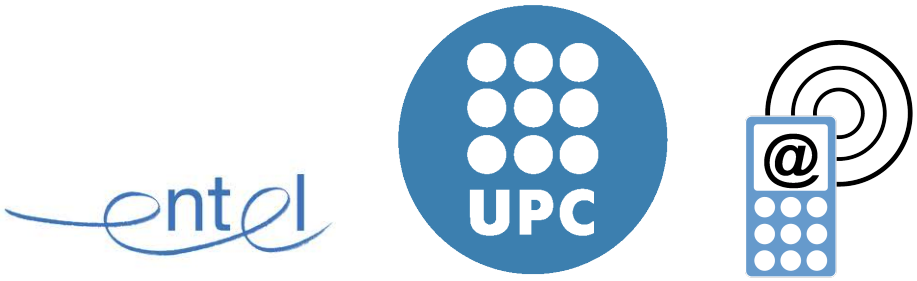
ADVERTIMENT. La consulta d'aquesta tesi queda condicionada a l'acceptació de les següents condicions d'ús: La difusió d'aquesta tesi per mitjà del servei TDX (www.tesisenxarxa.net) ha estat autoritzada pels titulars dels drets de propietat intel·lectual únicament per a usos privats emmarcats en activitats d'investigació i docència. No s'autoritza la seva reproducció amb finalitats de lucre ni la seva difusió i posada a disposició des d'un lloc aliè al servei TDX. No s'autoritza la presentació del seu contingut en una finestra o marc aliè a TDX (framing). Aquesta reserva de drets afecta tant al resum de presentació de la tesi com als seus continguts. En la utilització o cita de parts de la tesi és obligat indicar el nom de la persona autora.

ADVERTENCIA. La consulta de esta tesis queda condicionada a la aceptación de las siguientes condiciones de uso: La difusión de esta tesis por medio del servicio TDR (www.tesisenred.net) ha sido autorizada por los titulares de los derechos de propiedad intelectual únicamente para usos privados enmarcados en actividades de investigación y docencia. No se autoriza su reproducción con finalidades de lucro ni su difusión y puesta a disposición desde un sitio ajeno al servicio TDR. No se autoriza la presentación de su contenido en una ventana o marco ajeno a TDR (framing). Esta reserva de derechos afecta tanto al resumen de presentación de la tesis como a sus contenidos. En la utilización o cita de partes de la tesis es obligado indicar el nombre de la persona autora.

WARNING. On having consulted this thesis you're accepting the following use conditions: Spreading this thesis by the TDX (www.tesisenxarxa.net) service has been authorized by the titular of the intellectual property rights only for private uses placed in investigation and teaching activities. Reproduction with lucrative aims is not authorized neither its spreading and availability from a site foreign to the TDX service. Introducing its content in a window or frame foreign to the TDX service is not authorized (framing). This rights affect to the presentation summary of the thesis as well as to its contents. In the using or citation of parts of the thesis it's obliged to indicate the name of the author

Contributions to Presence-Based Systems for Deploying Ubiquitous Communication Services



Victoria Beltran Martinez

Wireless Network Group, Department of Telematics

Technical University of Catalonia

Advisor: PhD Josep Paradells Aspas

Co-advisor: PhD Henning Schulzrinne

A thesis submitted for the degree of

PhilosophiæDoctor (PhD)

2011 December

10

Context-Aware Rule-Based Service Composition Platform: Sense Everything, Control Everything

There is a current lack of service cooperation and automation that restrains users from exploiting telecommunication technologies during their daily routine, as described in Section 2.8. To date, there is no easy way to create new services which integrate location, presence, calendar, address book, IM, SMS, calls, email, Facebook and Twitter. Networked sensors and actuators for lights, temperature, humidity, smoke, and motion are also becoming popular both in residential and commercial environments. Some platforms for creating user-defined services have been proposed, as described in Section 2.8. These platforms are however only partial solutions (e.g., call handling) and does not offer easy-to-use and intuitive interfaces or languages for general users. Furthermore, the Semantic Web is working towards fully automation of service composition and invocation. Although the technologies that support such automation are mature, the Semantic Web is still lacking of pioneer applications that attract large populations of users and therefore motivate web services to move towards automation. With these issues in mind, we have developed Sense Everything Control Everything (SECE), a platform for context-aware service composition based on user-defined rules. The contributions of SECE are two-fold: a user-friendly rule language and the design

and implementation of a context-aware service composition framework.

SECE takes actions automatically on behalf of the users depending on the monitored information and triggered events. In order to build such a system, the user has to define event-action rules. There are several ways to allow users to define these rules such as using XML, forms or scripts. We choose to develop SECE using a natural-English-like formal language because it is more powerful and easy-to-use than XML and form-based solutions. Below, an example script that turns the home lights on every sunset shows the end-user friendliness of SECE.

```
every sunset {
   homelights on;
}
```

Section 10.1 outlines the main benefits of SECE. The SECE language is described in Section 10.2. Section 10.3 discusses the architecture of SECE. Section 10.4 presents enhancements of SECE towards the Semantic Web. Lastly, some conclusions are given in Section 10.5.

10.1 Overview

SECE differs from other rule-based systems in that it provides an interface for creating rules in natural English-like-language commands. The main drawback of rule-based systems is that the rule languages involve complex formulaic or XML descriptions. Lay people are not as inclined to use these systems as the learning curve for these languages may be steep. Thus, we have defined a formal rule language which resembles English. With a simplified English-like interface to creating rules, users will be more prone to incorporate rule-based systems into their lives, making context-aware computing a seamless part of everyday life. Figure 10.1 compares SECE to some proposed solutions for user-created services described in Section 2.8. The second column indicates the user language for defining events and conditions that trigger action scripts. The third column indicates the language for action scripts. The fourth column shows the kinds of communication services that the users can use. The following columns show the types of information handled by the systems. It can be seen that the proposed solutions tackle partial domains. CPL [152], LESS [47], SPL [153], VisuCom [154] and DiaSpec [155] are all limited to controlling call routing. CPL and LESS use XML, and hence even

Systems	User rules	User actions	Communications	Time	Location	Presence	Sensors	Web services	Actuators
SECE	**Natural-language-like rules**	**Tcl scripts**	**Call, email, IM**	✔	**User & buddies**	**Rich**	✔	✔	✔
CPL	XML tree	Fixed XML actions	Call	✖	✖	✖	✖	✖	✖
LESS	XML tree	XML actions	Call	✔	✖	Basic	✖	✖	X10, vcr
SPL	script	Signaling actions	Call	✖	✖	✖	✖	✖	✖
VisuCom	Graphical UI	Signaling actions	Call	✖	✖	✖	✖	✖	✖
CybreMinder	Form based	Reminder	✖	✔	✔	✖	✔	✖	✖
Task.fm	Time rule	Reminder	✖	✔	✖	✖	✖	✖	✖
DiaSpec	Java	Java	✔✖	✖✔	✖✔	✖✔	✖✔	✖✔	✖✔

Figure 10.1: Comparison to related work

simple services require long programs. Moreover, XML-based languages are difficult to read and write for non-technical end-users. DiaSpec is very low level and therefore only suitable for advanced developers. VisuCom allows users to create services visually via GUI components. CybreMinder [159] is a context-aware tool which allows users to setup email, SMS, print out and on-screen reminders. This tool is not as powerful as scripting-based systems due to its form-based nature. Task.fm [160] is a similar SMS and email remainder system that uses natural language to describe time instants when email or SMS reminders will be sent. This tool only supports time-based rules. It can be seen that only SECE considers rich presence information about the user and his or her buddies.

SECE provides a platform for context-aware service composition for a number of services, such as, presence, telecommunication, sensors and location-aware services. Users can create compositions of services by formulating simple rules. The rules trigger event-based service composition depending on the user's context, such as her location, time, and communication requests. Traditional rule-based systems are mostly designed to handle a single service domain. SECE, on the other hand, interacts with a few service domains. SECE connects services that until now were isolated, leading to new, more useful and user-personalized, composite services. These services do not require user interaction; they are automated and embedded into users' life. SECE converges fixed and mobile services by integrating the Internet, cellular and sensor networks. This integration requires interacting with Internet servers, web services, home gateways, and wireless and fixed user devices.

	GContact	GCalendar	GVoice	Twitter	PS	SER	GMail	GMaps	Flickr
RULE TYPES									
Time		Optional							
Calendar		Required							
Context					Required				
Communication	Optional		SMS, voicemail			SIP call, IM	email		
Location								Required	
ACTIONS									
email	Optional						Optional		
tweet				Required					
flickr									Required
sms	Optional		Required						
call	Optional		phone number			SIP address			
status					Required				
forward	Optional					Required			
schedule		Required							
homelights					Required				

Figure 10.2: Third-party services of SECE rules and some actions

10.2 The SECE Language

The SECE language provides five types of rules: time, calendar, location, context and communication. As a formal language, it states the valid combinations of keywords and variables for each kind of event, and provides a set of commands such as "sms", "email", "tweet" or "call". SECE rules interact with different third-party services based on their actions and subscribed events. Thus, SECE users need to learn about the services needed by the rule types and actions that they wish to use for configuring their SECE accounts for such services. SECE will provide online documentation about each rule's and action's syntax and required services. This documentation will also contain example rules to help users build rules for specific events and goals, and get familiarized with SECE rules. Figure 10.2 summarizes the required and optional services for the SECE rules and some actions.

Any SECE rule has the structure "*event* { *actions* }". *Event* defines the conditions that need to be satisfied to execute the *actions* that are delimited by braces. The SECE language for describing events is a formal language similar to English that has been designed to be easy to use and remember by end-users. This language is generated by an ANTLR grammar [213]. We use the Tcl language [214] as the syntax for the rule actions. This choice is due to Tcl extensibility that allows adding new commands to its core with relative ease. Tcl provides a command that receives the name, arguments

Command	Description	Parameters	Example
accept	accepts a SIP call	none	accept
reject	rejects a SIP call	none	reject
call	sends a SIP or phone call request	destination (from origin)	call *0034687137921* from *1646428479*
email	sends an email	destination subject body (from origin)	email *azi40@gmail.com "hi" "do not forget the milk!"*
event	returns a calendar event's info	title, description, location, duration, participants, start, end	event *title*
facebook	post a message on FB	message	facebook *"Hi all!"*
incoming	returns a request's info	origin, destination, content	incoming *origin*
my	returns the user's info	shortcut	my *activity*
status	change the user's info	shortcut value	status *activity busy*
schedule	publish a calendar event	name	schedule *"meeting finished"*

Table 10.1: Some SECE actions (optional parameters in parenthesis)

and code of a new command as parameters, constructs the corresponding Tcl command and incorporates it into the Tcl interpreter. Thus, SECE users can describe events in a user-friendly and natural way while taking advantage of the expressive power of Tcl to define actions. Moreover, Tcl syntax is simple if no complex control statements and structures are considered, which can be seen in the rule examples given by the following subsections. SECE provides a set of new Tcl commands, such as "sms", "email", "tweet" or "call". Some commands are specific for particular events as for example the "accept" and "reject" commands can only be used in request-based rules. Table 10.1 shows some Tcl actions provided by SECE. We may add support for other scripting languages like Ruby [215] or Python [216] in the future. However, a promising although challenging future step would be to extend the SECE language to define rule actions.

SECE tries to make it easy to integrate external knowledge seamlessly without having to explicitly invoke libraries or functions. User information can be accessed by natural expressions such as "my mobile" and "John's address" and their equivalent shortcuts "me.mobile" and "John.address". SECE makes it easy to automate actions

since the context about any rule event can be accessed by means of commands such as *incoming* and *event* that retrieve information from communication and calendar events, respectively. Address books, IM/presence names and Internet calendars are put together for creating a single view of the user's world. Convenient daily times such as sunset, sunrise and dawn, as well as landmarks names such as "Columbia University" facilitate rule writing. The following subsections describe the types of SECE rules and their involved services. In order to clearly display the structure of the rule and action language, the variables that are set by the user are highlighted in bold. Appendix H shows the structure of the SECE language's BNF-based grammar. Moreover, Section 10.2.6 addresses some issues about error detection and event handling.

10.2.1 Time-Based Rules

Time-based rules define single and recurring events, which start with the *on* and *every* keyword, respectively. The iCalendar specification [217] covers single and recurring events but it is designed to be processed by computers rather than users. We designed the SECE's time sublanguage to be easy-to-write while maintaining the full expressive power of the iCalendar specification. Figure 10.3 shows an example SECE recurrent event and its equivalent iCalendar definition. As SECE time events are fully iCalendar-compliant, any time-based rule can be exported to other iCalendar-compliant calendars, and SECE can import external calendars' events. Appendix G describes the mapping between iCalendar and SECE expressions. Sections 10.2.1.1 and 10.2.1.2 describe time-based rules about single events and recurrences. As well, special considerations about some particular types of recurrences are given in Section 10.2.1.3.

10.2.1.1 Single-Event rules

A single-event rule defines a particular date at which the rule has to be executed. Figure 10.4 deptics the structure of this kind of rule. The *dateExpr* and *timeExpr* parameters are date and time expressions, respectively, which are described below. The *timezone* parameter is the identifier of the time zone of reference as for example "America/Los_Angeles". A city name or a user's location may be indicated, rather than a complete time zone name. In this case, SECE needs to deduce the corresponding time zone. If a time zone is not indicated, the default local time zone for the user who created

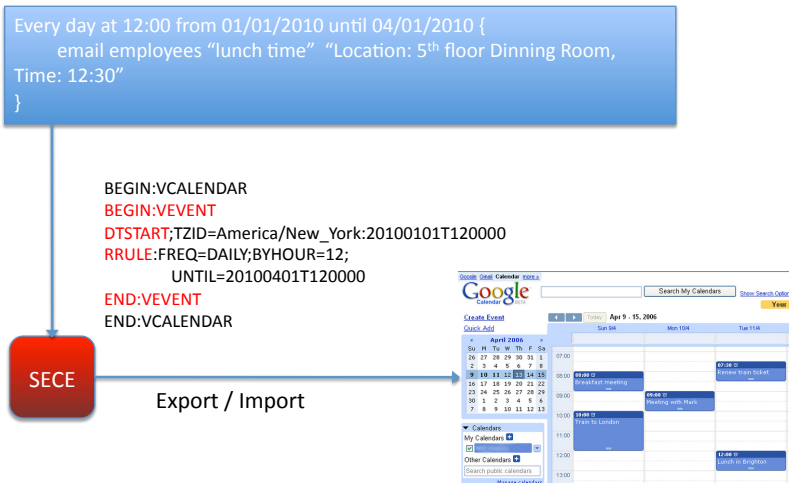


Figure 10.3: iCalendar-SECE mapping

Figure 10.4: Sketch for single-event rules (optional parameters in parenthesis)

the rule is assumed. Below, some example rules are shown. Figure 10.5 shows the main grammar rule for single-event rules.

on **Anne's birthday, 2010** at **12:00** in **Europe/Zurich** {
  sms **Anne "Happy Birthday!!!kisses. John"**;
}
on **July 16, 2011** at **10:00 am** in **bob.location** {
  call **bob**;
}
on **today** at **sunset** { turn **office.lights** on }

Date expressions A date expression is a particular day in a year that can be expressed in multiple formats. Figure 10.6 shows the grammar rule for date expressions. In addition to the common formats *yyyy-mm-dd* and *mm/dd/yyyy*, such as 2012-12-20 and 12/20/2010, other flexible formats are allowed:

- *A day in a year*: It is an ordinal number followed by the word *day*. A minus sign means that the number of day is counted from the last day of the year. In addition, textual ordinal numbers are allowed such as *first*, *second* and so on. Some examples are "last day, 2011", "-5th day, 2012" and "45th day, 2012".

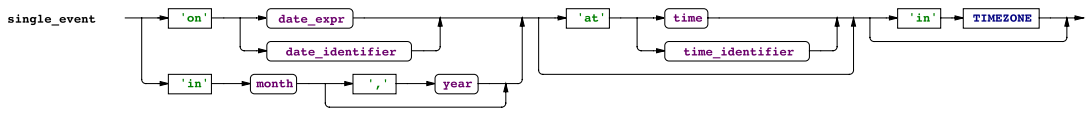


Figure 10.5: Grammar rule for single-event rules

- *A day in a month*: It can be expressed as an ordinal number like "25th day of February, 2012" or a cardinal number like "February 25, 2012". Months abbreviations are given by the first three letters of the month such as "Feb" and "Jan". A minus sign indicates that the day is counted from the last day in the month and textual ordinal numbers can be used instead of "*th*". For example, "-1th day of April, 2012" is equivalent to "on last day of April, 2012" and "on April -1, 2012".

- *A weekday in a month*: A weekday can be written as its name such as "Sunday" or its abbreviation, which is the first two letters of its name, such as "SU". A particular weekday in a month is expressed as a cardinal or ordinal number such as "2th TU of December, 2011", "2 TU of December, 2011", "December 2 TU" and "December 2th TU". A minus sing indicates that the weekday number is counted from the last day of the month as in "-1 MO of January, 2012". Textual ordinal numbers can be used instead of "*th*" such as "Aug first SU, 2012".

- *A weekday in a particular week*: It is expressed as the weekday followed by "in" and the week number such as "MO in 38th week, 2012". A minus sign is allowed to count the week number from the last day of the year. For instance, "WE in -1th week, 2012" means the last Wednesday in 2012.

In addition to the previous kinds of date, users can indicate special dates. A special date is a well-known date, such as "Halloween, 2013" or "Thanksgiving day, 2011", or a user-defined event, such as "my birthday, 2012", "bob's speech, 2011" or "interview with Tom, 2012". Users need to define these dates in their calendars. For example, "my birthday" is defined in one of the user's calendars for 2012. Moreover, SECE uses a database of well-known dates in the case the date is not found in the user's calendars. The special value "today" is an exception that means the current date and is not therefore defined in any calendar. Special dates can be combined with the *after*

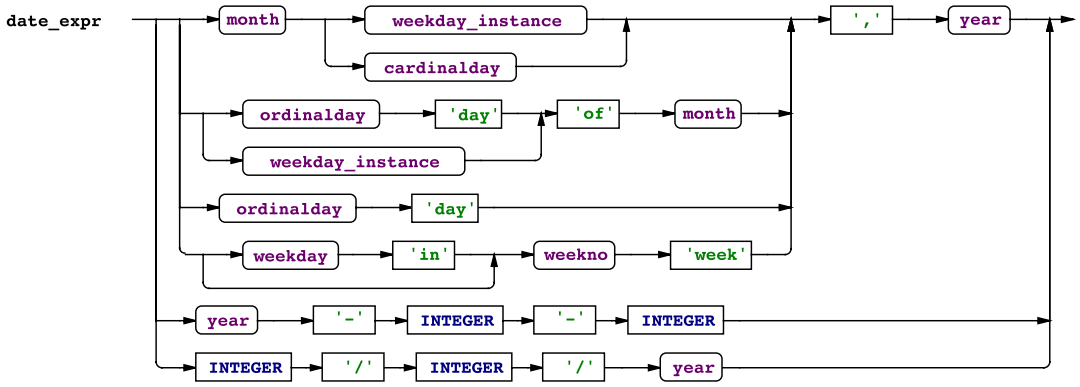


Figure 10.6: Grammar rule for date expressions

and *before* operators for referring to future and past dates, respectively, as described below.

Future and past time points The *after* and *before* operators refer a future and past point in time, respectively, from a given special date (e.g., "Christmas day") or time (e.g., "sunset"). A future time is pointed by adding an amount of time to the given time like "2 days after Christmas day" and a past time by subtracting that time from the given time like "1 month before Christmas day". If the given expression is a special date, the time that is added or subtracted can be expressed as days, weeks, months or years. If the given time is a special time, a number of hours, minutes or seconds can be used. The *after* and *before* operators are exchangeable with the plus (+) and minus (-) signs, respectively. Some examples are "1 day before bob's birthday, 2011", "Thanksgiving day, 2012 + 1 month", "1 day after annual meeting, 2012", "30 minutes before midnight" and "sunrise - 1 hour".

Time expressions A time expression determines the time at which a rule is executed. Times has the format *hh:mm:ss*, and *am* (ante meridiem) and *pm* (post miridiem) can be indicated. The number of seconds (*ss*), is optional. Some examples are "9:00", "4:34:20 am" and "1:00:08 pm". Moreover, there are special times in every day that can be referred to in SECE rules: "sunrise", "sunset", "evening twilight", "morning twilight", "afternoon" and "midnight". User-defined special times are also allowed, such as "lunch break", "TV news" and "first working hour". A special time can be a

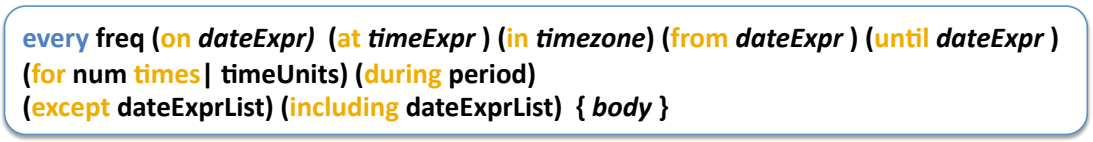


Figure 10.7: Sketch for recurrent-event rules (optional parameters in parenthesis)

reference point for past or future times by means of the *before* and *after* operators, as described above.

10.2.1.2 Recurrent-Event Rules

A recurrent rule is a recurrence that determines a set of repetitive events over time and has the general form depicted by Figure 10.7. A recurrent rule always starts with the keyword *every*, followed by the type of frequency: second for secondly, minute for minutely, hour for hourly, day for daily, week for weekly, month for monthly and year for yearly (e.g., every year, every second, every day, etc). It is possible to indicate at which intervals a recurrence repeats as for example "every 2 years". If the frequency interval is not present, the unit is assumed. All the conditions of SECE recurrences are optional, which are in parenthesis in Figure 10.7. Some simple example rules are shown below:

```
every day at last working hour except August {
    backup;
}
every day at sunset { turn home.lights on; }
every last monthly day {
    email me "Reminder: Check the students' monthly report";
    tweet "one more month is finished.";
}
every week on WE at 6:00 PM from 1/1/10 until May 10, 2010 except 3th WE of Feb
including first day of June, 2010 {
    email irt-list "reminder: weekly meeting today at 6:00 PM";
}
```

A rule's recurrence pattern is mainly defined by the frequency (i.e, yearly, monthly, weekly and so on), and optionally by the date expression in its *on* condition. An *on* condition describes when the occurrences within the recurrence take place, as for example "every week on WE". A recurrence will repeat indefinitely if no *until*, *during*, or *for* conditions are indicated. If the recurrence's frequency is bound by the date

expression in the *on* part, the frequency limits the number of occurrences in the recurrence, as in "every week on January". On the contrary, if the date expression in the *on* part is bound by the frequency, the frequency expands the number of occurrences, as in "every year on January". When the frequency interval is the unit, the *on* term is present and the frequency expands the recurrence, the frequency can be omitted. For example, "every week on MO" is equivalent to "every MO" while, in "every week on January", the frequency can not be omitted because it determines the recurrence. The *from*, *until*, *for* and *during* conditions do not define the recurrence pattern but delimit it. *Except* and *including* conditions do not affect the recurrence pattern either. The former removes occurrences from the recurrence set and the latter includes occurrence in the set. Recurrent-event rules provide a powerful flexibility and expressiveness that allows users to define a wide variety of recurrence patterns. End users are therefore responsible of expressing semantically correct recurrences. As instance, the rule "every Thanksgiving Day from August, 1, 2011 until September, 1, 2011" does not make sense because no Thanksgiving Day occurs between the specified date range. Table 10.2 shows the meaningful combinations of the frequency types and date expressions in the *on* condition of recurrent-event rules. Below we describe the possible conditions of recurrent events and some example time events. Lastly, some considerations about omitted information in recurrent rules are given.

***On* condition** This term describes when the occurrences within the recurrence occur. An *on* condition is a date expression similar to that described in Section 10.2.1.1 but can indicate multiple dates as described below. A date in an *on* condition can indicate a year as long as the kind of frequency is not yearly.

- *A set of months*: A list of months such as "every day on Jan, Feb, Mar". In this case, the preposition "in" can be used instead of "on" for the sake of readability, as in "every day in Jan, Feb, Mar".

- *A set of days in a set of months*: A list of ordinal or cardinal day numbers before a list of months, such as "every year on 23th, 24th, 25th days of Jan, Feb, Mar" and "every year on Jan, Feb, Mar 23,24,25".

- *A set of week days in a set of months*: A set of weekdays before a list of months, such as "every MO, TU, WE of Apr, May" and "every year on Apr, May SU".

on CONDITION	YEARLY	MONTHLY	WEEKLY	DAILY
A day	*every year on 20th day*	*every month on 20th day*		
A yearly day	*every 20th yearly day*			
A monthly day		*every 20th monthly day*		
A month	*every year on Jan*		*every week on Jan*	*every day on Jan*
A day in a month	*every Jan 21*			
A weekday in a month	*every 2th MO of Jan*			
Weekdays in a month	*every Jan MO* (four MO days)			
Any weekday			*every week on MO*	
A particular weekday	*every year on 1th SA*	*every month on 1th SA*		
A particular week	*every year on 50th week*			
A weekday in a particular week	*every year on MO in 50th week*			
A day that is a weekday		*every month on 13th day and FR*		

Table 10.2: Date expressions and frequency types

A particular weekday can be determined by an ordinal or cardinal number as in "every year on January 2 Monday" and "every year on 4th SU of Feb".

- *A set of days that are some particular weekdays*: A list of ordinal day numbers followed by the keyword "and" and a list of weekdays, such as "every month on 13th day and Friday". These expressions only have sense within monthly recurrences, and hence if no recurrence type is specified, a monthly one is assumed.

- *A set of weekdays in a set of weeks*: A list of weekdays followed by the keyword "in" and a list of ordinal week numbers as in "every year on Saturday, Sunday in 20th, 50th weeks".

- *A set of weekdays in every month or year*: Time events at some weekdays can be expressed by means of a weekly recurrence such as "every week on WE, TH",

which generates an event every Wednesday and Thursday. A particular occurrence of weekday in every month or year can be determined by an ordinal or cardinal number such as "every month on 1th Monday" and "every year on 8th SU". Since weekdays are more commonly bound to months, a monthly frequency is assumed if the kind of recurrence is not specified. As instance, "every month on 2th MO" means the second Monday in every month, which can be rewritten as "every 2 MO". The second Monday of every year can only be written by specifying the yearly recurrence as in "every year on 2th MO".

- *A set of days in every month or year*: A list of ordinal day numbers before the keywords "yearly days" and "monthly days" for yearly and monthly recurrences, respectively, as in "1th, 365th yearly days" and "20th, 25th monthly days". If "monthly" or "yearly" is not specified, the day is bound to every month or year based on the kind of recurrence in the *every* term (e.g., "every year on 30th day"). If no kind of recurrence is indicated, a yearly recurrence is assumed (e.g., "every 30th day"). The keywords "yearly" and "monthly" are thought to help differentiating between a day in every month or every year when the type of recurrence is not indicated. For example, the expression "every month on 30th day" can be expressed as "every 30th monthly day". Likewise, the expression "every year on first day" is equivalent to "every first yearly day".This is also helpful in hourly, minutely and secondly recurrences because it makes it possible to write rules like "every hour on last monthly day".

***From* condition** This term specifies the time from which the rule gets active in an inclusive manner. This time is a date expression, which is described in Section 10.2.1.1. Users can indicate only a year rather than a complete date, as in "every month from 2012". A *from* term can include a time after the date expression as in "from Feb 10, 2010, 9:00". This time determines the time at which the recurrence starts and therefore the time at which all the occurrences occur. In the case that an *At* term specifies a different time, this time always prevails over that in the *from* term. Moreover, *from* terms can specify any of the special values described in Section 10.2.1.1, such as sunset and sunrise. *From* conditions are incompatible with *during* conditions, which are described below. Some example events with *from* terms are "every month on 14th day from Jan, 1, 2012" and "every MO from March 1, 2012, 11:00 am"

***Until* condition** This condition indicates when the recurrence finishes in an inclusive manner. This is a date expression, which is described in Section 10.2.1.1. Users can also specify a year alone, as in "every month from May, 14, 2011 until 2012". A time can also be specified as for example "until my birthday, 2014, 11:00", which is is only useful for hourly, minutely and secondly recurrences. *Until* conditions can specify any of the special values described in Section 10.2.1.1 such as Christmas Day or sunrise, and are incompatible with *for* and *during* conditions. Two example events are: "every SA, SU from April,1, 2012 until Aug, 1, 2012" and "every hour from today, 19:00 until today, 21:00"

For condition This condition limits the number of occurrences in a recurrence by means of a maximum number of instances or amount of time. The first case uses the keyword "times" and means "repeat until n occurrences are done", as in "every day for 2 times". The second case uses a time unit, which can be seconds, minutes, hours, days, months or years, and means "repeat during n time-units", as in "every week on FR from Jun, 1, 2012 for 2 months". The *for* condition is incompatible with the *until* term.

During condition This condition bounds the recurrence by a period of time defined by the user. A period of time has a start date and an end date. Using a during condition in a rule is equivalent to combining a *from* term set as the condition's start date and an *until* term set as the condition's end time. Thus, *during* conditions can not coexist with *from* and *until* conditions. An example is "every day at 23:00 during summer".

In condition The *in* condition allows indicating a time zone different of the local one. A user's location or a city name can be used instead of a time zone identifier, which involves SECE deducing corresponding time zone. E.g.: "every month on last day at 9:00 in alex.location".

Except and Including conditions The *except* and *including* conditions are lists of date expressions that are taken out of and added to the final set of occurrences, respectively. Section 10.2.1.1 discusses the format of date expressions. Multiple *except* and *including* terms can coexist in a particular rule in an additive manner (i.e., "every

day except today except Halloween, 2011" is equivalent to "every day except today, Halloween, 2011"). An *except* condition takes precedence over any *including* condition. This means that if a particular date is included in both an *except* and an *including* condition, the date is removed from the recurrence set. Two examples of these conditions are "every day in November, 2011 except Thanksgiving day,2011" and "every month until December, 2011 including Jan 15, 2012, Jan 11, 2012". Date expressions in *except* and *including* terms can specify a time. In the case of an *including* term, a date with a time is simply added to the recurrence set regardles whether or not that time is equal to that in the recurrence set that is given by the *from* and *at* conditions. However, if a time is specified in an *except* term, that time should be equal to that of the recurrence set. Otherwise, the mismatched *except* term is ignored and therefore has no effect on the final set of occurrences. As instance, the *except* term in the rule "every day at 10:30 except Sept, 2011, 12:00" has no effect. If no time is specified, the time of the added or removed occurrences is assumed to be the time of the recurrence set. For example, the rule "every day at 12:00 except September 22, 2010" takes out the event "September 22, 2010, 12:00" of the recurrence set.

When a particular month is specified in an *except* or *including* condition without a particular day, the set of occurrences that are taken out or added depends on the particular recurrence. This is the set that matches the rule's recurrence pattern for the specified month. For example, "every week except June" takes out the four events that corresponds to the weeks in June while "every day except June" takes out 30 events, one for each day in June. The *on* condition affects the set of occurrences that are derived from an *except* or *including* condition. For example, in the rule "every week on Saturday, Friday until February, 2012 including April", the *including* term adds eight occurrences, which are all the Saturdays and Fridays in April, to the final set of occurrences.

A rule's occurrences can be set to occur at particular hours, minutes or seconds by defining *at* terms or by hourly, minutely or secondly frequencies. For example, "every MO at 12 hours at 20, 40 min" generates one event at 12:20 and other event at 12:40 every Monday, and "every 2 hours on 30th monthly day" generates events at 9:00, 11:00, 13:00, etc., every last day of month. In these cases, if an *except* or *including* date does not specify a time, this date is split into multiples occurrences that match the rule's recurrence pattern. As instance, "every MO at 12 hours at 20,40 min except first

MO of April,2012" takes out the two events that occur on the first MO of April, 2012, at 12:20 and at 12:40. The example "every 2 hours on 30th monthly day including 3th day of April, 2011" includes the events that occur every 2 hours on April 3, 2011, into the set of occurrences. As mentioned above, if an *except* date includes a time, it must match the rule's time pattern. Otherwise, that date is not taken out of the recurrence set. On the contrary, any time can be specified in an including condition because it is an addition. For instance, the rule "every MO at 12 hours at 20, 40 min including first MO of April, 2011, 11:00" includes the specified event that occurs at 11:00.

At condition This condition specifies the time at which the occurrences happen in a recurrence (e.g., "every TU at 10:00 am"). Special times can be specified such as sunrise, as described in Section 10.2.1.1. *At* conditions allow defining a set of occurrences by nesting hours, minutes and seconds. This means that recurrence instances can be set to occur at particular hours in a day such as "at 8, 9, 10 hours", at particular minutes into an hour like "at 20, 45 minutes" and at particular seconds into a minute "at 10, 20 seconds". For example "every day at 8, 9 hours at 30 minutes at 15, 40 sec" generates occurrences at the times 8:30:15 , 8:30:40, 9:30:15 and 9:30:40. Nested *at* terms expands the rule recurrence since they split every occurrence in the recurrence into multiple times. When an *at* condition does not include any textual time unit (e.g., "hours"), this means the time at which all the occurrences occur and no more *at* conditions are permitted.

Default information in recurrences Any occurrence within a recurrence is uniquely defined by a year, month, day and time. If a rule does not contain an *on* condition, the set of occurrences is determined by the *from* condition. This term specifies the first occurrence in the set, which will repeat at the given frequency. As instance, "every year from September 1, 2011" generates a recurrence that repeats the first day of September every year. If neither an *on* nor a *from* condition is specified (e.g., "every week"), only the frequency determines the recurrence. In this case, the first day of the frequency type (i.e., every year, month or week) is assumed. If a rule does contain an *on* condition but lacks the particular day (e.g., "every year on April"), a default value defined by the user is assumed or, otherwise, the first day that matches the recurrence is assumed (e.g., first day of April).

The time at which each occurrence occurs can be specified by means of the *at* and *from* conditions. For instance, the rules "every day at 12:00 from 12th day of September,2011" and "every day from 12th day of September, 2011, 12:00" are equivalent. If a rule contains an *at* condition, it should not include a time in a *from* condition. The time in *at* conditions always prevails over that in *from* conditions. If no time is specified in an *at* or a *from* condition, a default time defined by the user is assumed. Nested *at* conditions can be used to split the occurrences into different times, as described previously. In this case, the time at which the occurrences occur is the result of combining the *at* conditions and the time in the *from* term. For example, the rule "every day at 20, 30 minutes from March 1, 2011, 11:00" generates two events at 11:20 and 11:30 every day.

10.2.1.3 Hourly, Minutely, and Secondly Recurrent-Event Rules

In addition to the description of recurrent-event rules given in Section 10.2.1.2, this section gives special considerations about hourly, minutely and secondly recurrences. SECE distinguishes two types of hourly, minutely and secondly recurrence: continuous and discontinuous recurrences. Continuous recurrences start and end at the times specified by the *from* and *until* conditions, respectively. The period during which the recurrence lasts is continuous and the recurrence is not restarted when a new day begins. This type of recurrence follows the iCalendar definition of hourly, minutely and secondly recurrences and its rule structure is depicted in Figure 10.7. For example, the rule "every 3 hours from January 15, 2011, 22:00 until January 16, 2011, 8:00" generates the events "'January 15, 2011, 22:00", "January 16, 2011, 1:00", "January 16, 2011, 4:00" and "January 16, 2011, 7:00". A discontinuous recurrence also starts and ends at the times indicated by the *from* and *until* terms, respectively. This kind of recurrence however is only active in a given time range. SECE provides a flexible way to define discontinuous recurrences. There are two ways of defining the time range during which a recurrence gets active: the *from/to* and *in* condition. The *in* condition is an identifier that represent a start and end time defined by the user. For example, the rule "every hour in working hours from October 1, 2011 until June 1, 2012" generates an event every hour from 9:00 to 17:00 in every day, assumed that "working hours" is the time range [9:00, 17:00]. The *from/to*' condition indicates the start and end times explicitly rather than a predefined variable. For example, the afore-mentioned rule can

at CONDITION	HOURLY	MINUTELY	SECONDLY
Particular hours	boundary (*every hour at 10, 11 hours*)	boundary (*every minute at 10, 11 hours*)	boundary (*every second at 10, 11 hours*)
Particular minutes	multiplier (*every hour at 30, 45 mins*)	boundary (*every minute at 80, 45 mins*)	boundary(*every second at 80, 45 mins*)
Particular seconds	multiplier (*every hour at 300 sec*)	multiplier (*every minute at 50, 40 secs*)	boundary (*every second at 300 sec*)

Table 10.3: Effect of *at* conditions on hourly, minutely, and secondly recurrences

be rewritten as "every hour from 9:00 to 17:00 from October 1, 2011 until June 1, 2012". Time ranges can be defined with any of the special times described in Section 10.2.1.1 as for example "sunrise". If an *on* term is not specified in a discontinuous recurrence, the recurrence repeats every day. If an *on* term is included, the recurrence is restarted on the specified days. Thus, a discontinuous recurrence is an hourly, minutely or secondly recurrence that repeats every day or the days determined by an *on* condition.

If an hourly, minutely or secondly recurrence does not include a *from* condition, the current date is taken as start point for the recurrence. The time of the occurrences is based on the type of recurrence. In the recurrence is continuous and the *from* condition does not contain a time, the first hour in the day (00:00:00) is assumed. If the *until* condition does not contain a time, the latest time in the day (23:59:59) is assumed. If the recurrence is discontinuous, the time at which the first and last occurrences occur is determined by the *from/to* or *in* condition, as described above. If a discontinuous recurrence contains a *from* or *until* condition with a time, this time should be according to the time range during which the recurrence is active. Otherwise, the time is ignored. Nested *at* conditions, which were described in Section 10.2.1.2, can be used in hourly, minutely and secondly recurrences for expanding or limiting the recurrence. Table 10.3 outlines the effect of nested *at* conditions on these types of recurrences.

10.2.2 Calendar-Based Rules

These rules specify events that are defined in the user's Google Calendar (GCalendar) and always start with the keyword *when*. Thus, the user needs to configure his GCalendar in his SECE account before entering rules of this kind. Figure 10.8 depicts the syntax of calendar-based rules.

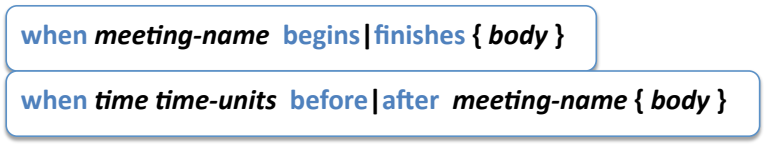


Figure 10.8: Sketch for calendar-based rules

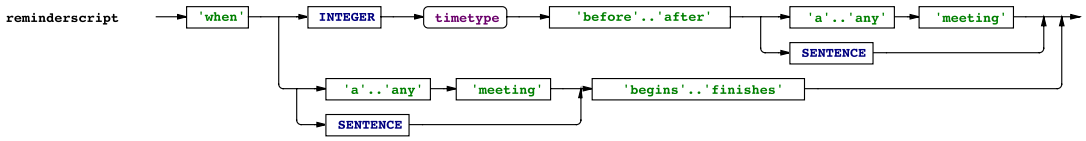


Figure 10.9: Grammar rule for calendar-based rules

These rules can be triggered some time before or after calendar events occur, as well as when these events begin or finish. These rules may be useful to create user-personalized reminders, as the first example below, but also for other services, as the second example. When a calendar-based rule is entered, SECE checks out that the event exists in any of the user's calendars (e.g., "weekly meeting" in the examples below). Otherwise an error message is displayed and the rule is not created. SECE determines when the rule should be triggered based on the rule's conditions and the event's starting and end times. Calendar-based rules accept the text "any meeting" instead of an event's name for referring to any event in the user's calendars. Figure 10.9 shows the main grammar expression for the Calendar-based rules' syntax. Some example rules are given below:

```
when 30 minutes before "weekly meeting" {
      email [event participants] "The weekly meeting will start in 30 minutes";
      if {![me within 3 miles of campus] } {
         email [status bob.email] "I'm away" "Please, head the conference room and prepare
         everything for the weekly meeting. Not sure if I will be on time.";
      }
}
when "weekly meeting" begins {
      status activity busy;
      sms [event participants] "Please, switch your cell phone off or set silent mode";
}
```

Figure 10.10: Sketch for location rules

10.2.3 Location-Based Rules

A location rule starts with the keyword *me*, if it is about the user that is entering the rule, or an identifier of one of her friends such as a nickname, email or SIP address. Figure 10.10 shows the general structure of these rules. Five types of location information are supported: geospatial coordinates (longitude/latitude), civic information (street addresses), well-known places, user-specified places and user locations. Well-known places are unique and widely-known landmarks such as "Columbia University" or "Rockefeller Center". User-specific locations are places that are of interest for the local user and therefore are defined by the user in the system, such as "office", "home" and "school". The system resolves these constants via the user's address book. SECE interacts with the Google Maps (GMaps) web service [218] for performing geocoding (i.e., obtaining geographic coordinates) of civic information and well-known places. The GMaps service returns more than one solution when the user's input location is inaccurate. In this case, SECE will request the user to select one of the possible locations before creating the rule. In the future, SECE will guess the best possible candidate locations based on the user's position. User position could be given via two means: the PS and the Google Latitude (GLatitude) service [219]. The former requires the user to use a mobile presence application that publishes his location information such as mobile instant messengers (see Section 2.1), and configure this application with the SECE PS (see the SECE architecture in Section 10.3). The latter requires the user to have installed GLatitude [220] on his mobile device. This application determines the user device's location through Wi-Fi, 2G/3G/4G mobile or GPS satellite signals.

Different geospatial operators can be used in location-based rules, such as *near*, *within*, *in*, *outside of* or *moved*. Below, a location rule using the *near* operator is shown.

Bob *near* **"Columbia University"** {
    if{ [my **status** is **idle**] } { call **bob**; }
}

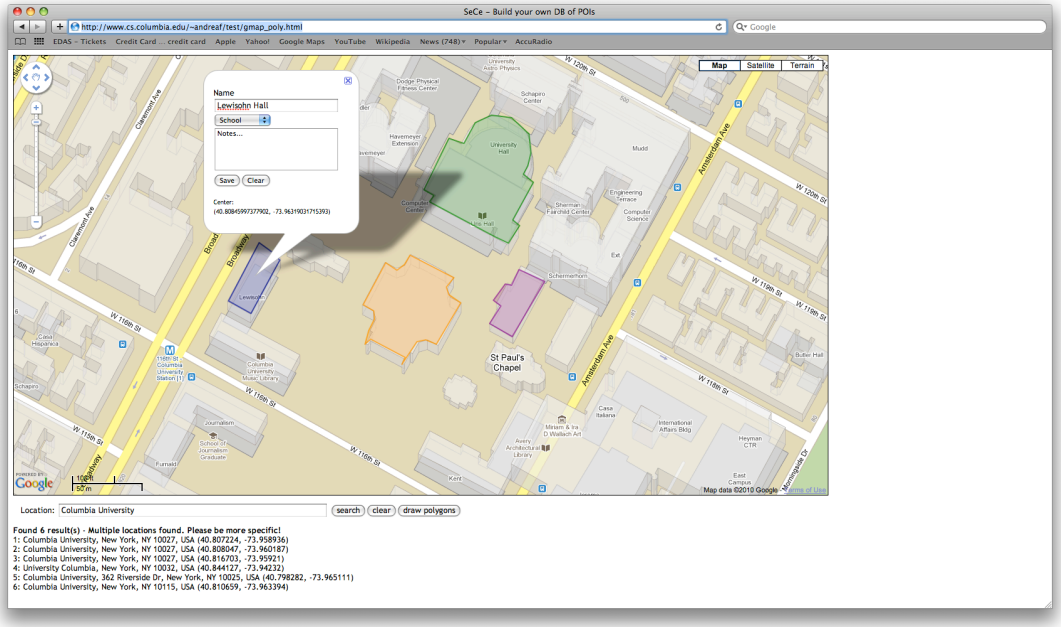


Figure 10.11: SECE geographical database GUI

The *Within* operator means that the user is within a radius of the given location. The *Near* operator means the same but the radius is a default distance that the user defines in his SECE account. The *Outside of* and *in* operators mean that the user is outside of and inside the given location, which must be represented as a polygonal structure. We are working on a location database that allows users to predefine places and polygonal locations through a GMaps-based GUI. Figure 10.11 shows a snapshot of this interface. The *moved* operator means that the user moved the given distance from where he was located when the rule was entered or triggered for the last time. Some example location events are shown below:

Bob near **"Columbia University"** { ... }
me near **37:46:30N,122:25:10W** { ... }
me within **3 miles** of **"1000 Massachusetts Avenue, Washington, DC"** { ... }
Alice in **clubhouse** { ... }
Tom within **5 miles** of **me** { ... }

Figures 10.12 and 10.13 show the main grammar expressions for location rules; Section 10.2.3.1 describes breifly the operation of location rules.

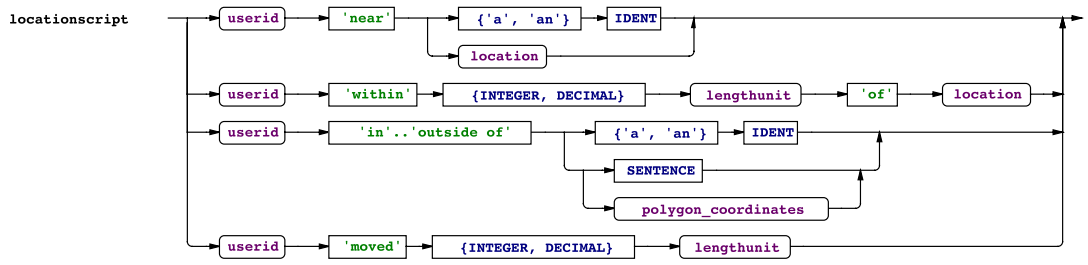


Figure 10.12: Grammar rule for location rules

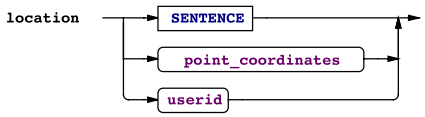


Figure 10.13: Grammar rule for the location element of location rules

10.2.3.1 Operation

Location rules are logically split into three parts: the user or subject, the location operator and the given location. For instance, in the first example rule above, the user is "Bob", the operator is "near" and the given location is "Columbia University". The operation of a location rule depends on the type of the rule's given location. As described above, this location can be expressed as geospatial coordinates, civic information, a well-known place, a user-defined place and other user's location. We are working on integrating a server based on the Location-to-Service Translation Protocol (LoST) for handling civic information and generic locations such as "a museum" and "a postal office". LoST [221] is an XML-based protocol for mapping service identifiers and civic information to contact URIs. This protocol is being enhanced for finding location-based services according to the user's location [222]. Generic locations will be accepted by any of the location operators described previously by preceding them with the "a" or "an" indefinite articles as in "me near a post office".

Figure 10.14 depicts the operation steps when a new location rule is entered into SECE, and it interacts with the above-mentioned LoST server. If the rule's given location (i.e., *location* in Figure 10.14) is a set of geographic coordinates or a user-defined place, SECE does not need to find out the location's geographic coordinates. In the case of a user-defined place, SECE stored its coordinates in a geo-cache at the

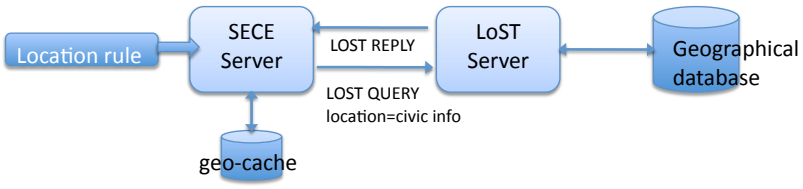


Figure 10.14: Operation flows for a new location rule entered into SECE

moment the place was defined by the user. If the given location is a user or generic place, SECE does not need to resolve this location when the rule is created. Since this location is variable, geocoding will be done whenever the rule is evaluated. SECE only asks the LoST server to geocode the given location if it is a civic address or well-known place and its coordinates have not been stored yet in the geo-cache. Figure 10.15 shows the operation flows when a location rule is evaluated. This occurs when the location of the rule's subject changes. Although in Figure 10.15, location changes are notified by the PS, the GLatitude web service may also be configured to detect the user location. Whenever SECE detects a change in a user's location, it checks out whether the user is the subject of any location rule. In this case, the actions to take depend on the type of the rule's given location. If the given location is a user, SECE retrieves the user's current location from the presence database. If this information is outdated or does not exist in the database, SECE tries to find it out via the PS or GLatitude. If the rule's given location is a civic address or well-known place, SECE retrieves its geographical coordinates from the geo-cache (i.e., they were stored at the time the rule was created). Then, SECE checks out that the geographic coordinates of the rule's given location and subject satisfy the specified geospatial relationship (i.e., the rule's operator) and, in this case, executes the rule. Only if the given location is a place type (e.g., "a museum"), SECE will pass the LoST server the place type, the geospatial relationship and the subject's geographic coordinates for obtaining the geographic coordinates of the places that satisfy the relationship with the subject. If the LoST server's response is not empty, the location rule is executed.

10.2.4 Request-Based Rules

Request-based rules specify the actions to execute in response to (a) incoming calls, IMs, emails, SMSs and voicemails, (b) outgoing calls and IMs, and (c) missed calls.

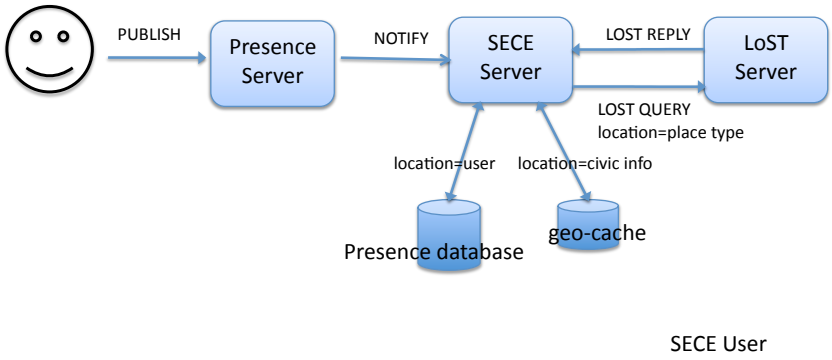


Figure 10.15: Operation flows for a location change

Figure 10.16: Sketch for request-based rules (optional parameters in parenthesis)

A request rule can start with the keyword *incoming*, *outgoing* or *missed*, followed by the type of event. While an incoming or outgoing call is always a SIP call, a missed call could be also a phone call. As described in Section 10.3, SECE tightly interacts with a SIP Express Router (SER) server for handling SIP requests. Users need to configure their SIP-based applications to forward any SIP communication to the SER server. When this server receives an incoming request for multimedia session or instant messaging (i.e., SIP INVITE and MESSAGE methods, respectively), it notifies SECE of this event through an efficient proprietary protocol. If the request is for a session, the SER server stands by until SECE determines the action to take, that is, to accept or reject the call. Then, SECE checks out whether any rule matches the incoming event. In this case, it executes the rule's actions, which most probably include some action that interacts with the SER server such as "reject" or "call". In the case that the incoming request is a call and the rule does not decide whether or not to accept it, SECE rejects the request. SECE uses the GVoice service [223] for detecting new missed calls, voicemails and SMSs. Thus, users need to configure the phone numbers that they wish to use in request-based rules into their GVoice account. As regards incoming emails, the Google e-Mail (GMail) service [224] is used.

Figure 10.16 depicts these rules' structure. All these events can be filtered by the

user destination and origin by using the *from* and *to* parameters, respectively. Some request-based rules are given below. The parameter *to* can be an address, phone number or shortcut (e.g., phones.personal). As a missed call may come from the cellular network or Internet, if the *to* condition is missing, both sources are considered. Users can restrict missed calls to cellular calls by typing a Google Voice (GVoice) identifier such as "to google voice" or "to gv", or specifying a particular phone number. SIP missed calls are indicated by a SIP address. The parameter "from" specifies the requester, which can be expressed in different ways. A nickname, as for example *bob*, implicitly includes all the communication means associated with the user identified by the nickname. An address identifies a SIP or email requester. Expressions like "John Brown's phone.personal" or "alice_smith@gmail.com's 1-646-32-8412" are used to refer to a particular user's communication means. Figure 10.17 shows the grammar rule for request-based rules.

```
incoming call from a workmate {
    if {[my activity is "on the phone"] } { forward sip:bob@example.com; }
}
missed call {
    if { [my activity is meeting] } {
        sms [incoming caller] "Sorry,I am in a meeting but will call you back asap.";
    }
}
incoming call to me.phone.work {
    if { ![my location is office] } {
        autoanswer audio no_office.au;
        email me "[incoming caller] tried to reach you on your work phone at
        [incoming time]";
    }
}
incoming email from my boss {
    if { ![my activity is working] } {
        sms me "New email from the boss at [incoming time]. Subject:
        [incoming subject]";
    }
}
incoming im {
    if { [my status is away] } {
        sms me "[incoming from] sent this IM: [incoming message]"
    }
}
```

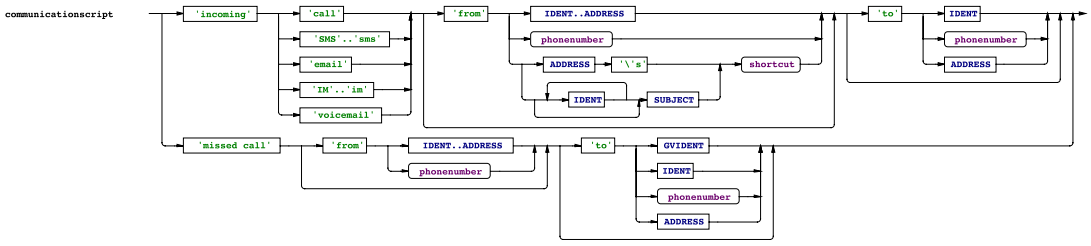


Figure 10.17: Grammar rule for request-based rules

Figure 10.18: Sketch for context-based rules

10.2.5 Context-Based Rules

These rules specify the action to execute when some context information changes, such as presence or sensor state. These rules always start with the keyword *if*. If the rule is about the user that is entering the rule, this keyword is followed by *my*. Otherwise, the *if* keyword is followed by an identifier of one of the user's contacts. Figure 10.18 depicts the structure of this kind of rule and some examples are shown below.

if my **activity** changed { schedule "**activity:** [status **activity**]"; }
if **bob@example.com**'s **status** is **available** { alarm **me**; }
if my **stock.google** >**14** { sms **me** "**google stock:** [stock **google**]"; }

The user's context (e.g., **activity**, **status** and **stock.google** in the above rules) is a hierarchical variable in the form of x.y.z.t, such as phone.office, activity and office.temperature built from the user's tree registry, which is described in Section 10.3. The rule's subject is given by the *my* and *'s* operators (e.g., "bob's phone.office" and "my activity"). Shortcuts can be used instead of these operators, so that for example *bob.device.mobility* is equivalent to *bob's device.mobility*. Internally, each context-based rules generates a listener for a node of the user's registry. Figure 10.19 shows the grammar rule for context-based rules. Relational operators can be expressed as symbols or text (e.g., the equal relation can be given by "=" and "is"'). The **changed**, **removed** and **added** conditions are satisfied when the rule's context is changed, removed or created, respectively. When no shortcut is included, only the **changed** condition is allowed. This means "any change in the specified user's registry".

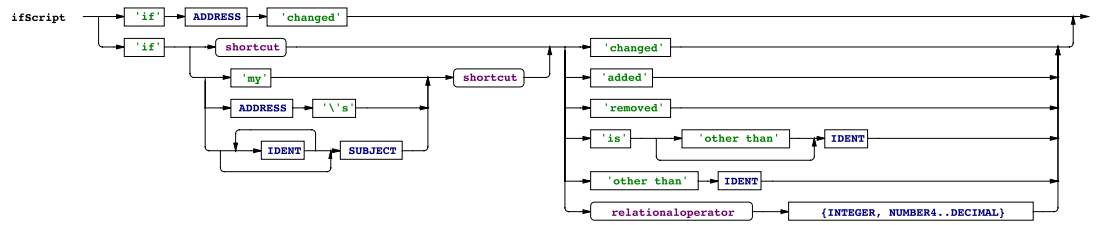


Figure 10.19: Grammar rule for context-based rules

Information derived from sensors such as smoke, light, humidity, motion and temperature can be also used in context-based rules. Naming of sensors is an open problem that, for now, is beyond our scope. We have adopted a simple solution that consists in a translation table from internal, machine-friendly names (e.g., 00-0C-F1-56-98-AD) to more user-friendly identifiers (e.g., office.smoke).

if my **warehouse.motion** equals **true** { sms **me "person in the warehouse."**; }
if my **office.smoke** equals **true** {
  sms **me "fire in the office"**;
  calltts **firedepartment** "**fire in** [status **office.address**]";
}

10.2.5.1 States vs. Events

SECE is designed for handling events (i.e., state transitions) that trigger a set of actions. State transitions are suitable for discrete events such as calls and calendar events. However, state transitions have some limitations on behavior that combines a set of variables to define the state of another variable. For example, to manage the home heating systems, events would have to be defined for people entering and leaving the house, along with temperature and time-of-day conditions. It is much easier to write such cases as predicates, such as "turn on the air conditioner if the indoor temperature is higher than 80 F and I am at home". One possible syntax for such conditions is shown in the example below. Only one predicate can exist for a variable and, hence, rule conflicts on actuators are avoided. We are currently exploring the applicability of predicate- and event-based systems, and whether it makes sense to integrate them or keep them separate.

```
ac := temperature > 80 and me in home;
```

10.2.6 Error Detection and Handling

Users can create multiple rules of a variety of kinds, which are running parallel on SECE. It is therefore possible that a user creates a rule that involves unexpected results, specially if the user has little experience in creating rules. Although unexpected results are very hard to detect, we can address this issue in several ways. SECE provides online documentation about events and how to create correct compositions. Nevertheless, we can not assume that users enter the correct rules that will execute exactly what they intended. Thus, we are planning to add a dry-run mode that allows users to know what SECE would do when rules are triggered. Actions are just logged rather than executed. A functionality to test rules before inserting them into SECE will also be provided. This will show the sequence of rule actions and possibly the value of rule variables that will be executed when triggering the rule. Moreover, we will avoid infinite loops by simple preventions such as limitations on the times a particular action is invoked and the CPU time taken by a rule. Moreover, we are working on formalizing SECE events through an event algebra for tackling instantaneous (point-based) and durative (interval-based) events. This algebra will formally define how these events can be composed and their temporal relationships (e.g., sequence and concurrency). This will permit SECE to exactly detect when the state of resources, such as actuators (see Section 10.2.5.1) and voicemails, have to be changed, thereby avoiding unexpected or wrong results. For example, if the event of the rule "if me in home { ac on; }" is treated as an instantaneous transition state, the user's air conditioner will be switched on at any time when he gets home. However, this rule does not necessarily involve turning the air conditioner off when the user goes out. This event would better be considered as the durative event "during(me in home){ ac on; }", and hence SECE would switch the air conditioner off whenever the event "me in home" gets false. Composite events require more complex semantics definition. The length of durative events should be considered when composing events. Let us take the example "If me in home and temperate $>$ 80F { ac on; }" that means "if I get home and afterwards my home temperature is higher than 80F, switch the air conditioner on". Then, let A be the set of all user location events in his house. Let B be the set of all temperature events higher than 80 Fahrenheits. If both A and B events were considered as instantaneous events, SECE would wrongly detect the situation depicted on the upper line in Figure 10.20 as a matching event. The

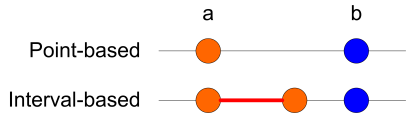


Figure 10.20: Example of point-based and interval-based event timestamps

user got home but, before the temperature rose above 80 Fahrenheits, he left home. On the contrary, if A events were considered as durative events (the bottom line in Figure 10.20), SECE would detect that this event finished before the event B arose and the rule would not be triggered incorrectly.

10.3 Architecture

Figure 10.21 gives an overview of the overall SECE architecture and how it interacts with its environment. SECE is a web service that interacts with other web services, namely Google Services and Social Media services such as Twitter, Flickr and Facebook. The rules that are running on SECE and the rules' actions that will potentially be executed determine the services with which SECE needs to interact. Thus, based on the kinds of rule that the user wishes to create and the actions that she wishes to compose, the user will need to configure the proper third-party services in her SECE account. Section 10.2 explains the SECE rules and actions, and their required services in more detail.

We are developing two services that tightly collaborate with SECE: the PS and the VoIP proxy server. The PS is built on the Mobicents Presence Service [225], which is compliant with SIMPLE . This server is responsible for collecting and aggregating context from different sources, and sending it to SECE. This server receives presence publications from context sources that contain the latest information about a user, and in turn notifyies SECE of the context changes. In the SECE framework, context sources include user devices' presence applications and gateways that control sensor networks, energy consumption and user location via Radio Frequency IDentification (RFID). To use the presence service, the end user needs to create an account from the SECE website in order to obtain the SECE PS access information. Thus, the user can configure the SIMPLE-compliant presence applications that run on her mobile devices or desktop computers to use the SECE PS. Section 2.1 mentions some SIMPLE-compliant instant

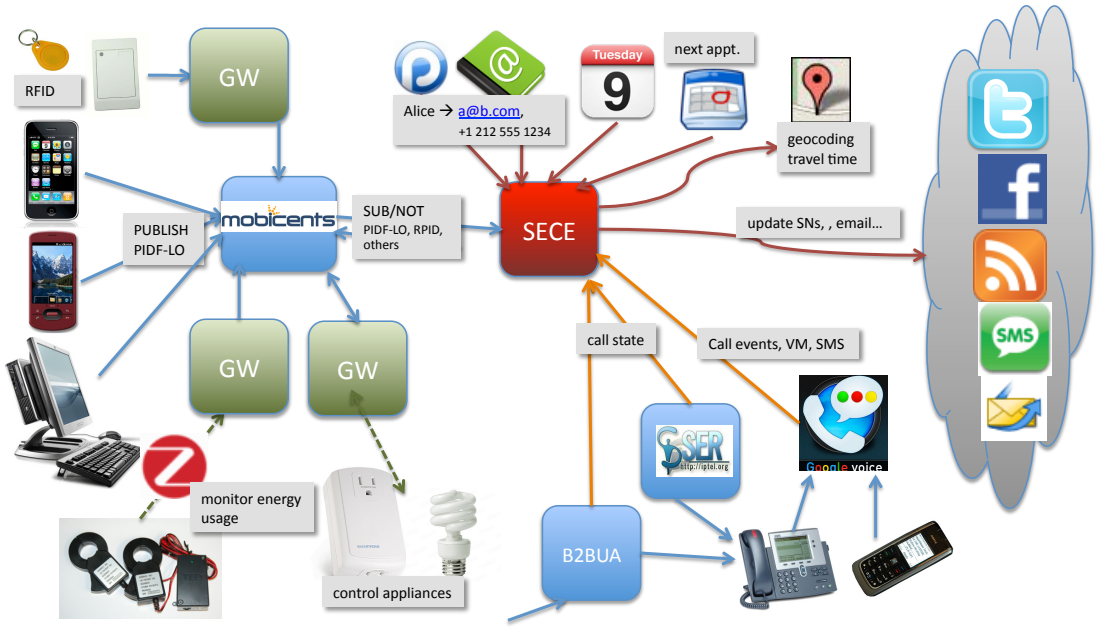


Figure 10.21: SECE architecture

messengers. In the future, the SECE PS will interact with the SECE home gateway for obtaining information from sensor networks and changing the state of actuators. The VoIP proxy server is a SER [226] extended to interact with SECE for handling users' SIP communications. This server and SECE implement an efficient binary protocol that lets SER inform SECE of a SIP event and lets SECE notify SER of the action to take for this event. Basically, SER informs SECE of calls and IMs. If a call or an IM matches a rule, the rule is triggered, and hence it decides to forward, reject, or modify the call by invoking an action. Then, SECE will let SER know about the action to take. Figure 10.22 depicts the interaction model between the SER server, SECE and media servers. The user needs to create a SER account through her SECE account for using the VoIP proxy service. The user also needs to set her SIP-compliant multimedia applications to use the SECE VoIP proxy server as outbound/inbound SIP proxy.

SECE considers not only the user's context but also information about external entities other than sensors, such as her buddies. SECE keeps the user information in a Document Object Model (DOM) [227] tree registry. The user information is not restricted to personal information but also includes contextual information from sensors and Internet services. Context-based rules associate events with the registry's nodes. A

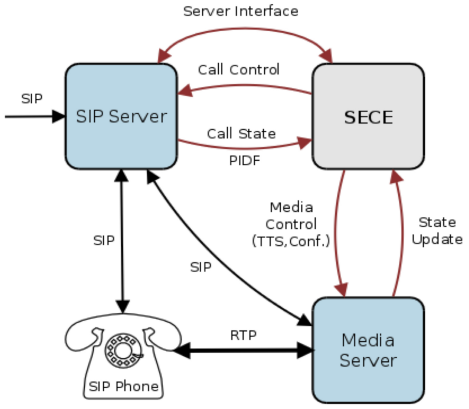


Figure 10.22: SER SIP server and SECE interaction model

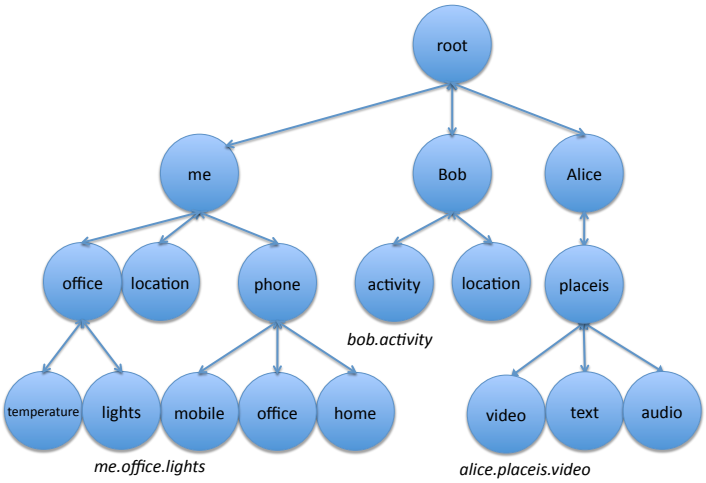


Figure 10.23: Partial user information registry

rule does not have to be associated with a leaf node; it can be associated with any node. The benefit of associating rules with top-level nodes is to write generic rules like *"if Bob changes* {...}" to allow monitoring any activity related to a subtree. Figure 10.23 shows a user's partial information registry. Each node of a user's tree registry represents an attribute, which is identified hierarchically by concatenating the intermediate attributes from the root element, which is the user's name, as in *me.phone.home*. We refer to a registry node's identifier as shortcut.

Since SECE is aimed to converge multiple Internet services and information sources, the address book of SECE users can be composed from multiple external services. Currently, SECE obtains the user's address book from his Google Contacts (GContacts)

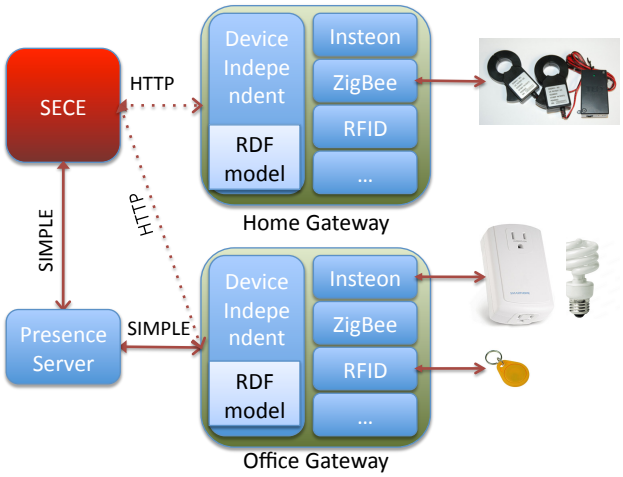


Figure 10.24: The architecture of sensors and actuators gateway

account [228]. In the future, users will be capable to aggregate their contacts in social networks and presence-enabled applications into their SECE address books. SECE will communicate with the PS for downloading the user's buddy lists in presence applications. When a new contact is downloaded, SECE analyzes its information and maps it to the user's tree registry. A contact can have multiple identifiers such as SIP and email addresses, and phone numbers. Users can also identify their contacts with more convenient nicknames (e.g., Bob or Anne) for convenience. Nicknames therefore should be unique. Users can type any kind of identifier in their rules and SECE will figure out what contact is associated with the identifier.

Figure 10.24 shows the interactions between SECE, its PS and home gateway for handling sensor information. SECE obtains sensor information through SIMPLE notifications that include RDF [170] documents, which makes SECE sensor-network agnostic. Actions on actuators are described in RDF documents that are sent to the gateway via POST HTTP. The gateway is split into two layers: the device-independent layer maintains an RDF database that represents the conceptual sensor model. The protocol layer carries out the necessary translations between the RDF model and the device-, network-dependent information and actions. SECE automatically creates Tcl commands for each actuator after being notified of the RDF model. Currently, we are experimenting with ZigBee and Insteon wireless device control modules.

A first prototype of SECE has already been developed as a web service and is being tested by members of the Internet Real Time Lab (IRT) at Columbia University. Figure

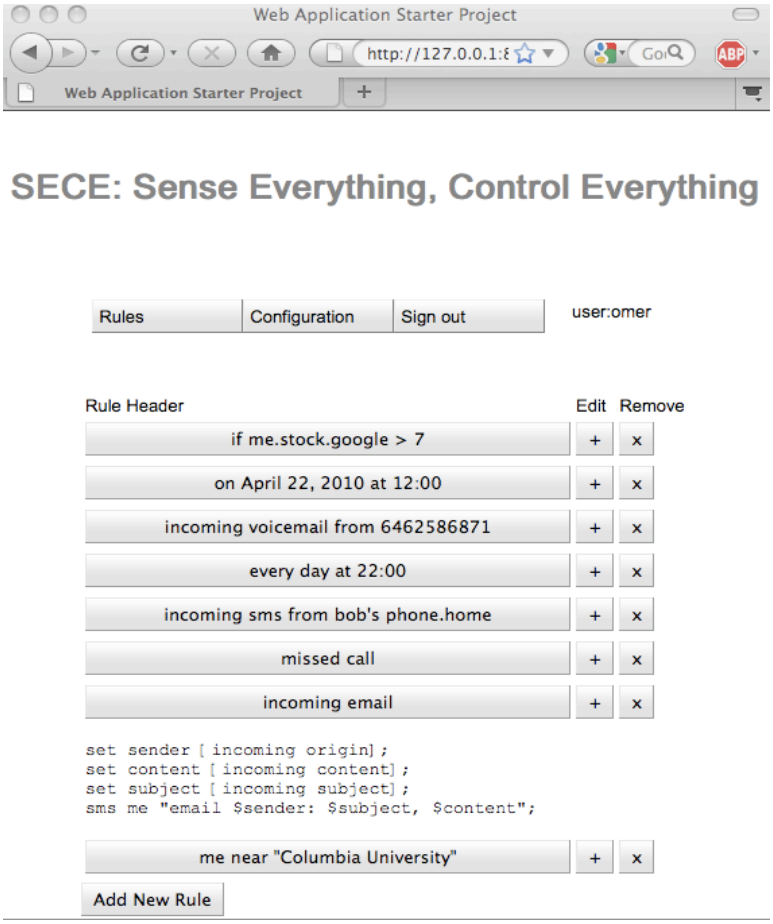


Figure 10.25: Snapshot of the SECE web service

10.25 shows a snapshot of the SECE web service that shows the rules created by a user. By default, only the rule events are shown; double clicking a rule header displays the rule's actions.

10.3.1 The Software Components of SECE

Figure 10.26 shows the software components of SECE. We are developing SECE in Java due to its extensive libraries and support for any OS. Figure 10.26 only shows some relevant Java libraries such as ANTLR, which is used by the language compiler, JACL [229] that is a Tcl implementation in Java, JAINSIP for SIP signaling and GDATA to access the Google web services. The agent layer contains the agents that communicate with external services. Agents can generate events (e.g., the Mobicents agent creates

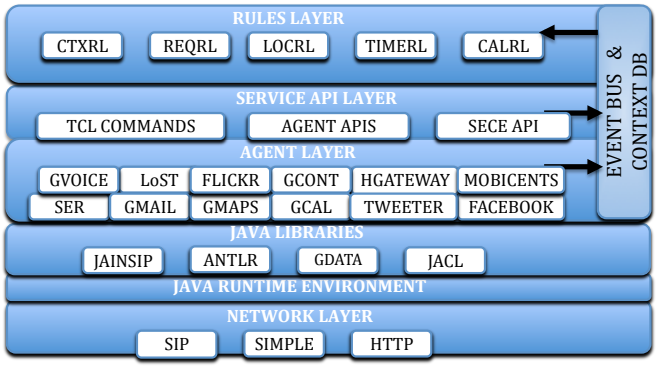


Figure 10.26: The software components of SECE

presence events), provide some useful functions (e.g., the GMaps agent provides direct and reverse geo-coding) or take some action (e.g., the Gmail agent can send emails). The rules layer contains the rule implementations. These implementations utilize the service API layer to subscribe to interesting events, to check rule conditions and to execute rule actions if necessary. The context DB contains all the users' and their buddies' context, including presence, location, preferences, configuration data and sensor information. Rules only can modify or read this DB through the APIs in the Service API layer.

10.4 Enhancing SECE Toward Ontology-Based User-Defined Rules for Automatic Service Discovery

As it stands, SECE has no way of automatically discovering a new type of service, generating a rule language for it and incorporating it in its system. The set of services that are supported in SECE are hard-coded. Thus, we have enhanced SECE to support ontology-based user-defined rules for automatic service discovery. The simple but illustrative example below emails the user whenever a new restaurant that satisfies the given conditions is found.

Any **japanese restaurant** that is cheaper than **20$** and whose **location** contains **Manhattan** {
email **me "new restaurant found"** "**Details:** [event **description**]";
}

We have incorporated GloServ (Global Service Discovery Architecture) [230] [231], an ontology-based service discovery system, within SECE's back-end architecture. GloServ

classifies services in an ontology and provides ontology descriptions of different service domains. GloServ provides an API whereby service ontology descriptions, for a number of domains, can be downloaded and queried for with an ontology query. GloServ uses the OWL DL (OWL Description Logic) ontology to describe its services. Thus, SECE can access these OWL specifications in order to dynamically define rules for each specific service domain. Users are made aware of these services by a front-end application to SECE that displays the discoverable services' descriptions. Currently, users still need to learn how the rules are constructed, however, for the future, we plan on building a GUI that will use the ontology description to aid the user in constructing the rules. Sections 10.4.1 and 10.4.2 describe the design and implementation of these enhancements. We have developed a SECE sublanguage for web service events, which is discussed in Section 10.4.3. Section 10.4.4 discusses some future steps for further enhancing SECE.

10.4.1 Design

Figure 10.27 outlines the main interactions between SECE, GloServ, front-end applications and web services. Although SECE is a standalone web service, we are enhancing it toward a more flexible architecture. We envision SECE as a common layer on which advanced front-end applications can be built. In this mode of operation, end users are connected to front-end applications that provide more functionality or fancy GUIs. Users that are not comfortable with scripts will therefore be able to use more sophisticated graphical tools with probably advanced online guides. On the other hand, the SECE web service provides a lightweight solution for the sake of simplicity and efficiency. Users can enter rules into the SECE web service quickly without any resource-demanding graphical application, which is very convenient for mobile user devices with limited resources.

From the moment at which a web service rule is entered into SECE on, SECE will periodically communicate with GloServ for discovering web services that match the rule. A GloServ request specifies the web service of interest as a SPARQL query [232] and matching services' profiles, if any, are sent to SECE into a GloServ response. Below, An example SPARQL query that requests a vegetarian restaurant that is inexpensive and open 24 hours, and whose address contains "new york" is shown:

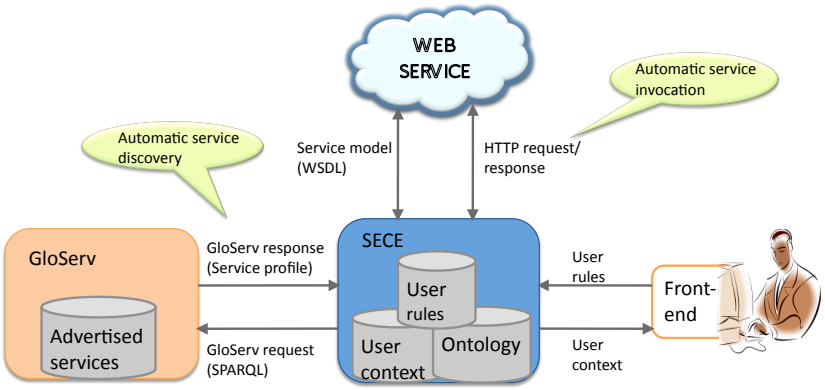


Figure 10.27: SECE, GloServ, front-end applications, and web services

```
SELECT ?x WHERE{?x ?p0 <http://www.owl-ontologies-gloserv.com/Restaurant.owl#inexpensive> .
 ?x ?p1 <http://www.owl-ontologies-gloserv.com/Restaurant.owl#vegetarian> .
 ?x <http://www.owl-ontologies-gloserv.com/Restaurant.owl#open24Hours> true .
 ?x <http://www.owl-ontologies-gloserv.com/Restaurant.owl#streetAddress> ?var0 .
FILTER(regex(?var0, "new york","i"))
}
```

If a new web service matches a rule, SECE executes the rule's body. In the future, some SECE actions will be capable to compose and communicate with services automatically based on service descriptions encoded in WSDL, as addressed in Section 10.4.4. Section 2.8.1 gives an overview of the technologies involved in web service automation. We have integrated a Jena ontology model, which contains the necessary ontologies' schemes, and an agent that pulls GloServ for checking out new web services of interest, which we refer to as GloServ Context Mediator.

10.4.2 Implementation

SECE stores the OWL specifications of web services in an ontology database that is built upon the Jena Framework [233]. When a web service rule is entered into SECE, it takes the following steps: 1) parse the rule (i.e., syntactic checking); 2) verify that the described kind of web service exists (i.e., semantic checking); 3) subscribe to the described web service event; and 4) take the rule's actions whenever this event occurs. Figure 10.28 outlines the main interactions for creating a web service subscription. In this figure, the component *SECECore* is the orchestrator for all the software agents in SECE.

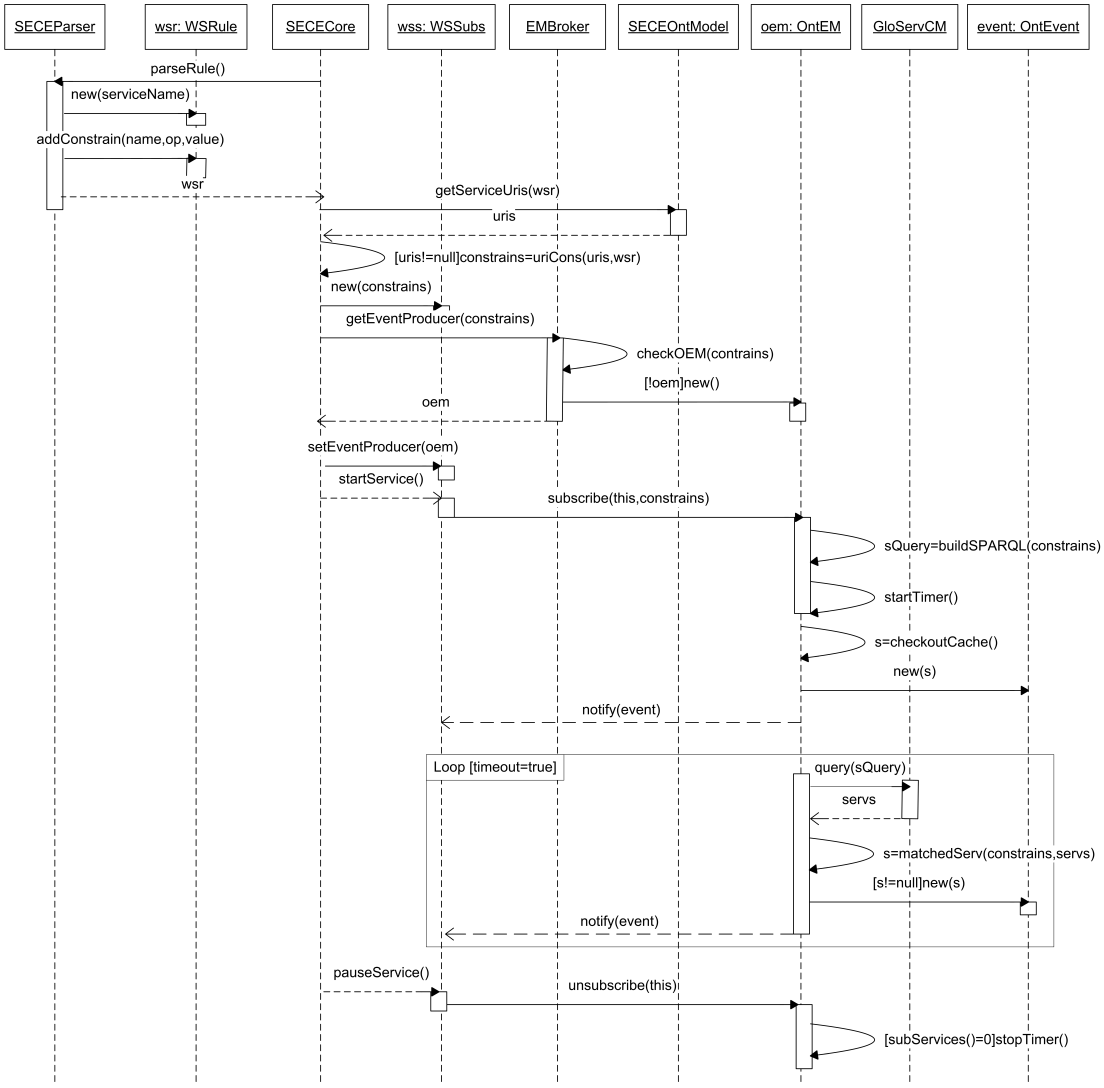


Figure 10.28: Sequence diagram from entering a web service rule to querying GloServ

When a web service rule is entered into SECE, the SECE parser checks that the input rule is consistent with the SECE ontology-based sublanguage, which is generated by t he ANTLR grammar outlined in Section 10.4.3. As a result, the parser creates a *WSRule* object that encapsulates information about the rule, namely a web service event and the actions that will be taken if this event occurs. A web service event is defined by the web service's name and optionally a set of property constraints in the form of *(propertyName, operator, value)*. If the rule parsing is successful, SECECore verifies that the rule's web service description corresponds to a web service's ontology. To this end, SECECore interacts with the SECE ontology model (i.e., *SECEOntModel* in Figure 10.28). The SECE ontology model encapsulates the Jena database that contains the web services' ontologies and provides convenient functions for searching and retrieving information about them. A web service description is semantically correct if there exists an ontology that describes a service type named as the web service and associated with the described properties. SECE will ask the SECE Ontology Model for the namespace URI of the web service type and its properties. If the web service does not correspond to any ontology, the SECE ontology Model returns null values. This means that the rule's web service event is semantically incorrect, which results in aborting rule creation and warning the user. Otherwise, the rule's web service event is semantically correct and SECECore proceeds to create the corresponding subscription (i.e., *WSSubs* in Figure 10.28). SECECore, therefore, retrieves an event monitor from the Event Monitor Broker (*EMBroker* in Figure 10.28). An event monitor (*OntEM* object in Figure 10.28) is an agent that watches a particular service type and generates an event whenever a new service of this type is discovered. The Event Monitor Broker maintains a list of the event monitors that are monitoring web service types. Thus, if an event monitor for the web service event already exists, the Event Monitor Broker returns it. Otherwise, the Event Monitor Broker creates a new one, appends it to the list of monitors and returns it. Then, SECECore associates the event subscription with the event monitor and starts the subscription. Event monitors maintain a cache of discovered events. When a new subscription is associated to an event monitor, it checks out its cache of web services and notifies the new subscription of any cached web service that matches the subscription.

Starting and pausing an event subscription makes the corresponding WSSubs object subscribe and unsubscribe to the associated event monitor, respectively. When an

event monitor receives a subscription request and there are no other subscribers, it creates the corresponding SPARQL query that describes the web service event. The event monitor also starts up a recursive timer to query the GloServ Context Mediator (i.e., *GloServCM* in Figure 10.28) with the SPARQL query periodically. Whenever GloServCM is invoked, it sends the SPARQL query given as parameter to GloServ, which replies with the matching web services, if any. Thus, the event monitor creates an OntEvent object for each matching service, and notifies the subscriber of this event. When an event monitor is associated with more than one subscriber, the SPARQL query represents the least restrictive subscription. When a web service matches this subscription, the event monitor checks out whether this service matches any of the other subscriptions. Figure 10.28 outlines this process through the *matchedServ* call.

10.4.3 SECE Ontology-Based Sublanguage

SECE provides a simple and generic ontology-based language for end-users to define web service rules. In line with SECE philosophy, this language looks like natural English and is easy to learn. Its basic structure is "any *service* whose *prop rel value*" given that *service* is a web service class, *prop* is one of this service class' properties and *rel* and *value* represent a restriction on the property. *Rel* is a relational operator that depends on the property's type: *contains* and *is* for textual values, and $=, <, >, \leq$, and $\geq$ for numbers. Multiple property constraints can be added by the *and* and *or* boolean operators as for example "any shopping offer whose type contains "ski boots" and whose price is cheaper than 150$". Equality on numeric properties can be expressed by the verb *has* followed by a number and the property name as in "any happy hour and inexpensive bar that has 20 free seats". Users can place property values before the class name when the property works as adjective. In the previous example, the *bar* class has the boolean properties *happyHour* and *inexpensive*. Boolean constraints can also be expressed by the operators *that has (no)* and *that is (not)* as in "any restaurant that has delivery", "any restaurant that is open 24 hours" and "any cultural exhibition that is free and is not crowded". Boolean constraints can be applied to class properties or classes themselves, which depends on the ontology's structure and is transparent for end-users. An example of boolean property is the above-mentioned *delivery* property whose domain is the *restaurant* class. Boolean constraints on classes restrict inherited

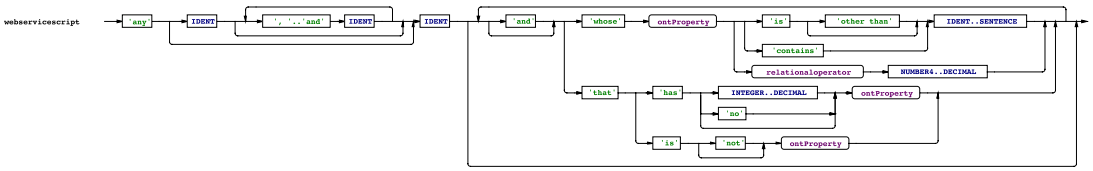


Figure 10.29: Grammar rule for web service rules

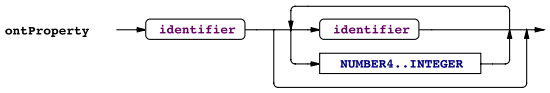


Figure 10.30: Grammar rule for ontproperty elements in web service rules

types as for example "any restaurant that is southamerican" subscribes to restaurants of type Southamerican, which is a subclass of the class Restaurant.

The SECE ontology-based sublanguage is a simplistic approach for testing web service rules that will be enhanced in the near future. This language is built on some simple agreements between SECE and GloServ about ontology class and property naming. For example, ontology class names starts with an upper-case letter while property names with a lower-case letter. Property names composed of more than one word are made up by concatenating the first word with the other words having their their first letter in upper-case, as for example "open24Hours". Based on these simple rules, SECE can analyze the class and property names of each web service ontology and therefore check out the syntax of web service rules. This first approach presents some limitations in expressiveness that will be overcome in the future. Figures 10.29 and 10.30 show the grammar rules for web service rules added to the SECE grammar.

10.4.4 Future Work Towards Automation

Integrating web service rules into SECE brings out many possibilities in the Semantic Web. Sections 10.4.4.1 and 10.4.4.2 briefly introduce some of these possibilities.

10.4.4.1 Automatic Learning of SECE Rules

Automatic suggestions about service composition, events and actions are essential for beginner users, whom can feel lost when creating rules. We are growing SECE towards automation, and hence we find that suggestion systems for SECE rules should

be automated too. Thus, in the near future, we will provide a mechanism for front-end applications to build suggestions dynamically. We will model the semantics and syntax of SECE rules ontologically. Front-ends applications will therefore be able to obtain the rules' ontologies and reason about them, and hence they will dynamically create suggestions on rule construction. This mechanism will provide front-end applications with the ability to learn rules' semantics and syntax automatically.

Although SECE offers a set of in-built rules, front-ends may want to offer more sophisticated rules, for example, by combining multiple kinds of events or using an event syntax other than SECE's. To allow front-ends to add new event syntax dynamically into SECE, we will provide proper interfaces to subscribe to events, obtain user context and interact with SECE core functions. Ontologies for rules, events and context as well as semantic paths will provide developers with a high-level interface to SECE data, independent from any underlying data structure. This mechanism may be considered as a plugging system whereby third-parties can insert customized event descriptions into SECE, and SECE will learn these descriptions automatically. The SECE web service will provide an API for entering new rule event descriptions.

10.4.4.2 Event-Based Context-aware Web Service Composition

SECE provides a set of actions for users to build up compositions. Some actions interact with web services, such as *tweet*, *publish* and *email*; other actions send protocol-specific requests, such as *call* (i.e., SIP INVITE); and others are supportive routines. The set of web services with which SECE communicate and the communication flow is hard-coded. Therefore, SECE compositions are static in the sense that, once a composition is created, it will not change. We are planning to incorporate dynamic compositions to SECE through automatic web service discovery and composition. Section 2.8.1 outlines the main features of web service automation. SECE will interact with web services automatically by retrieving their models and, according to their WSDL specifications, constructing HTTP requests. Two new SECE actions will add this functionality: *find* and *plan* for discovery and composition, respectively. An example rule is shown below, in which the *plan* and *find* commands are pseudo-code because they have not been implemented yet. In this example, whenever a new flight is found, other web services are discovered (i.e., hostels, car rentals and restaurants) and composed (i.e., trip planning). Note that the *plan* action could invoke *find* in order to discovery web services that are

necessary for the composition. As the discovered web services and the communication with them can be different each time the composition is executed, we say that this composition is dynamic.

```
Any domestic flight that is cheaper than 200$ and whose date is after June 1, 2011 {
    p=plan flight with hostel and car rental;
    r=find good restaurants according to $p;
    email me "new plan found" "Details: $p $r";
    sms me "New Plan discovered. See email inbox for details!";
}
```

With these two new actions, SECE would perform semantic web service discovery and composition that does not need user interaction to be executed; it is automatically triggered by events. In addition, SECE would allow combining static and dynamic composition. For example, the rule above provides dynamic composition through the *plan* and *find* actions and static composition through the *email* and *sms* actions. Besides web service discovery events, semantic compositions could be triggered by any SECE event, such as location, context, calendar, communication and time. For instance, the example below discovers web services based on time events. Web services of kind "brunch offer" are found according to the user's location and are emailed to her.

```
Every Sunday at 12:00 {
    offer=find brunch offer whose location is near me;
    email me "Brunch offer" $offer;
}
```

As the Semantic Web is not widely adopted yet, hybrids platforms like SECE are necessary to offer users flexible and powerful composition tools. Table 10.4 indicates the types of composition that SECE already supports (white column) and will support in the future (gray columns). Rows define the events that trigger the compositions and columns the types of web service communication in the compositions.

10.5 Conclusions

SECE enables end-users to create advanced services. Although users today can use several individual Internet services, there is currently no easy way to create new services that integrate diverse information, such as location, presence, IM, SMS, calls, Facebook, Twitter, sensors and actuators. Facing it, we are developing a context-aware platform

	Semantic service communication	**Hard-coded service communication**	**Both kinds of communication**
Web service events	Dynamic composition triggered by discovered web services	Static composition triggered by discovered web services *(enhanced SECE)*	Mixed composition triggered by discovered web services
Other events	Dynamic composition triggered by real-world events	Static composition triggered by real-world events *(typical SECE composition)*	Mixed composition triggered by real-world events

Table 10.4: Types of SECE composition

and associated language to create user-personalized composite services and automate their execution. SECE is intended for not only developers but also end-users without programming skills. SECE users create natural-language-like rules to compose their own services. Every rule specifies the event that triggers the rule's actions (i.e., its service); SECE monitors the event and proactively executes the service whenever the event occurs. In the future, it will be possible to develop a more advanced GUI (e.g., suggestions and templates for service composition) without modifying the SECE core. The definition and syntax of the language has been finalized. We developed a multi-user server to allow users to edit, compile, and deploy SECE scripts, which provides a web-based interface. From the components shown in Figure 10.26, all the rules are fully implemented except some location operators. The home gateway and Mobicents PS are also still under development. Although rule conflicts have already been considered for a particular user, multi-user conflicts should be studied in the future, as well as run-time error handling, specially for resource rules as described in Sections 10.2.5 and 10.2.6. Future work also includes experimenting with real-life scenarios in order to demonstrate the usability of the language by end-users and the system scalability.

Moreover, we integrated SECE with GloServ (i.e., a scalable network for web service discovery) for providing a context-aware, event-based platform for web service discovery and composition. The Semantic Web is investing much effort in developing standards for providing automatic web service discovery and composition. Although many authors have been interested in this exciting topic in the last decade, complete solutions do not yet exist. Most authors describe or propose theoretical work. The few that present real implementations are partial solutions and domain-specific. Thus, there is a strong need for general-purpose platforms for automatic web service discovery

and composition. Such platforms should provide intuitive and user-friendly interfaces that do not require engineering or technical skills. Besides template-based composition, end users should be able to orchestrate service composition. Service discovery and composition should be user-centric, context-aware and proactive. SECE is a user-centric, context-aware service composition platform that provides a formal language similar to natural English for defining event-based rules. Thus, SECE satisfies many of the above-mentioned requirements to become an underlying platform for automatic web service composition. To move SECE towards the Semantic Web, we implemented the communication between GloServ and SECE. We extended SECE with an ontology database that stores the web services' schemes that are downloaded from GloServ. We also developed an ontology-based language for creating rules that work as subscriptions to web service discovery events. This language is independent from any particular web service description, and hence new kinds of service supported by GloServ can be added transparently. We described the whole platform and the advantages it can bring to the Semantic Web. This allows subscribing to web service discovery events by creating rules in a user-friendly language that looks like natural English. SECE also allows creating service compositions that can be triggered by real-world events such as context changes, location, or time. This permits end-users to define and personalize context-aware web service discovery, invocation and composition based on a variety of events. In the future, the technologies developed by The Semantic Web for fully automation in web service composition and automation (see Section 2.8.1) will be integrated into SECE. SECE will be decoupled from front-end applications so that more fancy GUIs can be built on top of it. Moreover, modeling SECE rules ontologically will provide front-ends with the means of understanding and learning new SECE rules automatically.

11

Discussion

Next-Generation Networks (NGNs) are aimed to bring value to human life through new experiences and convenient services as well as to provide a playground for everybody to create, share, compose, and deliver services. Presence information is considered as a key service in NGNs. This provides applications with all the user context that is necessary to handle user communications in an intelligent, proactive manner. NGN relies on an all-IP approach, the IMS, that allows IP-enabled devices to access any service regardless of the access technology. NGN uses SIP for controlling communication sessions and a set of extensions of SIP, which are known as SIMPLE, for handling presence information and instant messages. Nowadays, there is an overwhelming emergence of technologies that will support all the deployment aspects of NGNs, from hardware, network infrastructure to application development. Adopting all these technologies implies a costly investment that is restraining operators from moving towards next-generation services. On one hand, operators need to adopt a customer-need-driven model for deploying the services that attract the largest number of people, thereby maximizing revenue the most. On the other hand, operators need to be sure that such revenue will compensate for the investment done and the impact of the provided services on the network. The capacity impact of next-generation services on the operator network is far from trivial due to multiple reasons. These services will be ubiquitous, access-network- and device-independent. They will in large measure rely on context information, enriched data, and social relationships. These features require complex functionality, and introduce traffic load, which operators should be capable to bear for world-wide used services. User context dissemination is specially challenging because information updates are

timely spread over the network, which traverse centralized servers and may involve terminal devices with limited resources. Moreover, the protocol for distributing user context in NGNs, that is, SIMPLE, is a subscription-based framework that involves periodic signaling traffic for keeping subscriptions alive.

This thesis is aimed to contribute to the deployment of presence-based next-generation services, and provide a comprehensive view of what the presence service is and what it can be used for. In a nutshell, the main goals of this thesis can be summarized as follows: to enlighten readers about the SIMPLE framework; to give a deep analysis about the parameters that affect presence overload and how they can be tackled for optimizing the presence service; to overcome some limitations of SIMPLE regarding the control on the timing of presence updates; to optimize the overall presence service by reducing the traffic sent over access links and the core network; to ensure that the presence service's performance is not degraded because of delays in presence updates that are due to some rate control; to probabilistically model changes in presence information; to provide a detailed analysis of presence traffic in large-scale scenarios and the impact of this traffic on the IMS; to reduce presence traffic in large-scale scenarios and therefore alleviate the impact of this traffic on the IMS; to optimize the PS' performance in large-scale scenarios for the sake of scalability; to provide an scalable, user-centric platform for handling presence information; to provide a web service for user-created and automatic service composition that relies on presence information.

The realization of this thesis required to acquire a good knowledge about both the industry and academia efforts in the frame of presence-based services. We transferred this knowledge and some of this thesis' results to several entities related to the Industry and Academia. We incorporated the presented personal proxy within a technological project directed by Fundacio I2Cat. As well, some of this proxy's functionality was integrated within other project mutually directed by Vodafone R&D and Fundació I2Cat. We contributed to an IETF Internet-Draft [193] that analyses presence traffic, which may be noticed in this draft's Acknowledgements section. A stay of one year in Columbia University, in New York, was granted for collaborating with the IRT group at the Department of Computer Science. During this year, we initiated a new project, which evolved into the presented software platform SECE. The contributions of this thesis have been published throughout 3 book chapters, 4 international journals (three of them indexed in Journal Citation Reports (JCR)), 10 international conference

papers, 1 demonstration in an international conference and 1 national conference paper. Moreover, 2 papers are under review (second round with minor changes and first round) to be published in international journals indexed in JCR. Appendix I gives information on these publications. Throughout this thesis, we discussed how we tackled the above-mentioned goals, and the resulting contributions. Nevertheless, below we summarize these contributions for the sake of clarification:

- SIMPLE is not a compact protocol but a collection of specifications, both IETF RFC and Internet-Drafts. Moreover, the large number of specifications related to SIMPLE and the numerous topics covered by these specifications (e.g., privacy, information, optimization, IM, location, etc.) makes it difficult to understand SIMPLE in depth. This may result in beginners giving up the SIMPLE framework. To face this issue, we gave a comprehensive vision of this framework, by analyzing the concepts involved, its specifications and how these specifications relate to each other.

- SIMPLE is not concerned about the subscriber's needs on presence attributes when limiting the rate of presence notifications. Although notification rate control is suitable for reducing presence traffic, it may involve watchers keeping obsolete information for too long. Watchers however should not perceive delays in receiving presence notifications, since the actual usefulness of the presence service is its instantaneous nature. Thus, a tradeoff between traffic optimization and the watcher's needs on information consistency should be found. We proposed an extension of the XML schema for SIMPLE notification filters that defines new trigger conditions for controlling the notification rate of particular presence attributes. This allows subscribers to indicate the maximum and minimum rates at which they wish to be notified of subsets of presence information. Furthermore, this extension provides other trigger conditions for pausing and un-pausing notifications of presence attributes as well as triggering one-time notifications. This allows a subscriber to adopt a pull approach while having its notifier maintaining the resource state information up-to-date. SIMPLE as it stands does not provide this functionality. However, it may be very resource-efficient under some circumstances. The proposed notification filtering is specially interesting in location-based systems. In this systems, location updates occur very frequently

and are timely spread over the network. Regardless of the rest of presence information, our approach permits to control the rate of location updates and to pause, un-pause, and pull them when necessary.

Filtering the information of notifications may involve some computational cost, as stated in the specification of this mechanism [140]. Given that a PS may handle a high number of presence subscriptions, filtering the notifications of all these subscriptions may impact the PS processing resources. Likewise, controlling the notification rate of subsets of each subscription's resource state information may be a resource-intensive task. We are therefore conscious that such a fine-grained rate control may not be viable in large-scale scenarios. Nevertheless, the proposed XML schema can be used to control the rate of presence updates in a scalable way by delegating this process to the presentities themselves, which is mentioned below.

- We tackled the traffic involved in presence publications, which affects all the components of the presence service: the presentity that publishes its presence information, the presentity's PS, all of the presentity's watchers that are authorized to see the published change, and their RLSs if these servers are used. Thus, presence information may degrade the overall presence service's performance. To reduce the number of presence publications sent by presentities, we proposed applying the above-mentioned extended notification filters to presentities, which we refer to as publication filtering. This approach lets presentities know what information is important and when it should be published. Publication filters allows controlling the publication timing by setting minimum and maximum publication rates of presence information, pausing and un-pausing publications, and requesting presentities to publish immediately. Such control of publications can be perform based on particular subsets of presence attributes rather than the complete presence information. This allows optimizing presence traffic on the access link while ensuring that certain presence attributes are updated frequently enough. Moreover, reducing the number of publications alleviates the PS load, and reduces the number of notifications, which in turn optimizes the presence traffic on the network core too. We proposed that the PS let presentities know about their publication filters by means of PUBLISH messages.

As mentioned previously, fine-grained control on the timing of presence notifications at the PS may not be scalable since the PS is likely to handle a high number of presence subscriptions. Since the timing of presence publications determines the timing of presence notifications, the proposed publication filtering implicitly controls the rate of presence notifications. This approach delegates the process of filtering to the presentities themselves, which is a fully-distributed alternative and therefore scalable. Thus, it is possible to control the rate at which presence updates are disseminated over the network in an attribute-based manner. The sacrifice of such scalability is that filtering is performed in a 1:N mode rather than 1:1. The PS creates the publication filter for each presentity according to some traffic optimization policy, and the interests and needs of all of the presentity's watchers. It is therefore necessary to find out a tradeoff between the needs of all of the presentity's watchers. Although this seems to be an arduous task, the presence of watchers is very helpful. For example, the PS may take only the watchers that are online into account to create publication filters.

We modeled presence changes through a continuous-time Markov chain. We used this chain to probabilistically estimate the traffic rate generated by a presentity when its publication rate is controlled by a single maximum value and multiple maximum values associated with different presence attributes. We approximated a maximum rate as a timer set to the inverse of the rate (i.e., throttling interval) that is constantly restarted. This assumption allows calculating the probability of presence changes during each throttling interval. We gave a guideline about how to model presence changes through Markov chains, and took an example chain for estimating presence traffic. The presented mathematical model is valuable since, to the best of our knowledge, there are not other models of presence changes that are as general as it and are studied in continuous time. Since presence information may include a wide variety of information, many kinds of applications may rely on this kind of information such as LBSs, social networks, messengers and so on. Thus, there is not a common pattern of the behavior of presence applications. Even, the presence information of a particular application's users usually do not follow any particular pattern because the actions that users take and affect their presence information (e.g., modifying personal state, mood, activities, location)

are highly subjective and depend on temporary circumstances. This makes the analysis of presence systems specially difficult.

We presented the mathematical formulas that calculate the probability of presence changes occurring during each throttling interval with a single and multiple intervals. From these formulas, we derived the total number and rate of bytes sent on the network access link because of presence publications during part of an application session. The reported results for a single throttling interval showed that minimum intervals of 5, 10, 15, and 30 minutes reduce the traffic of presence publications by 41%, 61%, 71%, and 85%, respectively. These results validated the mathematical model and confirmed reasonable assumptions: It is not recommended to apply a very short throttling interval because it saves few bytes, which does not compensate for the complexity of implementing it at end devices. Moreover, the longer the throttling interval, the largest the traffic saved on the access link. However, the delay perceived by watchers increases with the throttling interval length. When the delay in presence updates acceptable by watchers varies from some presence attributes to others, setting attribute-based maximum publication rates is necessary to ensure that watchers are updated frequently enough. We described an algorithm for calculating the probability of publishing presence changes after each throttling interval when multiple intervals are used. Thus, we calculated the presence traffic rate for multiple scenarios that give attributes different levels of importance. Multiple throttling intervals make it possible to set short intervals for the most important information, while the rest of information is regulated by longer intervals. The efficiency of this mechanism depends on the level of importance that is set for the presence attributes that change most frequently. In the best case, these attributes are not important and therefore can be associated with long throttling intervals. In the worst case, these attributes are the most important and therefore have to be notified at short intervals.

The length of throttling intervals should be chosen carefully because presence attributes that change frequently should not be published at low rates. Otherwise, watchers would keep obsolete presence information for too long and publications would even turn out inefficient. If the presence information changes much more rapidly than the publication rate, the short time during which watchers see valid

information does not compensate for the traffic generated to publish such information. We proposed calculating throttling intervals over time based on a Markov chain that models presence changes, which we refer to as sojourn-based intervals. This avoids setting too long throttling intervals, thereby preventing too long delays in publishing. We analytically estimated the traffic rate generated by sojourn-based intervals and compared it to that generated by predefined throttling intervals. While sojourn-based intervals change over time for adapting to the presence attributes' change frequency, predefined intervals are static. The reported traffic rate showed that applying sojourn-based intervals increases the traffic rate generated by predefined intervals by around 17%. This increase is because sojourn-based rates are higher than the predefined ones in order to publish the presence information that changes the most frequently enough. This shows that sojourn-based intervals can ensure a certain update rate.

- The scalability of SIMPLE is specially challenging in large-scale federation scenarios in which millions of users in a domain subscribe to millions of users in other federated domains. This is mainly due to the fact that two federated domains establish as many presence subscriptions between them as the number of different watcher-presentity interests. Each of these subscriptions is refreshed and kept up to date separately. Whenever a presentity's presence information changes, the presentity's domain sends a different presence document to each subscribed watcher within each federated domain regardless of whether these documents contain the same information. Some proposals for reducing the number of notifications or subscriptions in federation scenarios emerged as IETF Internet-Drafts, namely dialog optimization, CN and VS. We studied these proposals in depth, giving details about their operation and parameters. Based on this study, we analytically estimated the amount of traffic involved by dialog optimization, CN and VS. This study is intended to enlighten the reader about how complex inter-domain traffic optimization is, how the studied strategies operate, what parameters network administrators should take into account for optimizing inter-domain presence traffic and last, but not least, what the sensitivity of each technique to these parameters is. Based on the results of the presented traffic estimation, the conclusions below can be extracted.

Dialog optimization always generates much more traffic than CN and VS. This technique is discouraged because it adds more complexity to the overall presence service, and only reduces inter-domain traffic to a short extent. When subscribers implement conditional notifications, not to apply any inter-domain traffic optimization is preferable to dialog optimization since it adds traffic load. It is recommended to combine CN and VS with conditional notifications, since it greatly helps in reducing presence traffic. When the presentities set the same privacy rules for all the watchers in a federated domain (i.e., domain-based privacy filtering) and the presentities' domain does not require to know the list of watchers subscribed, VS with partial or full trust drastically reduces the inter-domain presence traffic (i.e., by more than 90%). Thus, VS is highly recommended in this case. However, when the presentities' domain needs to be aware of the watchers subscribed (for authorization or security reasons) or presentities set privacy rules to particular watchers (i.e., watcher-based privacy filtering), CN reduces more inter-domain traffic than VS (i.e., by around 60%). The operation and performance of VS is strongly affected by two parameters: the type of trust between the domains and the number of privacy filters. Partial trust always involves a smaller number of bytes than full and minimal trust. The number of privacy filters determines the number of views and, therefore, the number of presence subscriptions in VS with partial and full trust. This is the reason why the efficiency of VS drops considerably as the number of views increases. Although minimal trust generally generates much more traffic than partial or full trust, minimal trust is more efficient when there are numerous views. The reported results showed that when 80% of the watchers have a different view, minimal trust is preferable to partial or full trust. The increase in the number of presence changes has a harmful effects on VS because a single presence change may involve notifying through more than one subscription. Likewise, changes in the presentities' privacy filters may have disastrous effects on VS because a single change may involve modifying, creating or eliminating one or more presence subscriptions. The inter-domain traffic involved by CN is affected by the way the watcher lists are obtained. There are three methods: 1) the notifier domain adds the list to the body of NOTIFY messages, 2) the watcher domain subscribes to the presentities' winfo events and 3) the subscriber side PS maintains the list. Only the methods 1) and 2) involve

presence traffic. The increase in the number of watchers affects the latter more seriously than the former. The main parameter that affects the first method is the number of presence changes per presentity. We advise that the presentities' average activity be considered in making a choice between one of these two methods. In general, when presence changes occur very frequently, the second method is more efficient than the first.

Moreover, we studied the impact of CN and VS on the IMS capacity. We estimated the number of SIP messages that the IMS brain, the CSCF, has to process for implementing these strategies. The reported results showed that CN and VS add relevant capacity demands on the CSCF. Only when no privacy filtering is performed and the notifier domain does not need to know about the watchers subscribed, VS with partial or full trust is preferable to CN since it adds less load to the CSCF. Regarding CN, the afore-mentioned methods 1) and 2) for obtaining the watcher lists have the important advantage that they do not require an intermediate server in the watchers' presence subscriptions. Thus, CN combined with either of these methods injects fewer messages into the CSCF than VS. Since method 2) generates extra messages because of winfo subscriptions, we recommend the use of method 1), that is, the notifier domain adds the list to the body of NOTIFY messages.

- The need to make presence federation scenarios scalable still remains for the future large-scale converged networks. Nevertheless, the IETF has not yet found the standardization of inter-domain presence traffic optimization necessary. Thus, the only proposals for reducing inter-domain presence traffic, namely CN and VS, have been discontinued in the IETF. The fact that SIMPLE, the standard protocol for presence in the IMS, is no longer involved in reducing inter-domain presence traffic should alarm both the industry and academia. We, therefore, propose the strategies CS and FCS for reducing inter-domain traffic. These strategies reduce the number of inter-domain presence subscriptions up to one per presentity, regardless of the number of subscribed watchers. We analytically estimated the presence traffic involved by CS and FCS and compared it to that generated by CN and VS. The reported results showed that CS is considerably more efficient at reducing presence traffic than VS and CN. The only exception happens when

domain-based privacy filtering is performed and the presentities' domain does not require to know the list of subscribed watchers. In this case, VS with partial or full trust is preferable. FCS is preferable to CS only if conditional notifications are applied, which saves 3% of the CS traffic.

CS and FCS rely on the fact that the notifier domain lets the subscriber domain know about the presentities' privacy filters (i.e., authorization rules set by presentities) through subscriptions. Subscriptions to privacy filters account for a considerable part of CS and FCS traffic, and notify sensitive information. Thus, we enhanced CS and FCS by reducing the number of privacy rules that are disclosed. We analyzed the variables that affect the traffic related to privacy filters in depth. We analytically estimated the number of bytes related to privacy filters that is generated by two federated domains that apply CS and FCS with and without the proposed enhancements. The reported results showed that the enhanced FCS saves between 45% and 81% of the traffic related to privacy rules in the regular FCS. The enhanced CS saves between 18% and 60% of the privacy-filters subscriptions' traffic in the regular CS. We studied the effect of conditional notifications on our proposals. This optimization greatly reduces the traffic related to privacy rules, and is therefore strongly recommended in both FCS and CS. The application of conditional notifications to the enhanced FCS is always recommended. However, in the case of CS, the reported results showed that when more than half the presentities' privacy rules become active during the presence session, the enhanced CS with conditional notifications performs worse than the regular one with conditional notifications. In this case, the regular CS is preferable to the enhanced CS.

Moreover, we studied the impact of CS and FCS on the main IMS server, that is, the CSCF. These strategies as they stand increase the CSCF workload. However, when CS and FCS are combined with an RLS, the S-CSCF message load is decreased by approximately 29%. Since the RLS by itself increases the CSCF workload to a large extent, CS and FCS combined with an RLS are not only helpful in optimizing inter-domain presence traffic but also ease the impact of the presence service on the IMS.

On the basis of the presented study, we concluded that the proposed enhancement of FCS combined with conditional notifications is strongly recommended to save inter-domain presence traffic as far as possible. The main drawback of FCS is that the process of privacy filtering must be delegated to the subscriber side PS. This entails the notifier domain giving in the presentities' complete presence information to the subscriber domain. Nevertheless, the fact that two domains exchange presence information and allow their users to communicate is an indication that some degree of trust relationship exists between them. Some kind of trust relationship must always exist since the notifier domain trusts the subscriber domain to distribute the right presence documents to the right watchers. Thus, delegating privacy filtering to the subscriber side domain would simply mean an extension of an existing trust required for the sake of scalability. As regards the interoperability of privacy rules, PSs only need to exchange the rules encoded as SIMPLE authorization rules [177] regardless of their low-level implementations of privacy filtering.

- Although the RLS is widely adopted to reduce presence traffic in mobile presence applications, its performance has not been studied in depth yet. Furthermore, some studies raise doubts about how effective and suitable the RLS is. The authors of [193] concluded that dialog optimization, which consists in a federated RLS, only helps in reducing presence traffic when it is combined with conditional notifications. An analytical study presented in this thesis showed that a federated RLS adds traffic load to the network if compared when watchers subscribe directly and apply conditional notifications, as described above. An estimation of the impact of the RLS on the IMS CSCF presented in this thesis showed that the RLS seriously increases the CSCF workload if it is not optimized with other techniques. Thus, the reduction in presence traffic on the network access achieved by the RLS may not compensate for its impact. We estimated the presence traffic over the access link when a user subscribes to an RLS and his or her presentities directly. We studied these two alternatives with and without two optimizations: partial-state documents and conditional notifications. The reported results showed that the RLS is not efficient at reducing presence traffic. An RLS without any optimization is strongly discouraged. The performance of the RLS

decreases dramatically as the number of presence changes increases. Partial-state presence documents ease the impact of presence changes on the RLS. However, having an RLS handle each presentity's partial-state changes involves some computational cost. Even when the RLS is optimized with the two above-mentioned strategies, subscribing to presentities directly is more efficient under some circumstances. If subscribers and notifiers implement conditional notifications and partial-state documents, direct subscriptions are always more efficient than an optimized RLS except when presentities only change their presence once per hour on average. Even when direct subscriptions are not optimized at all, if presence changes occur frequently and presence documents are small, which is probable in some LBSs, direct subscriptions are more efficient than an optimized RLS. Network administrators should carefully study three parameters before configuring watcher applications to subscribe to an RLS: average size of resource lists, number of presence changes, and size of presence documents.

- Fine-grained rate control of presence notifications is not scalable in large-scale federation scenarios. This involves the PS handling multiple notification buffers, one for each different maximum notification rate for each watcher. Given that large-scale scenarios involve millions of presence subscriptions, this mechanism is obviously not viable. We proposed a queuing system at the PS for reducing presence traffic in large-scale federation scenarios that apply CS or FCS. Thus, the proposed system relies on the fact that the PS handles only one presence subscription per federated domain. This system is designed to controlling the rate of both presence notifications and publications in a scalable way. The PS only handles a notification buffer for each watcher domain, and delegates publication control to the presentities themselves. We mathematically modeled the proposed queuing system, and obtained its main probability features. We proposed an adaptive algorithm for changing publication and notification rates based on the watcher domains' needs on traffic optimization and information consistency. This algorithm allows the PS to adapt its behavior to the watcher domains' QoS needs as long as the probability of the PS getting saturated is kept under a threshold. This ensures a maximum delay in notifying each watcher domain while limiting

the rate of presence publications and notifications and ensuring the PS performance is not degraded. Based on the presented mathematical model, we studied the performance of the proposed adaptive algorithm with different QoS parameters. This algorithm effectively adapted both the notification rate for a watcher domain and the publication rates of the presentities to which this domain is subscribed for ensuring that presence change notifications are not delayed longer than a maximum delay set by this domain. Moreover, we discussed how sojourn-based throttling intervals can be used to avoid controlling the publication rate inefficiently.

- Figuring out the adequate applications that will motivate customers to use always-on valued-added services is crucial for the success of NGNs. We have developed "Sense Everything, Control Everything" (SECE), a platform for context-aware service composition based on user-defined rules. SECE rules trigger action scripts whenever events of interest occur such as communication requests, context changes, location updates, time events, etc. SECE relies on a SIMPLE-complaint PS that handles all the users' presence information, which includes personal information, device capabilities, location information and even information about sensor networks at the user's office or home. SECE differs from other rule-based systems in that it provides an interface for creating rules in natural-English-like language commands. With a simplified English-like interface to creating rules, users will be more prone to incorporate rule-based systems into their lives, making context-aware computing a seamless part of everyday life.

  Moreover, we improved SECE to use the semantic description of service domains to dynamically create a rule language for these service domains. We modify SECE architectural platform to integrate with a back-end ontology-based global service discovery system, GloServ, to access any type of service domain within the GloServ directory [230] [231]. With these improvements, SECE can now be generalized to include all types of service domains, described in an ontology, as well as issue more complex ontology-based queries for service discovery and composition. Having the ability to adapt a rule language to new service domains makes SECE into a powerful front-end context-aware system. Additionally, by using GloServ as its back-end, SECE can now interoperate with any type of service

that has an ontology description, broadening its scope drastically. We envision that SECE will enable services to seamlessly integrate into people's lives. A person can now create rules with ease and be notified of services at the right time and place. This will create a profound impact in how people interact with services. There will now be a closer connection between a person and services available, establishing a personalized network of services.

- We proposed and implemented a decentralized architecture for handling SIP/SIMPLE presence information and user communications in an efficient, user-personalized way. This solution is a logical, user-centric presence and multimedia functionality that may be added to a home gateway (or even a femtocell) as a value-added service. We call such functionality Personal Proxy (PP). This approach avoids a centralized PS that may easily become a bottleneck. A user's personal proxy works as the unique contact point with the user, which provides more security and flexibility for user personalization. PP allows users to set up fine-grained preferences about presence information sharing, privacy, and traffic optimization in a scalable way. This approach allows centralizing user access to certain Internet services, such as HTTP or VoIP, in a single point. Presence-based, user-centric personalization of these services paves the way for maximizing the users' QoE. SIP-based applications might be built on top of PP, and use its functionality to adapt their behavior efficiently. Thus, we present the personal proxy as an enabler of user personalization, which may be combined with proactive communication services that require advanced context and presence information processing. In particular, PP would improve the scalability of the presented SECE platform. Since this platform is aimed to handle as much context as possible, which includes sensor information, for each watcher, distributing presence handling among each user's personal proxy would upgrade the overall system's scalability. We envision the PP functionality integrated into the presented SECE gateway for handling sensor information. This would reduce the number of nodes to which the user's context is given in, thereby improving user privacy in SECE.

  PP is a middleware designed to optimize presence traffic and user communications. This implements presence traffic optimizations emerged from the IETF SIMPLE working group, such as notification filtering and partial-state documents.

Moreover, the PP client and sever parts can perform non-standard optimization techniques for reducing traffic on the air interface as much as possible when it is necessary. We analytically estimated the amount of presence traffic on the access downlink and uplink reduced by PP; thus, we concluded that PP can help in reducing this traffic. Moreover, PP decreases the response time perceived by the user when navigating the Web by means of an HTTP cache, a DNS cache, content compression and HTTP header reduction. The last one requires the client and server parts of the middleware to collaborate with each other. We experimentally measured the response time of HTTP requests when the user's end device is configured to use the user's PP as proxy. The reported results showed the PP is very helpful in decreasing the response time of HTTP requests.

Based on the above-mentioned contributions, we have several on-going and future lines of work. We will study about publish/subscribe systems in depth, specially their capability to incorporate presence features and be integrated into sensor networks. This study will include an analysis of the functionality, flexibility, extensibility, and performance on restricted networks of several publish/subscribe systems. As part of this study, we will analyze the applicability of the SIP/SIMPLE framework on constrained networks, which could result in optimizing this framework for this type of network or designing a mixed framework that combines SIP/SIMPLE with more optimized publish/subscribe protocols. We will continue working on SECE, specially on its aspects related to sensor networks and nearby services (e.g., museums, bars, shops, etc.). We will continue developing and enhancing PP. Moreover, we will update some of the PP's functions that have become obsolete with some advances in IETF WG (e.g., EXI and SIMPLE conditional notifications). We will implement the proposed strategies for reducing inter-domain presence traffic, namely CS and FCS. Thus, we will evaluate these strategies' performance through a simulated IMS environment. Last, but not least, we will study the applicability of the proposed queue system at the PS in more detail and evaluate its performance on a simulated environment.