Bridging communications and the physical world: Sense Everything, Control Everything

Omer Boyaci, Victoria Beltran and Henning Schulzrinne
Columbia University
1214 Amsterdam Avenue
New York, USA 10027
Email: {boyaci,hgs}@cs.columbia.edu, mvb2120@columbia.edu

***Abstract*—The *SECE* (Sense Everything, Control Everything) system allows users to create services that combine communication, calendaring, location and devices in the physical world. *SECE* is an event-driven system that uses a natural-English-like language to trigger action scripts. Presence updates, incoming calls, email, calendar and time events, sensor inputs and location updates can trigger rules. SECE retrieves all this information from multiple sources to personalize services and to adapt them to changes in the user's context and preferences. Actions can control the delivery of email, change the handling of phone calls, update social network status and set the state of actuators such as lights, thermostats and electrical appliances. We give an overview of the SECE language and system architecture.**

I. Introduction

Communication is not limited to telephony anymore, as millions use IM, SMS, email, Twitter, and Facebook everyday. These Internet services are not automated and programmable by end-users, decreasing their utility. Moreover, although these services handle very similar information (e.g., calendar, buddies status, presence, messages and user history), they do not interoperate with each other. Such a lack of service cooperation and automation forces users to check services one after another and manually copy data or configure services based on other services. Unfortunately, there is currently 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. To address these problems, we are developing SECE, a new language and supporting infrastructure which will enable users to create services for controlling their communication flow, and a range of different sensors, devices and services.

SECE is a context-aware platform that 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. SECE has to both sense and control because sensing without controlling is not very useful.

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. An example script which turns the home's lights on every sunset shows the end-user friendliness of SECE.

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

SECE has two fully-integrated components, the language itself and its supporting software architecture. IETF standard protocols are used to interconnect networked components.

The paper is organized as follows. Section II discusses related work. The SECE language is described in Section III, and the architecture of SECE is presented in Section IV. Section V presents conclusions and future work.

II. Related work

Several solutions for user created services have been proposed; some of these solutions are compared 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.

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

External knowledge	Example
access to the registry of personal information	My mobile, Bob's address
access to contextual information	me.location, bob.activity, bob.presence
access to in-context variables	inside an *incoming call* rule [call caller], reject
address book and IM/presence names	Bob's
calendar events, including public holidays	Thanksgiving, Bob's birthday
daily times	sunset, sunrise, dawn, dusk, twilight
usage of geocoding and gazettes to look up landmark names	"Columbia University"

TABLE I
SECE MAKES IT EASY TO INTEGRATE EXTERNAL KNOWLEDGE SEAMLESSLY

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

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.

To the best of our knowledge, there is no platform for composing services of different kind. Although service composition is being of great interest in the research community, most of the proposed solutions are only theoretical and do not provide any implementation. Yahoo Pipes [9] is a graphical tool for web service composition. However, it is not really easy-to-use and intuitive, which makes it very difficult for non-technical users. The scripting languages shown in Figure 1 are neither suitable for non-technical users and only support a limited set of context information.

There is not any solution that allows users to compose their own services and execute them when particular context events occur. The current solutions, as Yahoo Pipes, are not proactive because the end-user is who triggers the composite services. The Semantic Web [10] is making efforts to achieve automatic web service discovery, composition and execution based on ontologies. However, the implementation and acceptance of semantic composite services is being problematic due to practical issues such as user choice and long response times, among others.

III. THE SECE LANGUAGE

A SECE rule has two parts, the event description and the 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 natural English that has been designed to be easy-to-use and easy-to-remember by end-users. A very simple but illustrative example is below.

```
If Bob's status is working {
  sms me "Bob is already working";
}
```

The SECE language is only intended to define events, while rule actions are written in the Tcl language [11]. We chose Tcl due to its extensibility that allows adding new commands to its core in an easy and convenient way. 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's 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. We may add support for other scripting languages like Ruby [12] or Python [13] in the future. However, a promising although challenging future step would be to extend the SECE language to define rule actions.

The SECE language supports five types of events, which determine the kinds of rule that SECE handles: time, calendar, context, location and request. The following subsections

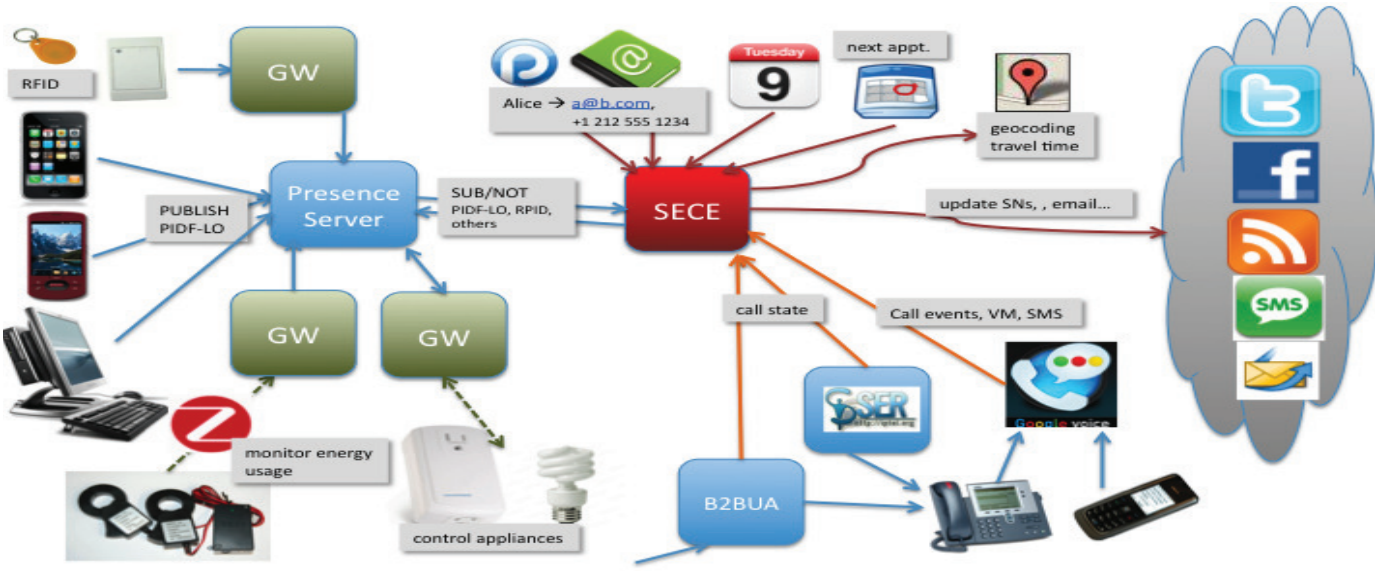

Fig. 2. The Architecture of SECE

explain each of these rules. As a formal language, SECE states the valid combinations of keywords and variables for each kind of event. In all the rule examples, the variables have been highlighted in bold to expose the structure of the language. 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. SECE tries to make it easy to integrate external knowledge seamlessly without having to explicitly invoke libraries or functions. The current status of this integration can be seen from Table I.

A. Time-based rules

Time-based rules support single and recurring events. The iCal specification (RFC5545 [14]) covers single and recurring events but it is designed to be processed by computers, not users. We designed the SECE's time sublanguage to be easy-to-write while maintaining the full expressive power of the iCal specification. Single events start with an *on* keyword, while recurring events start with an *every* keyword. An example of SECE time event and its equivalent iCal definition for a recurring event is given below.

```
SECE: every day at 12:00 until April

iCal: BEGIN:VCALENDAR
      BEGIN:VEVENT
      DTSTART;TZID=America/New_York:20100101T120000
      RRULE:FREQ=DAILY;BYHOUR=12;UNTIL=20100401T120000
      END:VEVENT
      END:VCALENDAR
```

The recurrence can be defined by the second, minute, hour, day, week, month or year. How long the recurrence takes is determined by the *from*, *until*, *during* or *for* parameters. A recurrence will repeat indefinitely if no *until*, *during*, or *for* parameters are indicated. The time sublanguage supports natural language constructs like *Thanksgiving*, *Tom's birthday*, *sunset*, *sunrise*, *lunch break*, and *tomorrow*. In the case of *Bob's birthday*, SECE will try to find the birthdate of Bob from available services like the users's calendar, Facebook or contacts. Similar lookup operations will be performed for *sunset*, *sunrise*, and *lunch break*. Some expressions like *sunset* and *sunrise* can be computed programatically whereas some of them like *lunch break* have to be defined by the user. Some example time-based rules are given below.

```
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@example.com.location {
    call bob;
}
every day at last working hour except August {
    backup;
}
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";
}
```

B. Calendar-based rules

Calendar-based rules specify events that are defined in the user's calendar and can be triggered some time before or after the events occur, as well as when the events begin or finish. These rules can 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 asks all of the user calendars about the event and, if it is found, determines when the rule should be triggered based on the rule's conditions and the event's starting and end times.

```
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.";
    }
}
when "weekly meeting" begins {
    status activity busy;
    sms [event participants] "Please, switch your cell phone off or set silent mode";
}
```

C. Location-based rules

The SECE's location sublanguage supports five types of location information that are commonly used: geospatial coordinates (longitude/latitude), civic information (street addresses), well-known places, user-specific places and other users. 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, but also allows the user to define custom terms, such as clubhouse in the list below. The supported location operators are *near [landmark]*, *within [distance] of [landmark]*, *in [landmark]* and *outside of [landmark]*. All these operators can be combined with the "a" and "an" indefinite articles to express generic locations (e.g., 'a postal office'). Some location events are given below.

```
Bob near "Columbia University" { ... }
me near a post office { ... }
me within 3 miles of "1000 Massachusetts Avenue, Washington, DC" { ... }
Alice in clubhouse { ... }
Tom within 5 miles of me { ... }
```

D. Request based rules

Request-based rules specify the action to execute in response to (1) incoming calls, IMs, emails, SMSs or voicemails, (2) outgoing calls or IMs, and (3) missed calls. While an incoming or outgoing call is always a SIP call, a missed call could be also a phone call. All these events can be filtered by the user destination and origin, using the *from* and *to* parameters respectively. Some request-based rules are given below.

```
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 not 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 not 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]"
    }
}
```

E. Context-based rules

Context-based rules specify the action to execute when context information changes, such as presence, call and sensor state.

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

The rule's context (e.g., activity, status and stock.google in the above rules) can be any hierarchical variable in the form of x.y.z.t, such as phone.office, activity and office.temperature. The context'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 equal to *bob's device.mobility*. The relational operators can be expressed as symbols or text (e.g., the equal relation can be given by "=", "is" or "equal"). Information derived from sensors, such as smoke, light, humidity, motion and temperature sensors 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]";
}
```

1) States vs. Events: SECE is designed for handling events, i.e., state transitions, that trigger a set of actions. This works well for discrete events, such as calls and calendar entries, but is somewhat more awkward for expressing 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;
```

IV. The architecture of SECE

Due to its integrative nature, SECE has to communicate with several third party applications, hardware, and APIs like Google services (e.g., GMail, GContacts and GCalendar), Facebook, Twitter, maps, VoIP proxy servers, presence servers, sensors and actuators (see Figure 2). SECE considers not only

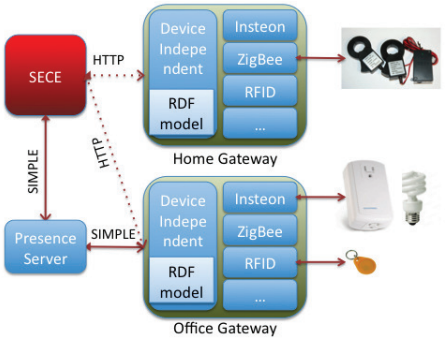

Fig. 3. The architecture of sensors and actuators gateway

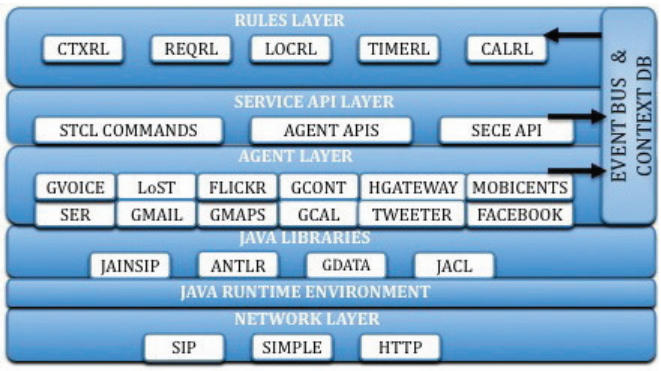

Fig. 4. The software components of SECE

the user's context but also information about external entities other than sensors, such as his or her buddies. SECE keeps the user information in a Document Object Model (DOM) [15] tree registry. The user information is not restricted to personal information like phone numbers but also includes contextual information from sensors and Internet services. Context-based rules associate events with the nodes of the registry. A 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.

As Figure 2 depicts, the Presence Server (PS) plays a key role in recollecting context from different sources. According to SIMPLE [16], the PS receives presence publications from the context sources that contain the most recent information and, in turn, it 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. Currently, we are using a Mobicents Presence Server [17].

SECE obtains sensor information through SIMPLE notifications that include RDF [18] documents, which makes it sensor network agnostic. Actions on actuators are described in RDF documents that are sent to the gateway via POST HTTP (see Figure 3). The gateway is split into two layers: a device-independent layer and a protocol layer. The former maintains an RDF database that represents the conceptual sensor model, while the latter carries out the necessary translations between the RDF model and the device- and 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.

Another external server that plays a key role in SECE is the SIP Express Router (SER) [19], which handles SIP communications. SER will inform SECE whenever an incoming or outgoing communication, such as a call or IM, takes place. Then, if a communication rule is triggered, a rule action could forward, reject, or modify the call.

A. *The software components of SECE*

The software components of SECE can be seen from Figure 4. We are developing SECE in Java due to its extensive libraries and support for all operating systems. Figure 4 only shows some relevant Java libraries such as ANTLR, which is used by the language compiler, JACL [20] 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 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 rules' conditions and to execute rules' 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.

V. Conclusion and Future Work

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 which integrate diverse information, such as location, presence, IM, SMS, calls, Facebook, Twitter, sensors and actuators. Facing it, we are developing a context-aware platform 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 composite their own services. Every rule specifies the event that triggers its service (i.e., the rule's body) and SECE monitors the event and proactively executes the service whenever the event occurs. The SECE language makes service composition GUI-independent as long as the GUI translates the user input into rules. In the future, it will make it possible to develop a more advanced GUI (e.g., suggestions and templates for service composition) without modifying the SECE's 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 4, all the rules are fully implemented except some location operators. Particularly, the "outside of" and "in" operators need the support of a tool for users to draw polygons and points of interest, which is still under development. Generic locations (e.g., a restaurant) are being implemented along with a LoST (Location-to-Service Translation Protocol) [21] server. The home gateway, Mobicents and Facebook agents are also still under development.

Although rule conflicts have already been considered for a particular user (i.e., resource rules in Section III-E), multi-user conflicts should be studied in the future, as well as run-time error handling. 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.

VI. Acknowledgments

We would like to thank Andrea G. Forte, Fan Yang, Gerald Scott Schuff, Jan Janak, Jaya Allamsetty and Ted Shin for helping us to build SECE. Victoria Beltran is supported by the scholarship grant FPU AP2006-02846 from the Government of Spain. Omer Boyaci is supported by a grant from CounterPath.

References

[1] J. Rosenberg, J. Lennox, and H. Schulzrinne, "Programming Internet telephony services," *Internet Computing, IEEE*, vol. 3, no. 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, June 2006, pp. 1795–1800.

[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: ACM, 2007, pp. 99–110.

[5] W. Jouve, N. Palix, C. Consel, and P. Kadionik, "A SIP-Based Programming Framework for Advanced Telephony Applications," in *IPTComm*, ser. Lecture Notes in Computer Science, H. Schulzrinne, R. State, and S. Niccolini, Eds., vol. 5310. Springer, 2008, pp. 1–20.

[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: ACM, 2009, pp. 137–146.

[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: Springer-Verlag, 2000, pp. 172–186.

[8] "task.fm Free SMS and Email Reminders," http://task.fm.

[9] "Yahoo pipes," http://pipes.yahoo.com/pipes/.

[10] "The semantic web," http://www.w3.org/2001/sw/.

[11] J. K. Ousterhout and K. Jones, *Tcl and the Tk Toolkit*, 2nd ed. Upper Saddle River, NJ: Addison-Wesley, 2009.

[12] "Ruby Programming Language," http:/www.ruby-lang.org/.

[13] "Python Programming Language," http:/www.python.org/.

[14] B. Desruisseaux, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)," RFC 5545 (Proposed Standard), Internet Engineering Task Force, Sep. 2009, updated by RFC 5546. [Online]. Available: http://www.ietf.org/rfc/rfc5545.txt

[15] "Document Object Model (DOM) Level 3 Core Specification," http://www.w3.org/TR/DOM-Level-3-Core/.

[16] "SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)," http://datatracker.ietf.org/wg/simple/charter/.

[17] "Mobicents," http://www.mobicents.org/.

[18] "Resource Description Framework," http://www.w3.org/RDF/.

[19] "About SIP Express Router," http://www.iptel.org/ser/.

[20] 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: USENIX Association, 1997, pp. 4–4.

[21] T. Hardie, A. Newton, H. Schulzrinne, and H. Tschofening, "LoST: A Location-to-Service Translation Protocol," RFC5222 (Proposed Standard), Internet Engineering Task Force, August 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5222.txt