**
Description	This class coordinates the searching process. The constructor receives a list of states and that must occur in a specific temporal ordering. With this information this class first coordinates the search of behaviour commiting to that specific ordering of goals and then it applies the Translation Process to abstract the policies that should reproduce such behaviour in runtime. The general process coordinated with this class is summarized as follows: **Input**: A LTL formula characterising goal fulfillment **Procedure**: • Produce a system execution trace ***K*** that satisfies the occurrence of states prescribed by the LTL characterisation. The execution should be expressed in terms of events, conditions and actions implemented by the real system • From the system execution, identify a set of decision-based (i.e. enforceable) transitions $\boldsymbol{T}[K]=[T_1,...,T_u]$. • Apply Translation Process: For (i = 1 to i=u): Find the precondition event of $\boldsymbol{T_i}$ = event$_{Ti}$ Find the managed object **MO** issuing event$_{Ti}$ **MO**(event$_{Ti)}$ = subject$_{Ti}$ Find the managed object **MO** executing the enforceable transition T_i **MO**(T_i)= target$_{Ti}$. Find the action **A** that represents the enforceable transition T_i **A**(T_i) = action$_{Ti}$

	Create a recipient $\mathbf{PF}_{Ti}$ for policy fields $\mathbf{PF}_{Ti}$ = [event$_{Ti}$, subject$_{Ti}$, target$_{Ti}$, action$_{Ti}$] **Output**: Policy fields **PF**=[PF$_{Ti}$,…, PF$_{Tu}$]

Operation Name	searchSystemBehaviour	
Description	Produces an execution trace of system behaviour that commits with the temporal ordering of goals prescribed by the LTL goal characterisation. It uses the SPIN searching engine attached as an argument of the SearchCoord class	
Argument Name	**Argument Type**	**Description**
ltlPropertyList	LTLPropertyList_T	Temporal ordering of goals that the execution trace should commit to
goal2Fulfill	GoalMetaData_T	For informative purposes, the High-level Goal to which the policy fields are correlated
Returns	Boolean	

Operation Name	getTransitionLinesFromTrace	
Description	Returns pointers to the policy-controlled transition plans within the execution trace	
Argument Name	**Argument Type**	**Description**
HLgoal	GoalMetaData_T	The High-level Goal to which the transition plans are returned
Returns	String []	

Operation Name	instantiateTranslationPrimitives	
Description	Returns the policy fields involved in the system trace execution correlated with the fulfillment of the HLgoal argument. It instantiates the TranslationPrimitive class for every transition plan correlated with the HLgoal argument	
Argument Name	**Argument Type**	**Description**
HLgoal	GoalMetaData_T	The High-level Goal for which the policy fields are returned
Returns	PolicyAttributesList_T	

Policy Encoder Implementation

The Policy Encoder is in charge of producing deployable Ponder policies from the policy fields provided by the Search Manager. Although these policies reflect the actual event, conditions and actions implemented in the managed system, the policies produced by the Policy Encoder are ready to be deployed onto the managed objects which would enforce them in runtime.

This component provides the i_PolicyEncoder interface to receive the policy fields and uses the i_Inventory interface to get the details of the actual Object Distribution of the managed objects and hence to store deployable policies in the LDAP (Lightweight Directory Access Protocol) policy repository. Although the framework is not limited to any particular language, we have used the Ponder policy specification language [Dam02] given that the latter provides the means to implement the Event-Condition-Action general structure of policies. In order to automate the policy edition and compilation step of our prototype, we have included a ponder syntax parser and a slightly modified Ponder compiler [Rub06b].

CLASS/INTERFACE NAME	**i_PolicyEncoder**
Description	The public interface of the Policy Encoder. This is used to receive the policy fields from the Search Manager.

Operation Name	encodePolicies	
Description	Starts the coding, compiling and storage of the Ponder policy	
Argument Name	**Argument Type**	**Description**
policyAttributesList	PolicyAttributes-List_T	The policy fields that would eventually result in deployable Ponder policies
adminConstraints	Administrative-ConstraintsList_T	Administrative constraints on which the policy instances should apply
goals2Satisfy	GoalMetaDataL-ist_T	The verified selection of goals for which the deployable policies are encoded
goal2Fulfill	GoalMetaData_T	The correlated High-level Goal for which the deployable policies are encoded
Returns	Boolean	

Operation Name	showPolicyEditor	
Description	Instantiates a Ponder policy editor. The Ponder GUI facility is provided to edit policies explicitly i.e. not correlated with goals	
Argument Name	**Argument Type**	**Description**
None	None	The method does not need any argument
Returns	Boolean	

CLASS/INTERFACE NAME	**PolicyEncoder**
Description	DSC Policy Encoder component. It coordinates the encoding, parsing, and compiling processes of the Policy Encoder component

Operation Name	getDomain	
Description	Returns the directory on which the policies would eventually be deployed. This method is applied to find the Object Distribution of subjects and targets in the ldap repository	
Argument Name	**Argument Type**	**Description**
policyAttributes	PolicyAttributes_T	The policy fields for which the directory of the corresponding subjects and targets are returned
adminConstraints	Administrative-ConstraintsList_T	The applicability of the policies resulting from the refinement process may be restricted to administrative decisions. This argument is a list of administrative constraints that may be used to abstract the domain of the subjects and targets on which the

		policies may be enforced
Returns	String[]	

Operation Name	getConstraint	
Description	The applicability of the policies resulting from the refinement process may be restricted to administrative decisions. This method abstracts the Ponder constraint field from the administrative constraints provided by the consultant when the Request for Policy Refinement (RPR) is submitted.	
Argument Name	**Argument Type**	**Description**
adminConstraints	Administrative-ConstraintsList_T	List of administrative constraints that express the conditions on which the Ponder policies should be enforced
Returns	String	

Operation Name	encodePonderPolicy	
Description	Parses the policy fields to Ponder syntax	
Argument Name	**Argument Type**	**Description**
updatedPolicyFields	PolicyAttributes_T	Subjects, targets, events, actions to be parsed
constraint	String	Administrative constraint (if any)
Returns	String	

Operation Name	compilePolicy	
Description	Compiles and stores the Ponder deployable policies	
Argument Name	**Argument Type**	**Description**
ponderPolicyString	String	The Ponder string of the deployable policy
Returns	Boolean	

Supporting Components Implementation

These components are the Behaviour Manager and the Inventory component. Both have been included to manage the System Model as to automate the generation of policies within our prototype. The former provides the means to translate UML standard representations into PROMELA code, the input language of SPIN. This approach is an added value of our prototype since the administrative parties are free from using complex or proprietary notations. On the other hand, the Inventory component provides the means to keep updates of the logical representation of the object distribution. The Policy Editor for instance makes use of the inventory component as to find the details of the managed objects involved in the to-be-encoded policies. This information is crucial to deduce the subject and target directories over which the policies may be deployed/enforced. The introduction of the Inventory Component has enabled to automate the policy refinement process considering that such database is populated for a specific policy applicability domain. The following is the outline of the services provided by the supporting component, provided through the i_BehaviourManager and i_Inventory interfaces.

CLASS/INTERFACE NAME	**i_BehaviourManager**
Description	The public interface of the Behaviour Manager. This is used to provide details about the System Model. The three methods provided by this interface are specialisations of the generic manageUMLSpec method implemented by the DSC Behaviour Manager component

Operation Name	uml2Promela	
Description	Returns the PROMELA specification of the UML model argument	
Argument Name	**Argument Type**	**Description**
model	String	Name of the UML model that will be translated into PROMELA code
Returns	File	

Operation Name	getPolicyControlledTransitions	
Description	Returns an updated list including the state transitions that are controlled by policies	
Argument Name	**Argument Type**	**Description**
candidateTransi-tions	PolicyAttributes-List_T	List of objects for which the policy controlled transitions are required
Returns	PolicyAttributesList_T	

Operation Name	getSystemStateDetails	
Description	Returns an updated list including the PROMELA specification of the state predi-cates that specialise the lowest-level goals of the list	
Argument Name	**Argument Type**	**Description**
stateDetails	GoalMetaDataL-ist_T	List goals for which the PROMELA specification is required
Returns	GoalMetaDataList_T	

CLASS/INTERFACE NAME	**i_Inventory**
Description	The public interface of the Inventory component. This is used to provide details about the managed system Object Distribution during the operation of the system and also to keep an updated database of managed objects during the design of the policy system.

Operation Name	getObjectDetails	
Description	Returns an updated list including the logic distribution of the managed objects that would specialise the subject/target fields of the deployed policies.	
Argument Name	**Argument Type**	**Description**
objectDescription	InventoryObject-DataList_T	The name of the object for which the directory representation is required
Returns	InventoryObjectDataList_T	

Operation Name	setObjectDetails

Description	This method is used to populate the internal database of managed objects.	
Argument Name	**Argument Type**	**Description**
object2Populate	InventoryObject-DataList_T	This contains attributes like object type, object ID, object domain, attributes, etc.
Returns	Boolean	

Appendix D. Publications of the Author related to the Thesis

• J. Gutierrez, J.L. Melus, J.Serrat, J.Rubio. "Pricing Pervasive Services Using Policy-based Mechanisms", 2007 International Conference of the Information Resources Management Association IRMA. May 19-23, 2007, Vancouver, Canada

• J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G.Pavlou. “A Methodological Approach towards the Refinement Problem in Policy-based Management Systems”. ” IEEE Communications Magazine, Vol. 44, No. 10, IEEE, October 2006

• J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G.Pavlou. “A Functional Solution for Goal-oriented Policy Refinement”. IEEE International Workshop on Policies for Distributed Systems and Networks, POLICY 2006. 5-7 June 2006 The University of Western Ontario. London , Ontario CANADA

• M.Charalambides, P. Flegkas, G. Pavlou, J. Rubio-Loyola, A. Bandara, E. Lupu, A. Russo, M. Sloman, N. Dulay “Dynamic Policy Analysis and Conflict Resolution for DiffServ Quality of Service Management” Network Operations and Management Symposium NOMS 2006 Vancouver, Canada

• J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G.Pavlou. “GOREMOCH: A Distributed Goal-oriented Policy Refinement Environment”. 10th IEEE/IFIP Network Operations and Management Symposium, NOMS 2006 (short paper) April 2006. Vancouver, Canada

• F. Karayannis, J. Serrat, J. Baliosian, J. Rubio-Loyola, et al. “In-field Evaluation of a Managed IP/MPLS over WDM Provisioning Solution”. IEEE Communications Magazine, Volume 43, Issue 11, Nov. 2005

• M.Charalambides, P. Flegkas, G. Pavlou, A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, J. Rubio-Loyola. “Policy Conflict Analysis for Quality of Service Management”. Sixth IEEE International Workshop on Policies for Distributed Systems and Networks POLICY 2005, Stockholm, Sweden June 6-8, 2005

• J. Rubio-Loyola, J. Serrat, M.Charalambides, P. Flegkas, G. Pavlou, A. Lluch. “Using Linear Temporal Model Checking for Goal-oriented Policy Refinement Frameworks” Sixth IEEE International Workshop on Policies for Distributed Systems and Networks POLICY 2005, Stockholm, Sweden June 6-8, 2005

• J. Rubio-Loyola, J. Serrat, J. Mata, F. Casals. “A Policy-Based Management Solution towards QoS-aware Video Streaming Services in WLAN Environments”. IEEE/ACM International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks, QSHINE 2004. Dallas Tx. USA Oct 2004

• J. Rubio, J. Serrat and A. Angeles. “An Initial Approach to Automate Policy Deployment in a Network Management Solution for IP/MPLS over Transport Networks”, 3rd International Congress of Telematic Engineering, La Habana, Cuba Nov 30- Dec 3, 2004.

• Vardalachos, N., Rubio, J., Galis, A., Serrat J., "A Policy Management System for Hybrid Networks", London Communications Symposium (LCS 2003), London, UK September 2003

• Grampín, E., Rubio, J., Vardalachos, N., Galis, A., Serrat, A., "Implementation Issues in PBNM Systems", published in Hiradastechnika Magazine (in Hungarian) Volume LVII 2002/3 Március

• Grampin, E., Rubio, J., Vardalachos, N., Galis, A., Serrat, J., "Implementation issues of policy based network management systems", proceedings of the 3rd IEEE International Workshop on Design of Reliable Communication Networks (DCRN2001), Budapest, Hungary, 7-10 October 2001

References

[Alb05] J.P. de Albuquerque et al. , H. Krumm, P.L. de Geus. “Policy Modeling and Refinement for Network Security Systems”. IEEE POLICY, Stockholm, Sweden 2005

[Amm99] Ammann, P.; Black, P.E “Abstracting formal specifications to generate software tests via Model Checking” Proceedings of 18th Digital Avionics Systems Conference. Volume 2, 24-29 Oct. 1999

[Arg06] Argo UML Tigris.org Open Source Software Engineering Tools. http://argouml.tigris.org/ (web page info updated in September 2006)

[Bac98] Bacchus, F., and F. Kabanza. “Planning for Temporally Extended Goals” Annals of Mathematics and Artificial Intelligence, Volume 22 Numbers 1-2, p.p. 5-27, 1998

[Bac00] F. Bacchus and F. Kabanza, "Using temporal logics to express search control knowledge for planning", Artificial Intelligence Journal, Volume 116, Number 1-2, January 2000

[Bal04] M. Balser, S. Bäumler, A. Knapp, W. Reif, and A. Thums. “Interactive verification of UML state machines”. Intl. Conference on Formal Engineering Methods 2004

[Ban03] A. K. Bandara, E. C. Lupu, and A. Russo. "Using Event Calculus to Formalise Policy Specification and Analysis." In Proceedings of 4th IEEE International Workshop on Policies for Networks and Distributed Systems (Policy 2003), Lake Como, Italy, IEEE, June 2003.

[Ban04] A.K. Bandara, E.C. Lupu, J. Moffett, A. Russo; "A goal-based approach to policy refinement" Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004

[Ban05a] A. K. Bandara et. al. ”Policy Refinement for DiffServ Quality of Service Management” IEEE International Symposium on Integrated Network Management (IM 2005) Nice, France, May 2005

[Ban05b] Bandara A. K., "A Formal Approach to Analysis and Refinement of Policies", PhD Thesis, Imperial College London, 2005

[Bar06] C. Bartolini, M. Sallé, D. Trastour. "IT Service Management driven by Business Objectives - An Application to Incident Management" IEEE/IFIP NOMS, April 2006

[Bea93] Bean, A.; Wood, D.; Fairclough, W.; Specifying goal-oriented network management systems. IEEE Communications Magazine, Volume 31, Issue 5, May 1993 Page(s):30 - 36

[Car04] Kevin Carey, David Lewis, Vincent Wade. Automated Policy-Refinement for Managing Composite Services. M-Zones White Paper June 04, white paper 06/04, Ireland, June, 2004

[Car05] Kevin Carey. Automatically Refining Composite Service's Adaptive Behaviour Policies. M-Zones White Paper July 05, white paper 07/05, Ireland, July, 2005

[Cas00] M. Casassa, A. Baldwin, C. Goh. “POWER prototype: towards integrated policy-based management”. Network Operations and Management Symposium, NOMS 2000

[Cha98] Chan, W.; Anderson, R.J.; Beame, P.; Burns, S.; Modugno, F.; Notkin, D.; Reese, J.D.; “Model checking large software specifications”. IEEE Transactions on Software Engineering. Volume 24, Issue 7, July 1998 Page(s):498 - 520

[Cha05] M.Charalambides, P. Flegkas, G. Pavlou, A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, J. Rubio-Loyola. “Policy Conflict Analysis for Quality of Service Management” Sixth IEEE Int. Workshop on Policies for Distributed Systems and Networks. (POLICY 2005), Stockholm, Sweden June 6-8, 2005

[Cha06] M.Charalambides, P. Flegkas, G. Pavlou, J. Rubio-Loyola, A. Bandara, E. Lupu, A. Russo, M. Sloman, N. Dulay “Dynamic Policy Analysis and Conflict Resolution for DiffServ Quality of Service Management” Network Operations and Management Symposium NOMS`06 Vancouver, Canada

[Chu00] L. Chung, B. Nixon, E. Yu and J. Mylopoulos, “Non-functional requirements in software engineering”. Kluwer Academic, Boston, 2000.

[Cim99] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. “NuSMV: A New Symbolic Model Verifier”. Proceedings of the 11th International Conference on Computer Aided Verification (CAV’99), July 1999, vol. 1633 of LNCS pages 495–499. Springer-Verlag, July 1999.

[Cim03] A. Cimatti , M. Pistore , M. Roveri , P. Traverso, “Weak, strong, and strong cyclic planning via symbolic Model Checking” Artificial Intelligence, v.147 n.1-2, p.35-84, July 2003

[Cla99] E.M. Clarke, O. Grumberg, and D.A. Peled. "Model Checking". The MIT Press, 1999

[Dam01] N. Damianou, N. Dulay, E. Lupu, M Sloman, : *The Ponder Specification Language* Workshop on Policies for Distributed Systems and Networks (Policy2001), HP Labs Bristol, 29-31 Jan 2001

[Dam02] N. Damianou, T. Tonouchi, N. Dulay, E. Lupu, and M. Sloman. "Tools for doamin-based policy management of distributed systems", NOMS, Friorence, Italy, 2002

[Damil02] IST IST-1999-11253-TEQUILA – Traffic Engineering for Quality of Service in the Internet at Large Scale. Deliverable 3.4. Final System Evaluation. Editor. Takis Damilatis. October 2002

[Dan04] Danciu, V., Kempter, B., “From processes to policies —– concepts for large scale policy generation”, *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Seoul, Korea, April, 2004.

[Dar93] A. Dardenne, A. van Lamsweerde and S. Fickas, “Goal-Directed Requirements Acquisition”, *Science of Computer Programming*, Vol. 20, 1993, 3-50.

[Dar95] R. Darimont, "Process Support for Requirements Elaboration", PhD Thesis, Université Catholique de Louvain, Dépt. Ingénierie Informatique, Louvain-la-Neuve, Belgium, 1995

[Dar96] R. Darimont and A. van Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration," 4th ACM Symposium on the Foundations of Software Engineering (FSE4) No. 179-190, 1996.

[Dar98]R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde, “GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering”, *Proc. ICSE’98 - 20th Intl. Conf. on Software Engineering*, Kyoto, April 1998, vol. 2, 58-62.

[Dav06] Davy, S., Jennings, B., & Strassner, J. Conflict Prevention via Model-driven Policy Refinement. Proceedings of 17th IFIP/ IEEE International Workshop on Distributed Systems: Operations and Management (DSOM) 2006

[Del03] E. Delor, R. Darimont, A. Rifaut. “Software Quality Starts with the Modelling of Goal-Oriented Requirements”. 16th International Conference on Software & Systems Engineering and their Applications (ICSSEA 2003). Paris, France – December 2-4, 2003

[DMTF-CIM] Distributed Management Task Force, Common Information Model (CIM), http://www.dmtf.org/standards/cim/

[Dwy98] M. B. Dwyer, G. S. Avrunin and J. C. Corbett. “Property specification patterns for finite-state verification” Workshop on Formal Methods in Software Practice 1998

[Dwy] M. Dwyer, G. Avrunin, and J. Corbett. “A system of specification patterns”. http://patterns.projects.cis.ksu.edu/

[Ede01] Edelkamp, S., and Helmert, M. 2001. “The model-checking integrated planning system”. Artificial Intelligence Magazine Fall 2001, 67–71

[Efs02] C. Efstratiou, A. Friday, N. Davies, and K. Cheverst. "Utilising the Event Calculus for Policy Driven Adaptation on Mobile Systems." In Proceedings of Third Int. Workshop on Policies for Distributed Systems and Networks (POLICY-2002), Monterey, CA, USA, IEEE Press, June 2002.

[Fle02] P. Flegkas et al.., “A Policy-based Quality of Service Management System for IP Diff-Serv Networks”. IEEE Network March/April 2002

[Gen86] H. J. Genrich, "Predicatenransition Nets." in Petri Nets: Central Models andtheir Properties, Brauer, Reisig, and Rozenberg. eds., LNCS 254, Springer Verlag, 1986.

[Giu99] F. Giunchiglia and P. Traverso, "Planning as Model Checking”. In Proceedings of the 5th European Conference on Planning (ECP`99). LNAI, Springer-Verlag

[Goh97] Goh, C., A Generic Approach to Policy Description in System Management, Proceedings of the 8th IFIP/IEEE International Workshop on DistributedSystems Operations and Management (DSOM’97), Sydney, Australia, 21-23 October, 1997.

[Gut07] J. Gutierrez, J.L. Melus, J. Rubio, J.Serrat. "Pricing Pervasive Services Using Policy-based Mechanisms", 2007 International Conference of the Information Resources Management Association IRMA. May 19-23, 2007, Vancouver, Canada

[Hol04] G. Holzmann. "The SPIN Model Checker: Primer and Reference Manual". A. Wesley. ISBN 0-321-22862-6. 2004

[Ibm01] IBM Autonomic Computing Manifesto. On Line, October 2001. http://www.research.ibm.com/autonomic/manifesto/

[IETFPol] IETF Policy Framework Working Group – Internet Engineering Task Force. IETF http://www.ietf.org/html.charters/policy-charter.html

[IETFRap] IETF Resource Allocation Protocol(RAP) – Internet Engineering Task Force. IETF http://www.ietf.org/html.charters/rap-charter.html

[Jho98] W. Johnston, S. Mudumbai and M. Thompson, “Authorization and Attribute Certificates for Widely Distributed Access Control,” IEEE 7th Int Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE), Stanford, CA. June, 1998, pp. 340 - 345 (see also http://www-itg.lbl.gov/security/Akenti/)

[Kar05] F. Karayannis, J. Serrat, J. Baliosian, J. Rubio-Loyola, et al.. “In-field Evaluation of a Managed IP/MPLS over WDM Provisioning Solution”. IEEE Communications Magazine. Volume 43, Issue 11, November 2005.

[Kab97] Kabanza, F., M. Barbeau, and R. St-Denis. “Planning Control Rules for Reactive Agents” Artificial Intelligence 1997, p.p. 67–113

[Kep03] Kephart, J.O. and Chess, D.M., “The Vision of Autonomic Computing”, IEEE Computer, Volume 36, Issue 1, Jan. 2003 Page(s):41 – 50

[Kep07] Kephart, J.O.; Das, R.; Achieving Self-Management via Utility Functions. IEEE Internet Computing Volume 11, Issue 1, Jan.-Feb. 2007 Page(s):40 - 48

[Lam95] van Lamsweerde, A., Darimont, R., Massonet, P., “Goal- Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learned”, *Proc. RE’95 - 2nd Int. Symp. on Requirements Engineering*, York, IEEE, 1995

[Lam99] Axel van Lamsweerde. “Goal-Oriented Requirements Analysis with KAOS”. Invited talk in the First International Workshop on Policies for Distributed Systems and Networks (POLICY) 1999, 15-17 November 1999, Bristol, United Kingdom

[Lam01] A. van Lamsweerde. “Goal-Oriented Requirements Engineering: A Guided Tour”. 5th IEEE International Symposium on Requirements Engineering, Toronto, August, 2001, pp. 249-263

[Lam04] A. van Lamsweerde. “Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice”. 12th IEEE International Requirements Engineering Conference RE’04, 2004

[Man92] Z. Manna and A. Pnueli. “The Temporal Logic of Reactive and Concurrent Systems: Specification”. Springer-Verlag, 1992

[Mea55] G.H. Mealy. A method for synthesizing sequential circuits. Bell System Technical Journal, 34(5): 1045-1079, 1955

[Mee00] H.B. Meeuwissen, H.J. Batteram, and J.L. Bakker. “The FRIENDS Platform: A Software Platform for Advanced Services and Applications”. Bell Labs Technical Journal, Vol. 5, No. 3, July-Sept. 2000, pp. 59-75

[Mil99] R. Miller and M. Shanahan. "The Event Calculus in Classical Logic - Alternative Axiomatisations." Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II. A. Kakas and F. Sadri, Springer. 2048: 452-490, 1999.

[Mil02] R. Miller and M. Shanahan, *Some alternative formulations of the Event Calculus*, in A. C. Kakas and F. Sadri (eds.): Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II. Lecture Notes in Computer Science 2408, Springer 2002, ISBN 3-540-43960-9, pages: 452-490.

[Mof93] J. Moffet and M. Sloman. “Policy Hierarchies for Distributed Systems Management”. IEEE Journal of Selected Areas of Communications, Dec 1993

[Mtnm] “Multitechnology Network Management (MTNM) Solution Suite v3.0”, TMF 513, 608 and 814, v3.0, October 2003, http: //www.tmforum.org/browse.asp?catID=1689

[Myk03] E. Mykoniati et al.., “Admission Control for Providing QoS in DiffServ IP Networks: The TEQUILA Approach”. IEEE Commun. Mag., Jan 2003

[Obj] “A Power tool to engineer your business and technical requirements”. http://www.objectiver.com

[Pon05] C. Ponsard, P. Massonet, J.F. Molderez, “Design of Fault-tolerant Systems using FAUST”, Workshop on Rigorous Engineering of Fault Tolerant Systems, Newcastle (UK), July 2005

[Ram89] P. Ramadge and W. M. Wonham, "The Control of Discrete Event Systems," IEEE Proc.. special issue on discrete event systems, vol. 77. no. 1. Jan. 1989.

[Reg05] G. Regev and A. Wegmann. “Where do Goals Come from: the Underlying Principles of Goal-oriented Requirements Engineering”. In 13th IEEE International Conference on Requirements Engineering (RE’05), 2005

[Rub05] J. Rubio-Loyola, J. Serrat, M.Charalambides, P. Flegkas, G. Pavlou, A. Lluch. “Using Linear Temporal Model Checking for Goal-oriented Policy Refinement Frameworks” Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005), Stockholm, Sweden June 6-8, 2005

[Rub06a] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G.Pavlou. “GOREMOCH: A Distributed Goal-oriented Policy Refinement Environment”. Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium NOMS (short paper) April 2006. Vancouver, Canada

[Rub06b] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G.Pavlou. “A Functional Solution for Goal-oriented Policy Refinement”. IEEE Workshop on Policies for Distributed Systems and Networks (POLICY 2006). 5-7 June 2006 The University of Western Ontario. London, Ontario CANADA

[Rub06c] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G.Pavlou. “A Methodological Approach towards the Refinement Problem in Policy-based Management Systems” IEEE Communications Magazine, Vol. 44, No. 10, IEEE, October 2006

[Ster86] L. Sterling and E. Shapiro, The Art of Prolog, The MIT Press, 1986.

[Str04] C.J. Strassner. "Policy-based Network Management, Solutions for the Next Generation". Elsevier, Morgan Kaufmann Publishers 2004. ISBN: 1-55860-859-1

[SuL05] Linying Su; Chadwick, D.W.; Basden, A.; Cunningham, J.A.; Automated decomposition of access control policies. Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, 6-8 June 2005, Stockholm, Sweden

[Thi06] Thiébaux, S., C. Gretton, J. Slaney, D. Price, and F. Kabanza. “Decision-Theoretic Planning with non-Markovian Rewards”. Journal of Artificial Intelligence Research (JAIR), Vol 25, January 2006

[Tri01] P. Trimintzios et al.., “A Management and Control Architecture for Providing IP Differentiated Services in MPLS-Based Networks”. IEEE Commun. Mag., May 2001

[Tri03] P. Trimintzios et al.., “Service-Driven Traffic Engineering for Intradomain Quality of Service Management”. IEEE Network. May/Jun 2003

[Udu07] Udupi, Yathiraj B.; Sahai, Akhil; Singhal, Sharad. A Classification-Based Approach to Policy Refinement. Hewlett Packard Technical Report. January 2007

[Val04] M. Balser, S. Bäumler, A. Knapp, W. Reif, and A. Thums. “Interactive verification of UML state machines. Inernational Conference on Formal Engineering Methods '04

[Ver00] Verma D. (2000) *Policy Based Networking* New Riders. ISBN: 1-57870-226-7 Macmillan Technical Publishing USA