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 [].

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. 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 availalbe, 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; Sections **??** gives an overview of the original SECE architecture and functionality; we discuss the enhancements to SECE and its implementation in Sections **??** and 4.1; finally, Section 7 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. CPL [1], LESS [2], SPL [3], VisuCom [4] and DiaSpec [5] 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 [6] 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	NL-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 [7] 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 [8] 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 [9] 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 [10] 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

[11] 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 [12] 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, in order 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.

3.1 SECE Architecture

As mentioned above and as Figure 2 depicts, SECE is connected to a number of external services. The Presence Server (PS), which is based on SIMPLE [13],

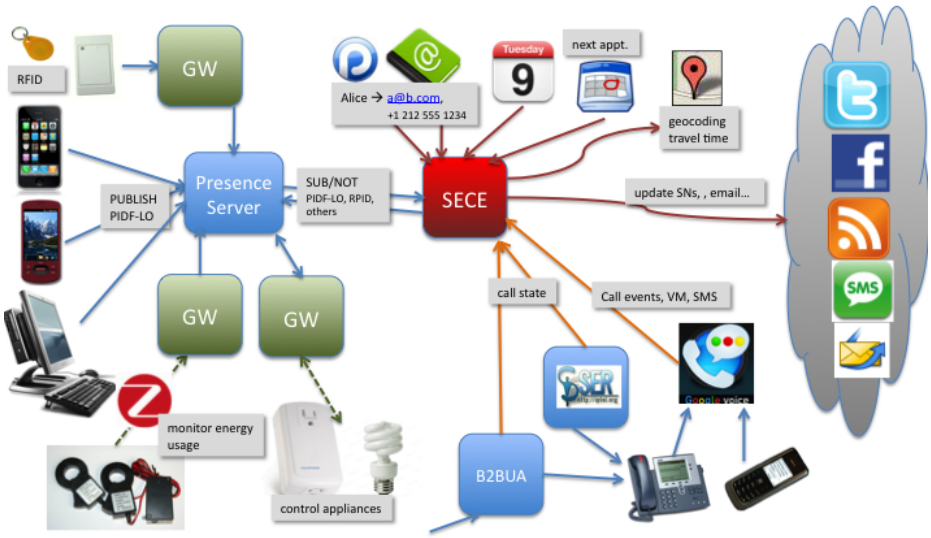

Fig. 2. SECE and its external components

plays a key role in collecting context from different sources. The PS receives presence publications from context sources that contain the latest information about a user and, in turn, notifies 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.

Another external server that plays a key role in SECE is the SIP Express Router (SER) [14], which handles SIP communication. SER informs SECE of incoming and outgoing communication notifications, such as a call or IM. The notification that matches a user-defined rule invokes an action to either forward, reject, or modify the call. An initial 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 SECE, we refer the readers to the following paper [15].

(????????? NO description on how these services are found/incorporated into SECE. how does this system boot up? the part where it says it is connected to a number of external services–how are these found and connected. one paragraph or a few sentence description is necessary)

3.2 SECE rules

A SECE rule is divided into two parts: event description (condition???) and actions. The event description defines the conditions that need to be satisfied to execute the actions. The SECE language is a formal language similar to English that has been designed to be easy to use and remember by end-users. This

language is only intended to define events in the first SECE prototype although it could be extended in the future. The SECE interface provides documentation which give examples on how to create rules for specific events. Users learn the rules and create rules accordingly.

The SECE language supports five types of rules. 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". Some commands are for a particular set of events. For example, "accept" and "reject" commands are only used in request-based rules.

We use the Tcl language [16] as the syntax for the rule actions. This choice is due to Tcl's extensibility that allows adding new commands to its core in an easy and convenient way. (a bit more description on how this is done–one or two sentences ????) Moreover, SECE (event language???) is not coupled with the action language and, hence, other scripting languages, such as Ruby or Python, may be added in the future.

Below, we list the five types of rules supported by the SECE language. In order to clearly display the structure of the rule language, the variables that are set by the user are highlighted in bold and the keywords of the rule language are italicized.

Time-based rules: These support single and recurring time events and are fully-compliant with the Internet Calendar (ICal) standard [17]. Two examples are:
(????????????
"Anne's birthday, 2010" how does SECE know what this is? what is it looking for and where? the term "Anne's birthday" in the 2010 calendar that the user registered, such as his google calendar?
there is no indication of how the event description verses the action rule is constructed. how do i know what to construct? you're not indicating which part of the actions are keywords verses user-defined. is "sms" a keyword? do you know that after the word "sms" you are expecting a user name? where will it find Anne? does it go to the contacts? some explanation is necessary. this is very very vague..
i suggest you take ONE type of rule and give a complete description of it 1) key words of event description verses action. what's the separation? how do users know they are supposed to write these words. since the rule is the biggest selling point of SECE you have to make these things very clear 2)describe end to end flow of logic. how is rule getting parsed? for the time rule below, what is happening?
when it sees the word "on" it knows it's a time rule? or is the user setting this as a time rule somewhere before even defining it?
after the word "on" it expects a variable. variable says "Anne's birthday 2010". what is it looking at in this sentence? is this user-defined variable have some kind of structure specified by SECE? [user][event][time] OR is it just a string "Anne's brithday 2010"? how is this variable processed by SECE? then it encounters "at" so now it's expecting a time? is it in hours

or a date? then it encounters "in" so after that it is expecting a location "Zurich". then it sees the braces and knows that the action rule is going to start. how is the action rule language processed?
all of this needs to be clearly specified. are you using a bnf grammar? it might help to do it in some kind of grammar to show how the language is being parsed.
also we need to discuss how a user starts using the system? logs in? where? the startup of this system is not described at all. all of a sudden we're jumping from describing SECE's overall architecture to specific rules for a user. how did we get here?
then indicate specifically where the user is setting things up. for example, it will have to set up a calendar/contacts and specify which calendar to use, etc. how does it do this? where is this set up done?
then give two or three sentence description on how the rule is executed. i know this is in the reference paper but you need to give an overview here bc it's really confusing. so a brief description i think is necessary.)

```
on Anne's birthday, 2010 at 12:00 in Europe/Zurich {
    sms Anne "Happy Birthday!!! John";
}

every week on WE at 6:00 PM from 1/1/10 until May 10, 2010
except 3rd WE of Feb including first day of June, 2010 {
    email irt-list "reminder: weekly meeting today at 6:00 PM";
}
```

Calendar-based rules: These rules specify events that are defined in the user's calendar. These can be triggered some time before or after these events occur, as well as when they begin or finish. An example is:

```
when 30 minutes before "weekly meeting" {
      email [event participants] "The weekly meeting will start in 30 minutes";
      if {me not 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-based rules: These define location events about the user and his or her friends. 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* and *moved*. Below we show a location-based rule using the *near* operator.
(???????? do you have some kind of formula for each operator? what does it mean for something to be "near"? 2 miles away? does the user define what these operators are and if so where do they do that?)

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

Context-based rules: These specify the action to execute when some context information changes, such as presence or sensor state.

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

Request-based rules: These specify the actions to execute in response to (1) incoming calls, IMs, emails, SMSs or voicemails, (2) outgoing calls or IMs, and (3) missed calls. All these events can be filtered by the user destination and origin. 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 Ontology-based User-defined Rules for Automatic Web Service Discovery

As Section **??** described, SECE enables end users to compose various services. However, the services that SECE currently interacts with are hard-coded and, therefore, extensible only if a developer reprograms it. As it stands, SECE has no way of automatically discovering new types of service, generating a rule language for it and incorporating it in its system.

We enhanced SECE to support automatic service discovery through service rules. This new kind of rule allows users to define web service events and the actions that are triggered when these events occur. With this enhancement, SECE users are capable to specify the kinds of service that they are looking for and the conditions that these services must satisfy in a natural-English-like language. When a web service that satisfies a web service rule's constrains is discovered (i.e., a web service event occurs), the rule's actions are taken. Web service rules constitute subscriptions to web service events and, therefore, provide a push approach for web service discovery. Below, a simple web service rule is shown. In this example, whenever a flight that satisfies the given constrains is found, an email and SMS are sent to the user.

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

Web service events are defined in the SECE sublanguage for web services that is described by Section 4.2. This ontology-based language can express any ontology's class as a set of constrains on its properties. Therefore, this is not bound up with any particular web service's ontology description. This decouples web service rules from service specifications and, hence, allows integrating new services transparently. Section 4.1 explains how we implemented web service rules into SECE. Although these rules are currently working into SECE, to make them useful, three crucial tasks are necessary. First, SECE has to be provided

somehow with the OWL-S specifications of the services about which end users can type rules. Second, the users have to be aware of these services' descriptions. Third, the users have to learn how to define web service rules. Regarding the first task, SECE has to interact with an ontology-based service discovery system that provides it with the ontology descriptions. At this point, we use GloServ as discovery system and manually configure the ontology descriptions that GloServ handles. However, in the future, it will be desirable to update the set of discoverable services dynamically. For instance, it could be achieved by implementing a subscription mechanism between SECE and GloServ. Regarding the second task, if SECE is used stand-alone, it should show the set of discoverable services and their descriptions on its graphical interface. If SECE is used by a front-end application (see Section 5), SECE should provide this application with the discoverable services' description and the application should let users know about these services somehow. Regarding the third task, to date, there is a manual for users to learn about the SECE language. As Section 5 describes, the use of front-end applications brings out several advantages that, among other things, pave the way for dynamic suggestion systems. This would very much ease the learning curve of SECE and its language.

4.1 Implementation of Web Service Rules

SECE stores the OWL-S specifications of web services in an ontology database that is built upon the Jena Framework [18]. When a web service rule is entered into SECE, it basically has to 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 3 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 constrains in the form of *(propertyName, operator, value)*, as it can be seen in the example rule above. 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 3). 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 constrains. Thus, SECE will ask the SECE Ontology Model for the namespace URI of the web service and its properties. If either the web service's or any of its properties' namespace can not be returned (i.e., its value is null), this means that some semantic checking failed

and the described web service does not correspond to any ontology. 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 3). The SECE core retrieves an event monitor from the Event Monitor Broker (*OntEM* and *EMBroker* in Figure 3). 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 not other subscribers, it creates the corresponding SPARQL [20] 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 3) 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 3 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 3 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 Sublanguage for Defining Web Service Events

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 constrains 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 constrains can also be expressed by the operators *that has (no)* and *that is (not)* as in

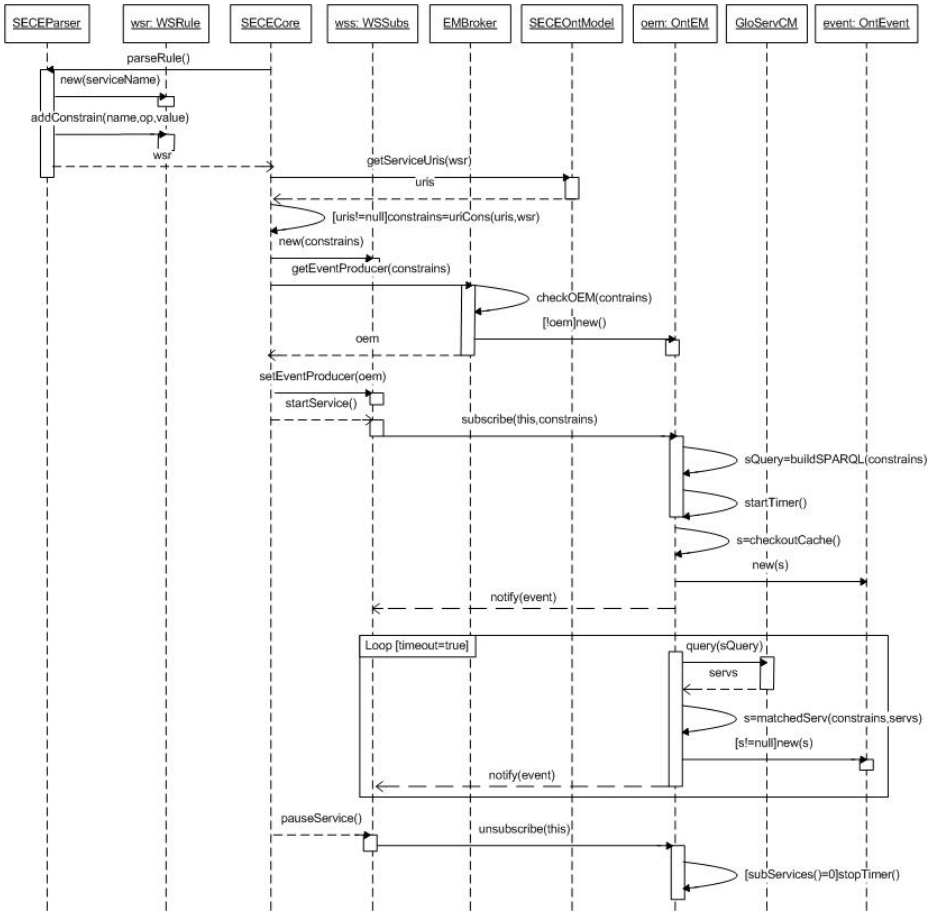

Fig. 3. Sequence diagram from entering a web service rule to querying GloServ

"any restaurant that has delivery", "any restaurant that is open 24 hours" and "any cultural exhibition that is free and is not crowded". Boolean constrains 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 constrains 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 Event-Based Context-aware Web Service Composition System

Integrating web service rules into SECE brings out exciting possibilities in the Semantic Web. This permits end-users to define and personalize context-aware web service discovery, invocation and composition based on a variety of events. 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. 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. Figure 4 outlines the main interactions between SECE, GloServ, front-end applications and web services. We assume that end users are connected to front-end applications, which detaches users from SECE and offers more flexibility. Front-end applications retrieves user data from SECE and allows users to create their rules probably by means of more fancy graphical

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

interfaces, suggestions and user preferences, for example. 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. In this frame, GloServ is the discovery agency and SECE is the service consumer in Figure **??**. A GloServ request specifies the web service of interest as a SPARQL query 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. Section 4.1 describes in more detail how we implement web service rules and events. Section 6 describes the SECE architecture and the necessary extensions to implement the described system.

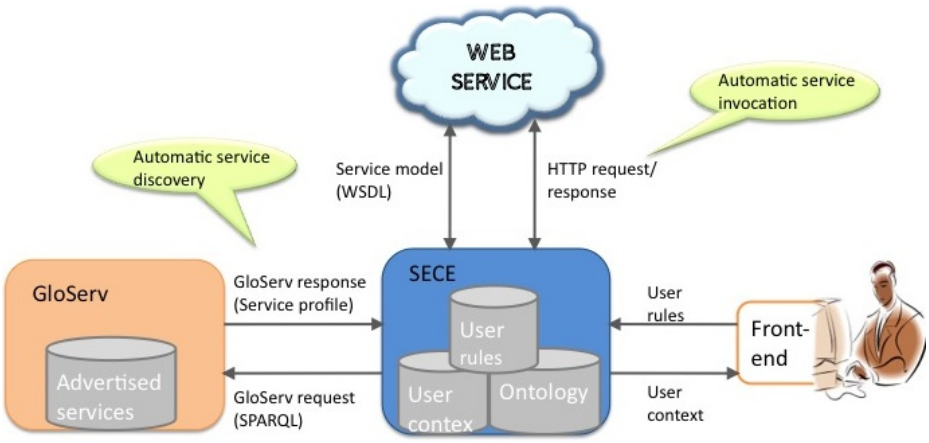

Fig. 4. SECE, GloServ, front-ends applications and web services

5.1 Architecture

SECE has a layered architecture as Figure 5 shows. SECE is being developed in Java due to its extensive libraries and support for all operating systems. The network layer contains the communication protocols. The Java runtime environment and the Java libraries layers constitute the platform on which all of the components rely. Figure 5 only shows some relevant Java libraries such as ANTLR, which is used by the language compiler, JACL [21] that is a Tcl implementation in Java, JAINSIP for SIP signaling and GDATA to access the Google's web services. The Service Agent Layer contains the agents that accomplish convenient functions by communicating with external web services. For instance, the Gmail agent send emails and the GMaps agent performs direct and reverse geo-coding. The Context Mediator Layer monitors changes in web services and accordingly generate SECE events. The SECE Core Layer provides all the logic for handling and executing SECE rules. This offers APIs to obtain user context, subscribe to events, execute actions and call service agents. The Jena Ontology Model is a database that contains all the ontology schemes, which relies on the Jena Framework [18]. The Actions Interpreter executes rule actions. The rule event broker receives event subscriptions from rules and redirects them to the CM orchestror. The CM orchestror is an intermediate point between the context mediators, which generate events, and the SECE Core Layer. Based on the kind of event, this requests a proper Context Mediator to monitor the event. When a subscribed event occurs, the CM orchestror is notified from the responsible Context Mediator and it, in turn, notifies the Rule Event Broker, which eventually lets the subscribed rules know about the new event. Rules can retrieve user context (such as personal data, presence and friends) and request the SA orchestror to perform some functions on their behalf (e.g., geo-coding). The SA orchestror coordinates the service agents to accomplish tasks. The Plugin Enabler component will be responsible for extending SECE dynamically in the future. The Rules Layer contains the rule instances that are subscribed to SECE events.

The new components that have been added to the SECE architecture in the presented proposal are: The *WBRL* rule, which implements the web service rule described previously, the Jena Ontology Model, which contains the necessary ontologies' schemes, the GloServ Service Agent, which is in charge of one-time service discovery and composition, and the GloServ Context Mediator, which periodically pulls GloServ for checking out new web services of interest. The GloServ Service Agent and Context Mediator translate SECE events as SPARQL queries which they send to GloServ.

6 Future Work

To this point, the architecture described in Section is partially implemented. We have already integrated the Jena ontology database into SECE, which stores the web service ontologies that GloServ provides. The GloServ Context Mediator has already been developed and is working in SECE. The generic SECE sublanguage for web service events has been implemented and tested. Apart from web services,

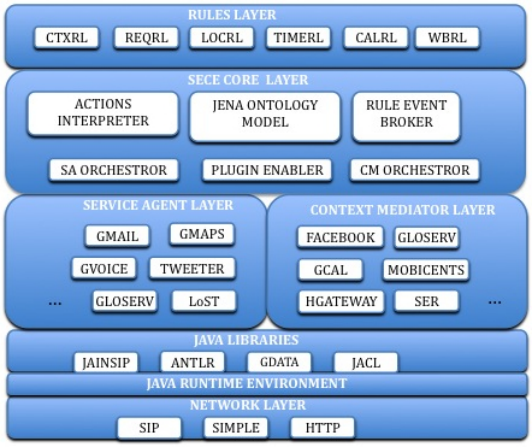

Fig. 5. SECE architecture

this language permits to define events about any entity defined by an ontology in SECE. There are still many tasks to tackle for implementing our proposal. Let us summarize the main next steps:

Subscription-based communication between SECE and GloServ: For the sake of scalability, GloServ should be extended to support web service subscription requests and so to avoid SECE sending periodic pull requests.

Implementation of service composition logic: SECE will be capable to compose web services automatically, given certain service capabilities or properties.

Implementation of find and plan commands: The syntax of these actions need to be defined and the actions themselves have to be integrated into the Actions Interpreter. The logic of one-time web service discovery is already implemented for the *find* command. However, the logic of web service composition needs to be implemented.

Implementation of the communication between SECE and front-ends: Front-ends will need to be able to retrieve user context from SECE, such as personal data, buddy lists and presence. As well, Front-ends will need to send SECE the rules that users enter into the system. SECE will, in turn, parse the rules and reply to the front-end with a success or fail response.

Mechanism for plugging new rules into SECE dynamically: 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. To allow front-ends to add new rules dynamically, SECE would have to provide proper interfaces to subscribe to events, obtain user context and interact with SECE core functions. Although this is a tricky issue that requires an extensive study, we envision the use of ontology schemes and semantic paths for providing a high-level interface to SECE data that is independent from any underlying document structure.

Mechanism for dynamic suggestions and learning of SECE rules: SECE could provide front-ends with rules' ontological models that let them reason about rule semantics. Thus, front-ends could learn new SECE rules automatically and offer end users suggestions and support in rule construction.

7 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, there is not any complete solution yet. Most of authors describe or propose theoretical works. The few works that present real implementations are partial solutions, domain-specific or lack of some desired features such as the following ones. There is a strong need of 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 (i.e., without human interaction). 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 come from GloServ. We developed a SECE sublanguage to subscribe to web services. This is independent from specific web services and therefore new kinds of service 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. This makes SECE suitable for non-technical users. SECE also allows creating service compositions that can be triggered by discovered web services and real-world events such as context changes, location, time, etc. SECE makes the current and future semantic web meet by mixing typical and semantic web services. This fact along with the integration of real-life events with web service discovery and composition is a key step to incorporate the Semantic Web into users' life. Moreover, the synergistic combination of SECE and GloServ paves the way for future extensions. 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.

References

1. J. Rosenberg, J. Lennox, and H. Schulzrinne, "Programming Internet telephony services," *Internet Computing, IEEE*, vol. 3, pp. 63–72, May/Jun 1999.
2. Xiaotao Wu and Henning Schulzrinne, "Programmable End System Services Using SIP," *Conference Record of the International Conference on Communications (ICC)*, May 2003.
3. L. Burgy, C. Consel, F. Latry, J. Lawall, N. Palix, and L. Reveillere, "Language Technology for Internet-Telephony Service Creation," in *Communications, 2006. ICC '06. IEEE International Conference on*, vol. 4, pp. 1795–1800, June 2006.
4. F. Latry, J. Mercadal, and C. Consel, "Staging telephony service creation: a language approach," in *IPTComm '07: Proceedings of the 1st international conference on principles, systems and applications of IP telecommunications*, (New York, NY, USA), pp. 99–110, ACM, 2007.
5. W. Jouve, N. Palix, C. Consel, and P. Kadionik, "A SIP-Based Programming Framework for Advanced Telephony Applications," in *IPTComm* (H. Schulzrinne, R. State, and S. Niccolini, eds.), vol. 5310 of *Lecture Notes in Computer Science*, pp. 1–20, Springer, 2008.
6. D. Cassou, B. Bertran, N. Loriant, and C. Consel, "A generative programming approach to developing pervasive computing systems," in *GPCE '09: Proceedings of the eighth international conference on Generative programming and component engineering*, (New York, NY, USA), pp. 137–146, ACM, 2009.
7. A. K. Dey and G. D. Abowd, "CybreMinder: A Context-Aware System for Supporting Reminders," in *HUC '00: Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, (London, UK), pp. 172–186, Springer-Verlag, 2000.
8. "task.fm Free SMS and Email Reminders." `http://task.fm`.
9. S. Ponnekanti and A. Fox, "Sword: A developer toolkit for web service composition," in *Proc. of the Eleventh International World Wide Web Conference, Honolulu, HI*, 2002.
10. J. Soriano, D. Lizcano, J. Hierro, M. Reyes, C. Schroth, and T. Janner, "Enhancing user-service interaction through a global user-centric approach to SOA," in *Networking and Services, 2008. ICNS 2008. Fourth International Conference on*, pp. 194–203, IEEE, 2008.
11. "Yahoo pipes." `http://pipes.yahoo.com/pipes/`.
12. R. Kazhamiakin, P. Bertoli, M. Paolucci, M. Pistore, and M. Wagner, "Having Services "YourWay!": Towards User-Centric Composition of Mobile Services," *Lecture Notes in Computer Science*, vol. 5468/2009, pp. 94–106, 2009.
13. "SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)." `http://datatracker.ietf.org/wg/simple/charter/`.
14. "About SIP Express Router." `http://www.iptel.org/ser/`.
15. O. Boyaci, V. Beltran, and H. Schulzrinne, "Bridging communications and the physical world: Sense everything, control everything," in *Proceedings on the IEEE Globecom (UbiCoNet Workshop)*, December 2010.
16. J. K. Ousterhout and K. Jones, *Tcl and the Tk Toolkit*. Upper Saddle River, NJ: Addison-Wesley, 2nd ed., 2009.
17. B. Desruisseaux, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)." RFC 5545 (Proposed Standard), Sept. 2009. Updated by RFC 5546.
18. "Jena - A Semantic Web Framework for Java." Website. `http://jena.sourceforge.net/index.html`.

19. T. Parr, *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 2007.
20. W3C, "SPARQL Query Language for RDF." Website, January 2008. `http://www.w3.org/TR/rdf-sparql-query/`.
21. I. K. Lam and B. Smith, "Jacl: a Tcl implementation in Java," in *TCLTK'97: Proceedings of the 5th conference on Annual Tcl/Tk Workshop 1997*, (Berkeley, CA, USA), pp. 4–4, USENIX Association, 1997.