Ontology-Based User-Defined Rules and Context-Aware Service Composition System

Victoria Beltran[1], Knarig Arabshian[2], and Henning Schulzrinne[3]

[1] Dept of Telematics, Universitat Politècnica de Catalonia/Fundació I2Cat, Barcelona, Spain
[2] Alcatel-Lucent Bell Labs, New Jersey, USA
[3] Dept of Computer Science, Columbia University, New York, USA

Abstract. The World Wide Web is becoming increasingly personalized as users provide more of their information on the Web. Thus, Web service functionality is becoming reliant on user profile information and context in order to provide user-specific data. In this paper, we discuss enhancements to SECE (Sense Everything, Control Everything), a platform for context-aware service composition based on user-defined rules. We have enhanced SECE to interpret ontology descriptions of services. With this enhancement, SECE can now create user-defined rules based on the ontology description of the service and interoperate within any service domain that has an ontology description. Additionally, it can use an ontology-based service discovery system like GloServ as its service discovery back-end in order to issue more complex queries for service discovery and composition. This paper discusses the design and implementation of these improvements.

Keywords: context-aware systems, ontologies, semantic web, rule-based systems, service discovery, service composition, web services.

1 Introduction

In recent years, the World Wide Web has been advancing towards greater personalization. Services on the Web such as, social networking, e-commerce or search sites, store user information in order to profile the user and target specific products or ads of interest. Since web service functionality is increasingly relying on user information, a user's context is becoming more crucial towards creating a personalized set of services within the Web.

As these types of services proliferate, a framework is needed where multiple services can be discovered and composed for a particular user within a certain context. With this in mind, we have developed SECE (Sense Everything, Control Everything), a platform for context-aware service composition based on user-defined rules. The contributions to SECE are two-fold: a user-friendly rule language and the design and implementation of a context-aware service composition framework.

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.

Additionally, 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 subscribe to various services by formulating simple rules that create a composition of services. The rules trigger event-based service discovery and 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. For more information on the SECE architecture and rule language, we encourage the readers to refer to the following paper [1].

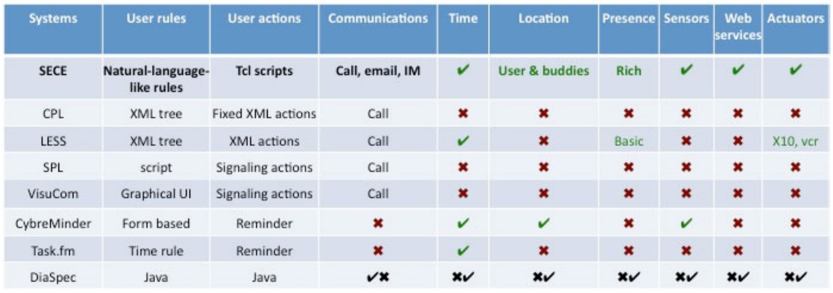
In this paper, we discuss enhancements to both aspects of SECE: its rule language and back-end architecture. Whereas previously SECE had a hard-coded rule language for a limited number of event-based service domains, we have now improved SECE to use the Web Ontology Language (OWL) description of a service domain to dynamically create a rule language for that service domain. Additionally, SECE's architectural platform has been modified to integrate with a back-end ontology-based global service discovery system, GloServ, to access any type of service domain within the GloServ directory [2] [3]. GloServ classifies services in an ontology and provides ontology descriptions of different service domains. It also has an ontology-based query interface for service discovery and composition.

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 OWL 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.

The organization of this paper is as follows: Section 2 describes current work in the field of context-aware computing and service composition; Section 3 gives an overview of the original SECE architecture and functionality; we discuss the enhancements to SECE and its implementation in Section 4; Section 5 discusses future work; finally, Section 6 summarizes the main contributions of this paper.

2 Related Work

Several solutions for user created services have been proposed; some of these solutions are compared to SECE in Figure 1. 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. CPL [4], LESS [5], SPL [6], VisuCom [7] and DiaSpec [8] are attempts to allow end users to create services, but they are all limited to controlling call routing. Also, CPL and LESS use XML and, hence, even 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. Writing a specification in DiaSpec and then developing a service using the generated framework is definitely not suitable for non-technical end users. The authors of DiaSpec extended [9] their initial work to support services beyond telephony, which include sensors and actuators. However, it is still only suitable for advanced developers. SPL is a scripting language which is suitable for end-users but only for telephony events. VisuCom has the same functionality as SPL, but allows users to create services visually via GUI components.



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	✓✗	✗✓	✗✓	✗✓	✗✓	✗✓	✗✓

Fig. 1. Comparison to related work

CybreMinder [10] is a context-aware tool which allows users to setup email, SMS, print out and on-screen reminders based not only on time but also location and presence status of other users. It uses local sensors to detect a user's location. It does not take any actions, but rather displays reminders to the end user. Also it is not as powerful as scripting-based systems due to its form-based nature. Task.fm [11] is a similar SMS and email remainder system which uses natural language to describe time instants when email or SMS reminders will be sent. However, Task.fm only supports time-based rules and does not include information from sensors. This tool does not take actions other than reminding users via SMS, email or phone call.

Regarding composition of web services, SWORD [12] was one of the first prototypes. However, this offers a quite limited composition that is not automatic and its scripting language is targeted at developers. Ezweb [13] is a graphical tool by which users can connect web services manually. However, this does not provide automatic web service discovery or a language for composing services. Moreover, service composition is not context-aware and proactive. Yahoo Pipes [14] is other graphical tool for web service composition. However, it presents the same limitations as Ezweb and its graphical interface is not really easy-to-use and intuitive, which makes it very difficult for non-technical users. We only found a prototype described in a research paper [15] that offers event-based web service composition. This means that service composition is triggered by events, such as changes in the user's context, instead of end users. However, this work does not provide any language or tool for specifying the web service compositions and events that trigger them. The authors seem to implement low-level compositions that may be personalized according to user preferences. Thus, this does not offer end users control of service composition. Moreover, this prototype seems not to be available in the Internet.

To the best of our knowledge, there is no implemented platform for allowing end users to compose services of different kind based on events. The current solutions are not proactive because the end-user is who triggers the composite services or only provides template-based compositions (i.e., the user is not who defines the compositions). There is neither a platform for event-based web service discovery. The composition tools that take user context into account, only consider a limited set of context. The graphical interfaces of the studied tools are quite limited and not flexible for non-technical users. The scripting languages provided by some tools are neither suitable for non-technical users and only support a limited set of context information. Moreover, none of the studied tools proactively discover web services based on the user preferences.

3 SECE

SECE is a rule-based context-aware system that connects services, that may have otherwise been disconnected, to create a personalized environment of services for a user. It has two fully-integrated components: user-defined rules in a natural English-like formal language and a supporting software architecture. Users are not required to continually interact with the system in order to query for or compose a set of services. They need to only define rules of what they want to accomplish and SECE does the rest by keeping track of the user's context, as well as information from external entities such as sensors, buddies, or social events in order to notify the user about a service. It accomplishes this by communicating with several third party applications and web services such as Google services (e.g., GMail, GContacts and GCalendar), Social Media services (e.g., Facebook or Twitter), VoIP proxy servers, presence servers, sensors and actuators. Figure 2 gives an overview of the overall SECE architecture and how it interacts with its environment. We will discuss these two components of SECE in this section.

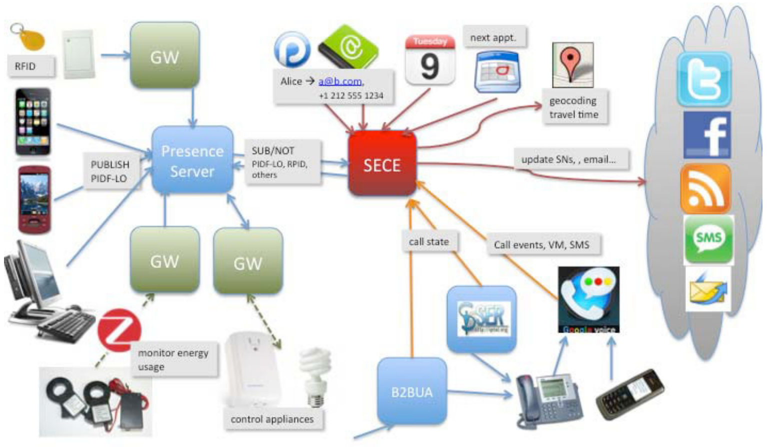


Fig. 2. SECE and its external components

3.1 SECE Architecture

As Figure 2 depicts, 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 rule 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 3.2 explains the SECE rules and actions, and their required services in more detail.

We are developing two services that tightly collaborate with SECE: the presence server and the VoIP proxy server. The presence server is built on the Mobicents Presence Service [16], which is compliant with SIMPLE (SIP for Instant Messaging and Presence Leveraging Extensions) [17]. It is responsible for collecting and aggregating context from different sources and sending it to SECE. It accomplishes this by receiving presence publications from context sources that contain the latest information about a user and, in turn, notifying 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 RFID. To use the presence service, the end user needs to create an account from the SECE website in order to obtain the SECE presence server's access information. Thus, the user can configure the SIMPLE-compliant presence applications (e.g. SIP Communicator and Pidgin) that run on her mobile devices or desktop computers to use the SECE presence server. In the future, the presence server will interact with the home gateway for obtaining information from sensor networks and changing the state of actuators.

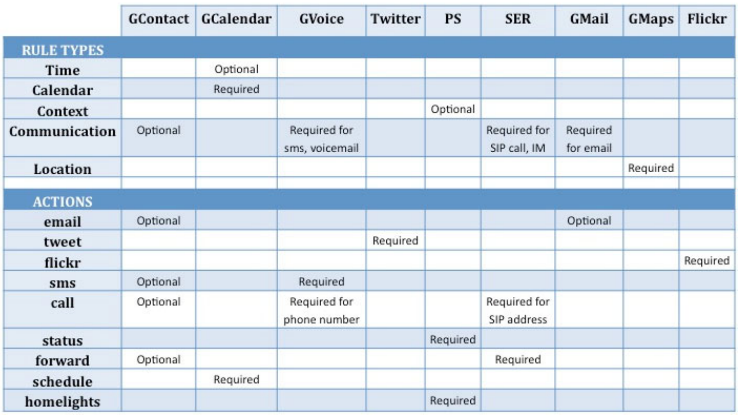
The VoIP proxy server is a SIP (Session Initiation Protocol) Express Router (SER) [18] extended to interact with SECE for handling users' SIP communications. This server and SECE implement an efficient binary protocol that lets SER inform SECE of a SIP event and lets SECE notify SER of the action to take for this event. Basically, SER informs SECE of users' incoming and outgoing calls and instant messages (IM). If an event of this kind matches a rule, the rule is triggered and, therefore, decides to either forward, reject, or modify the call by invoking an action. Then, SECE will let SER know about the action to take. As the presence service, 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. A first prototype of SECE has already been developed as a web service and is being tested by members of the Internet Real Time (IRT) group at Columbia University. For a more detailed description of the SECE architecture, we refer the readers to the following paper [1].

3.2 SECE Rules

The SECE language supports 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 and actions interact with different third party services based on their subscribed events and functions. Thus, SECE users need to learn about the services needed by the rule types and actions that they wish to use and configure their SECE accounts for such services. SECE will provide online documentation that gives users information 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 3 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 [19]. We use the Tcl language [20] as the syntax for the rule actions. This choice is due to Tcl's extensibility that allows adding new commands to its core with relative ease. Tcl provides a command that receives the name, arguments and code of a new command as parameters, constructs the corresponding Tcl command and incorporates it into the Tcl interpreter. Below, we 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 and the language keywords are italicized.

Time Rules: Below are two types of rules: single time events and recurrent time events. The former starts with the keyword *on* and the latter starts



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

Fig. 3. Third party services of SECE rules and some actions

with the keyword *every*. Both are fully-compliant with the Internet Calendar (ICal) standard [21]. The *on*, *until*, *except* and *including* keywords are always followed by a date expression that can have different formats (e.g., "December 31, 2011", "31st day of December, 2011" and "12/31/2011") or can be an entry in the user's GCalendar. In the first example below, the user defined an entry named "Anne's birthday" in her 2011 GCalendar.

```
on Anne's birthday, 2011 at 12:00 in Europe/Zurich {
   sms Anne "Happy Birthday!!! John";
}
every week on WE at 6:00 PM from 1/1/11 until May 10, 2011
except 3rd WE of Feb, 2011 including first day of June, 2011 {
   email irt-list "reminder: weekly meeting today at 6:00 PM";
}
```

Calendar Rules: These rules specify events that are defined in the user's 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.

```
when 30 minutes before "weekly meeting" {
      email [event participants] "The weekly meeting will start in 30 minutes";
      if {{ ! my location 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.";
      }
}
```

Location 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 his friends such as a nickname, email and SIP address. Five types of location information are supported: geospatial coordinates, civic information, well-known places, user-specified places and user locations. Different location-related operators can be used, such as *near*, *within*, *in*, *outside of* or *moved*. Below we show a location rule using the *near* operator. *Within* means that the user is within a radius of the reference point. *Near* means the same but the radius is a default distance that the user defines in his SECE account.*Outside of* and *in* means that the user is outside of and inside the reference point, which must be represented as a polygonal structure. We are working on a location database that allows users to predefine polygonal locations through a GMaps-based graphical interface. *Moved* means that the user moved the given distance from where he was located when the rule was entered or triggered for the last time.

```
Bob near "Columbia University" {
    if{ my status is idle } { call bob; }
}
```

Context Rules: These 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 if followed by *my*. Otherwise, the *if* keyword is followed by the friend's identifier. Below, we show an example of a context rule about a friend.

```
if Bob's status is available { alarm me; }
```

Communication Rules: These specify the actions to execute in response to incoming, outgoing or missed communication requests. A request rule can start with the keyword *incoming*, *outgoing* or *missed*, followed by the type of event. The following rule is an example of incoming call handling.

```
incoming call to me.phone.work {
    if { [ my location is not office] } {
        autoanswer audio no_office.au;
        email me "[incoming caller] tried to reach you on your work phone at
        [incoming time]";
    }
}
```

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, an ontology-based service discovery system, within SECE's back-end architecture. 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 ontology to describe its services. Thus, SECE can access these OWL specifications in order to dynamically define rules for the specific service domain. Users are made aware of these services by a front-end application to SECE that displays the discoverable services' descriptions. For each service domain, SECE will provide documentation on how to create rules. Currently, users will 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. This section will describe the design and implementation of these enhancements.

4.1 SECE Architecture

Design. Figure 4 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 graphical interfaces. 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 in SECE on, SECE will periodically communicate with GloServ for discovering the web services that match the rule. A GloServ request specifies the web service of interest as a SPARQL query [22] and matched services' profiles, if any, are sent to SECE into a GloServ response. If a new web service matches a rule, SECE executes the rule's body.

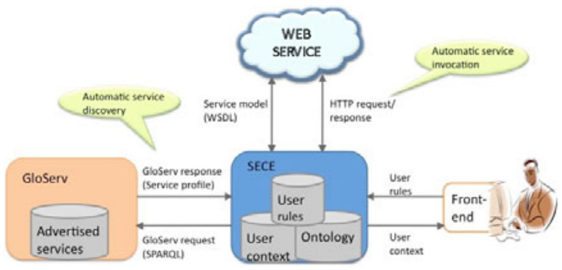


Fig. 4. SECE, GloServ, front-end applications and web services

SECE has a layered architecture, as Figure 5 shows. For details of each of the components, we encourage the reader to refer to the original SECE paper [1]. We will discuss the components that have been added to the enhanced SECE architecture in this section.

The new components that have been added to the SECE architecture are: 1) *WBRL* rule, which implements the web service rules; 2) Jena Ontology Model, which contains the necessary ontologies' schemes; 3) GloServ Context Mediator, which periodically pulls GloServ for checking out new web services of interest.

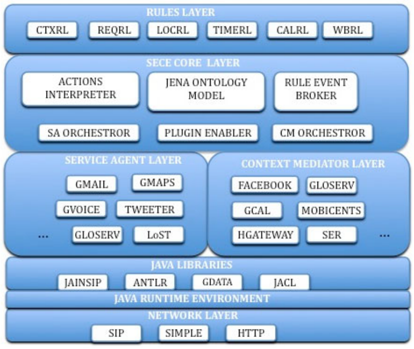


Fig. 5. SECE architecture

Implementation. SECE stores the OWL specifications of web services in an ontology database that is built upon the Jena Framework [23]. When a web service rule is entered into SECE, it has to go through 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 6 outlines the main interactions for creating a web service subscription.

The SECE core coordinates the software components in SECE. First, the SECE parser checks that the input rule is consistent with the SECE language, which is generated by an ANTLR grammar [19]. 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. The web service event is defined by the service name and optionally a set of property constraints in the form of *(propertyName, operator, value)*. If the rule parsing is successful, the SECE core verifies that the rule's web service description corresponds to a web service's ontology. To do it, this interacts with the SECE Ontology Model (i.e., *SECEOntModel* in Figure 6). 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 a web service's ontology that describes a service that is named as the described web service and can be associated with the described properties and constraints. Thus, SECE will ask the SECE Ontology Model for the namespace URI of the web service and its properties. If this 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 the SECE core proceeds to create the corresponding subscription (i.e., *WSSubs* in Figure 6).

The SECE core then retrieves an event monitor from the Event Monitor Broker (*OntEM* and *EMBroker* in Figure 6). An event monitor is the agent that watches a particular service and generates an event whenever a new instance of this service is discovered. The Event Monitor Broker maintains a list of the event monitors that are actually monitoring a web service. 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, the SECE core associates the event subscription with the event monitor and starts the subscription.

Starting and pausing an event subscription makes it 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. This also starts up a recursive timer to query the GloServ Context Mediator (i.e., *GloServCM* in Figure 6) at fixed intervals with the SPARQL query. If this query results in any matched service, the event monitor creates an OntEvent object that describes the discovered service and notifies the subscriber of this event. Note that the outbound messages between GloServCM and GloServ are omitted in Figure 6 because of lack of space. 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 the service matches any of the other subscriptions. Figure 6 only shows this check on the web service subscription *wss* through the *matchedServ* method. Furthermore, the event monitor maintains a cache of discovered events. When a new subscription is created, this cache is checked out and the matching web services are notified.

4.2 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's 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 strings 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

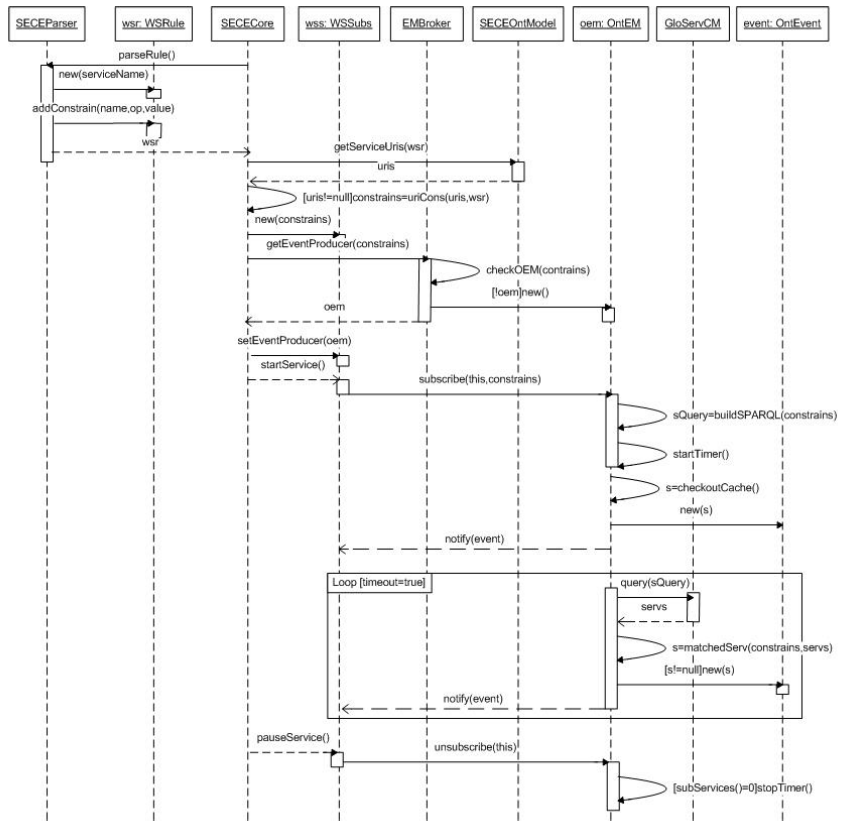


Fig. 6. Sequence diagram from entering a web service rule to querying GloServ

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 types, 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 class types restrict inherited types as for example "any restaurant that is southamerican" subscribes to restaurants that are subclasses of the *southamericanRestaurant* class.

5 Future Work

Integrating web service rules into SECE brings out many possibilities in the Semantic Web. Sections 5.1, 5.2 and 5.3 briefly introduce some of these possibilities.

5.1 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. To this end, the SECE web service will provide an API.

5.2 Error Detection

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.

5.3 Event-Based Context-Aware Web Service Composition System

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 is static and the communication 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. 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* 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 could perform semantic web service discovery and composition that does not need user interaction to be executed; it is automatically triggered by events. In addition, this would also 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 him or 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 1 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.

For dynamic compositions, SECE will interact with web services automatically, by retrieving their models and, according to their WSDL specifications, constructing HTTP requests.

Table 1. Types of SECE composition

	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 *(current contribution)*	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

6 Conclusions

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, domain-specific or lack some desired feature. 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. To face all these needs, we present a context-aware, event-based platform for service discovery and composition. This platform results from integrating two existing solutions: SECE and GloServ. SECE is a user-centric, context-aware platform for service composition that provides a natural-English-like language for creating event-based rules. GloServ is a scalable network for web service discovery. 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 to create 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 discovered web services and 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. SECE can be decoupled from front-end applications so that more fancy graphical interfaces can be built on top of it. Modeling SECE rules ontologically can provide front-ends with the means of understanding and learning new SECE rules automatically.

Acknowledgments. Victoria Beltran was supported by the Spanish Government through the CICYT project TIC2009-11453 and the FPU grant AP2006-02846.

References

1. Boyaci, O., Beltran, V., Schulzrinne, H.: Bridging communications and the physical world: Sense everything, control everything. In: Proceedings on the IEEE Globecom (UbiCoNet Workshop) (December 2010)
2. Arabshian, K., Schulzrinne, H.: An ontology-based hierarchical peer-to-peer global service discovery system. Journal of Ubiquitous Computing and Intelligence 1(2), 133
3. Arabshian, K., Dickmann, C., Schulzrinne, H.: Service composition in an ontology-based global service discovery system. tech. rep. Columbia University, New York, NY (September 2007)
4. Rosenberg, J., Lennox, J., Schulzrinne, H.: Programming Internet telephony services. IEEE Internet Computing 3, 63–72 (1999)
5. Wu, X., Schulzrinne, H.: Programmable End System Services Using SIP. In: Conference Record of the International Conference on Communications (ICC) (May 2003)
6. Burgy, L., Consel, C., Latry, F., Lawall, J., Palix, N., Reveillere, L.: Language Technology for Internet-Telephony Service Creation. In: IEEE International Conference on Communications, ICC 2006, vol. 4, pp. 1795–1800 (June 2006)
7. Latry, F., Mercadal, J., Consel, C.: Staging telephony service creation: a language approach. In: IPTComm 2007: Proceedings of the 1st International Conference on Principles, Systems and Applications of IP Telecommunications, pp. 99–110. ACM, New York (2007)
8. Jouve, W., Palix, N., Blum, A., Kadionik, P.: A SIP-Based Programming Framework for Advanced Telephony Applications. In: Schulzrinne, H., State, R., Niccolini, S. (eds.) IPTComm 2008. LNCS, vol. 5310, pp. 1–20. Springer, Heidelberg (2008)
9. Cassou, D., Bertran, B., Loriant, N., Consel, C.: A generative programming approach to developing pervasive computing systems. In: GPCE 2009: Proceedings of the Eighth International Conference on Generative Programming and Component Engineering, pp. 137–146. ACM, New York (2009)
10. Dey, A.K., Abowd, G.D.: CybreMinder: A Context-Aware System for Supporting Reminders. In: Thomas, P., Gellersen, H.-W. (eds.) HUC 2000. LNCS, vol. 1927, pp. 172–186. Springer, Heidelberg (2000)
11. task.fm Free SMS and Email Reminders, `http://task.fm`
12. Ponnekanti, S., Fox, A.: Sword: A developer toolkit for web service composition. In: Proc. of the Eleventh International World Wide Web Conference, Honolulu, HI (2002)
13. Soriano, J., Lizcano, D., Hierro, J., Reyes, M., Schroth, C., Janner, T.: Enhancing user-service interaction through a global user-centric approach to SOA. In: Fourth International Conference on Networking and Services, ICNS 2008, pp. 194–203. IEEE (2008)
14. Yahoo pipes, `http://pipes.yahoo.com/pipes/`

15. Kazhamiakin, R., Bertoli, P., Paolucci, M., Pistore, M., Wagner, M.: Having Services "YourWay!": Towards User-Centric Composition of Mobile Services. In: Domingue, J., Fensel, D., Traverso, P. (eds.) FIS 2008. LNCS, vol. 5468, pp. 94–106. Springer, Heidelberg (2009)
16. Mobicents, `http://www.mobicents.org/`
17. SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE), `http://datatracker.ietf.org/wg/simple/charter/`
18. About SIP Express Router, `http://www.iptel.org/ser/`
19. Parr, T.: The Definitive ANTLR Reference: Building Domain-Specific Languages. Pragmatic Bookshelf (2007)
20. Ousterhout, J.K., Jones, K.: Tcl and the Tk Toolkit, 2nd edn. Addison-Wesley, Upper Saddle River (2009)
21. Desruisseaux, B.: Internet Calendaring and Scheduling Core Object Specification (iCalendar). RFC 5545 (Proposed Standard), Updated by RFC 5546 (September 2009)
22. W3C, SPARQL Query Language for RDF. Website (January 2008), `http://www.w3.org/TR/rdf-sparql-query/`
23. Jena - A Semantic Web Framework for Java, Website, `http://jena.sourceforge.net/index.html`