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

Omer Boyaci, *Member, IEEE,* Victoria Beltran, *Member, IEEE,* and Henning Schulzrinne, *Fellow, IEEE*

Abstract—*SECE* (Sense Everything, Control Everything) allows non-technical end-users to create services that combine communication, social networks, presence, 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. Users associate actions with events; when an event occurs, associated actions are executed. Presence updates, incoming calls, email, calendar and time events, sensor inputs and location updates can trigger rules. SECE combines 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 the user's social network status and set the state of actuators such as lights, thermostats and electrical appliances.

Communication is not limited to telephony anymore, as millions use IM, SMS, email, Twitter, and Facebook everyday. Although these stand alone Internet services improve our daily life, they are not automated and programmable by end-users, decreasing their utility. For example, it is not easy to create a service which forwards incoming calls to voice mail while the user is in a meeting or turns on the air conditioner while the user approaches his home. Moreover, although these services handle very similar information (e.g., calendar, presence, messages and user history), they do not work together. 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.

SECE is a context-aware, intelligent, integrative, human-oriented and proactive platform that connects services that until now were isolated, leading to new, more useful and user-personalized, composite services will make our communication more efficient and our life easier. These services do not require user interaction; they are automated and embedded into users' life. SECE does not require user interaction except during the service creation phase due to its event-driven operation. Incoming and outgoing phone calls, IM or email messages, presence status updates, sensor inputs, location updates, social network activities such as incoming wall messages or tweets, changes in stock prices or weather are all possible SECE events. Whenever an event occurs, it triggers one or more user-created SECE services. 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 users depending on the monitored information and triggered events. In order to build such a system, the user has to define event-action rules. In general, there are several ways to define these rules such as using XML, forms or scripts. We chose 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. SECE addresses users who are comfortable with technology, but are not programmers. The main challenges were to keep the language user-friendly while not decreasing its power and to develop the software which has to integrate and communicate with several Internet services such as email, IM, phone, SMS, location, calendar, presence and translation services, and social networks.

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

Listing 1: SECE script which turns the home's lights on every sunset

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

IETF standard protocols are used to interconnect networked components. We first discuss the related work. Then we present the SECE language and integration of SECE with external services and resources.

RELATED WORK

Several solutions for user created communication-related services have been proposed; some of these solutions are compared in Table 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 a very low level domain-specific design language similar to Java. Writing a specification in DiaSpec and then developing a service using the generated Java 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 such as forwarding or rejecting incoming calls. VisuCom has the same functionality as SPL, but allows users to create services visually via GUI components. Although visual interface of VisuCom is suitable for end-users, its services are limited to telephony events.

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 such as active badges and floor-embedded pressure 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.

Yahoo Pipes [9] is a graphical tool for web service composition, but it only generates web mashups from public web feeds and public webpages, which means it could not even generate web mashups from users' private information such as their Facebook wall messages or emails.

There are also some web aggregation services like Timelimes [10] and netvibes [11], but they just combine all the news and events from users' social networking, webmail and news sites into one simple page. Ping.fm [12] can update several social network statuses from a single webpage, but it does not support actions triggered by events.

Google previewed an initiative called Android @ Home during Google I/O conference 2011, which allows Android apps to discover, connect and communicate with appliances and devices in your home. SECE may control and sense these devices using the same APIs available to Android applications.

AppleScript, which also tries to mimic natural English diction, allows end-users to write scripts for automating repetitive tasks and integrating a range of desktop applications.

In summary, the tools shown in Table 1 are either not suitable for non-technical users or only support a limited set of context information.

Systems	User rules	User actions	Communications	Time	Location	Presence	Sensors	Web services	Actuators
SECE	NL-like rules	Tcl scripts	Call, email, IM	✔	User & buddies	✔	✔	✔	✔
CPL	XML tree	Fixed XML actions	Call	✖	✖	✖	✖	✖	✖
LESS	XML tree	XML actions	Call	✔	✖	✔	✖	✖	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	✔✖	✖✔	✖✔	✖✔	✖✔	✖✔	✖✔

TABLE 1: Comparison to related work

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. We have designed the SECE language as formal language but similar to natural English, making it easy to remember and use. A very simple but illustrative example which sends an SMS to the user when Bob's presence status changes to available is given below:

Listing 2: An example script which SMS the user when Bob's presence status changes to available

```
If Bob's status is available {
  sms me "Bob is available now.";
}
```

The SECE language is only intended to define events, while rule actions are written in the Tcl language [13]. We chose Tcl due to its extensibility that makes it simple to add new commands. We introduced a set of new Tcl commands, such as "sms", "im", "email", "tweet" or "call" (complete list is shown in Table 2). Combined with the SECE's web-based editor which features auto-completion and shortcut buttons for the action commands; the end-users do not need any Tcl knowledge in most situations. Thus, SECE users can describe events in a user-friendly and natural way while taking advantage of the expressive power of Tcl to define advanced actions if they need to. (Support for other scripting languages like Ruby or Python may be added in the future.)

The SECE language currently supports five types of events: time, calendar, context, location and request. The following subsections explain each of these rules. As a formal language, SECE states the valid combinations of keywords and variables for each kind of event. Some newly introduced Tcl commands are specific for particular events as for example the "accept" and "reject" commands can only be used in communication-based rules. SECE tries to make it easy to integrate external knowledge and uses context such as addresses, phone numbers, weather, and stock prices seamlessly without having to explicitly invoke libraries or functions.

Time-based rules

Time-based rules support single and recurring events. We base our time sublanguage design on the iCal specification. iCal can express single and recurring events but it is designed to be processed by computers rather than users. We designed the SECE's time sublanguage to be easy to write while maintaining the full expressive power of the iCal specification. Single events start with an on keyword, while recurring events start with an every keyword.

An example of SECE time event which will trigger every noon till next April and its equivalent iCal definition for a recurring event is given below.

Listing 3: An example recurring time event which will trigger every day 3 PM till next April

```
SECE: every day at 15:00 until April

iCal: BEGIN:VCALENDAR
      BEGIN:VEVENT
      DTSTART;TZID=America/New_York
          :20100101T150000
      RRULE:FREQ=DAILY;BYHOUR=15;UNTIL
          =20100401T150000
      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, Bob's birthday, sunset, sunrise, lunch break, and tomorrow. In the case of Bob's birthday, future versions of SECE may try to find the birthdate of Bob from available services like the user's calendar, Facebook or contacts. Similar lookup operations can be performed for sunset, sunrise, and lunch break. Some expressions like sunset and sunrise can be computed programmatically whereas others like lunch break have to be defined by the user via SECE's web-based user interface. Some example time-based rules are given below.

Listing 4: Sends an SMS to Anne on her birthday

```
on Anne's birthday, 2010 at 15:00 in Europe
    /Zurich {
  sms Anne "Happy Birthday!!!kisses. John";
}
```

Listing 5: Sends a reminder email to a list about weekly meeting

```
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 "Meeting Reminder" "weekly
       meeting today at 6:00 PM";
}
```

Calendar-based rules

Calendar-based rules specify events that are defined in the user's calendar. They can be triggered some time before or after an event occurs, as well as when an event begins or finishes. SECE can download all the events from user's Google account using the Google Calendar API. We implemented the event command to get a Calendar event's information (title, description, location, duration, start time, end time and participants). Calendar-based rules can be useful to create user-personalized reminders, as in the example below.

Listing 6: Reminds the participants half an hour before the weekly meeting begins and if the user is not within the three miles of campus emails Bob to prepare everything

```
when 30 minutes before "weekly meeting" {
  email [event participants] "Reminder" "
      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

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 the location of 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 university. The system resolves these constants via the user's address book, but also allows the user to define custom terms, such as "office" 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 examples of location events are given below.

Listing 7: Examples of location-based rules

```
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 { ... }
```

Communication-based rules

Communication-based rules specify the action to execute in response to incoming calls, IMs, emails, SMSs or voicemails, outgoing calls or IMs, and missed calls. While an incoming or outgoing call is always a SIP call in our implementation, 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.

The Tcl environment of SECE is context aware. Properties of incoming events can be accessed via the incoming command. This command takes a parameter and returns the requested information about the incoming event. The supported parameters are origin, destination, content, timestamp, and subject. Depending on the incoming event type this command may return different results. For example, incoming content may return the message text for an IM event or the email body for an email event. There are other commands like accept, reject, and forward; these will only be available if the context is right.

Listing 8: Call forwarding in a busy situation

```
incoming call from a workmate {
  if {[my activity is "on the phone"] } {
    forward sip:bob@example.com;
    email me "[incoming origin] tried to
        reach you at [incoming timestamp]";
  }
}
```

Context-based rules

Context-based rules specify the action to execute when context information changes, such as presence, call, weather, stock prices, or sensor states. To be more extensible, SECE keeps all the contextual information in a Document Object Model (DOM) tree registry. The contextual information is not restricted to personal information like phone numbers but also includes information from sensors and Internet services. The contextual information (e.g., activity, status and stock.google in the below example rules) can be any hierarchical variable in the form of x.y.z.t, such as phone.office, activity and office.temperature. 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.

Each user has a separate and isolated tree in the current implementation, but future versions of SECE may enable users to share parts of their tree with other users. Any existing or newly introduced component of SECE can read and write to the registry tree. Registry trees are automatically stored in permanent storage and their state are restored in case the SECE server reboots. The following listings present example context-based rules.

Listing 9: Notifies the user via instant messaging as soon as Bob becomes available

```
if bob@example.com's status is available {
  im me "Bob is available.";
}
```

Listing 10: Notifies the user via SMS if Google's stock prices passes $580

```
if stock.google > 580 {
  sms me "google stock: [stock google]";
}
```

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 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 of 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).

Listing 11: Notifies the user via SMS and fire department via text-to-speech if office smoke detector detects a fire

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

Integration with External Services

Due to its integrative nature, SECE communicates with several third party applications, hardware, and APIs like Google services (e.g., GMail, Google Contacts and Google Calendar), Facebook, Twitter, online maps, VoIP proxy servers, presence servers, sensors and actuators, and location, photo sharing and calendaring services (Figure 1). SECE considers not only the user's context but also information about external entities other than sensors, such as his or her buddies.

In order to integrate with online services, SECE has to take actions on behalf of the user; such actions usually require authentication. Asking the user for his username and password is a possible but problematic solution. Users do not want to give their username and passwords to other services to prevent account theft and unauthorized activities. Fortunately, the IETF OAuth standard solves these problems by authenticating SECE to these online services without requiring the user's username and password.

External services can update a user's DOM tree registry which may trigger associated context-based rules.

Sensors and Actuators

Sensors and actuators are an important part of our daily life and we thus made them part of SECE. Electrical appliances can be controlled automatically depending on the information coming from sensors, from other web services such as weather services and from time of day. Presence sensors may update a user's availability. Combined with SECE's knowledge on the user's presence, location, and availability, actuators may take actions on behalf of the user. For example, when the user is approaching his office, SECE may turn on the air conditioner depending on the temperature information coming from office temperature sensor. Or SECE may turn off the lights if there is no motion sensor activity in the user's home or office.

SECE's sensors and actuators support is platform independent. Currently, we are experimenting with ZigBee and Insteon wireless device control modules and Phidgets USB sensors and actuators.

Location

SECE learns the user's current location from Google Latitude service. We choose Google Latitude since it has clients for almost all mobile platforms. The user's mobile device uploads his location to Google Latitude servers periodically. SECE retrieves the user's current location from Latitude servers.

In order to support location-based rules, learning the user's current location is not enough. SECE computes the distance between the user's location and an address and supports not only point-based locations but also polygon based ones. SECE uses online map

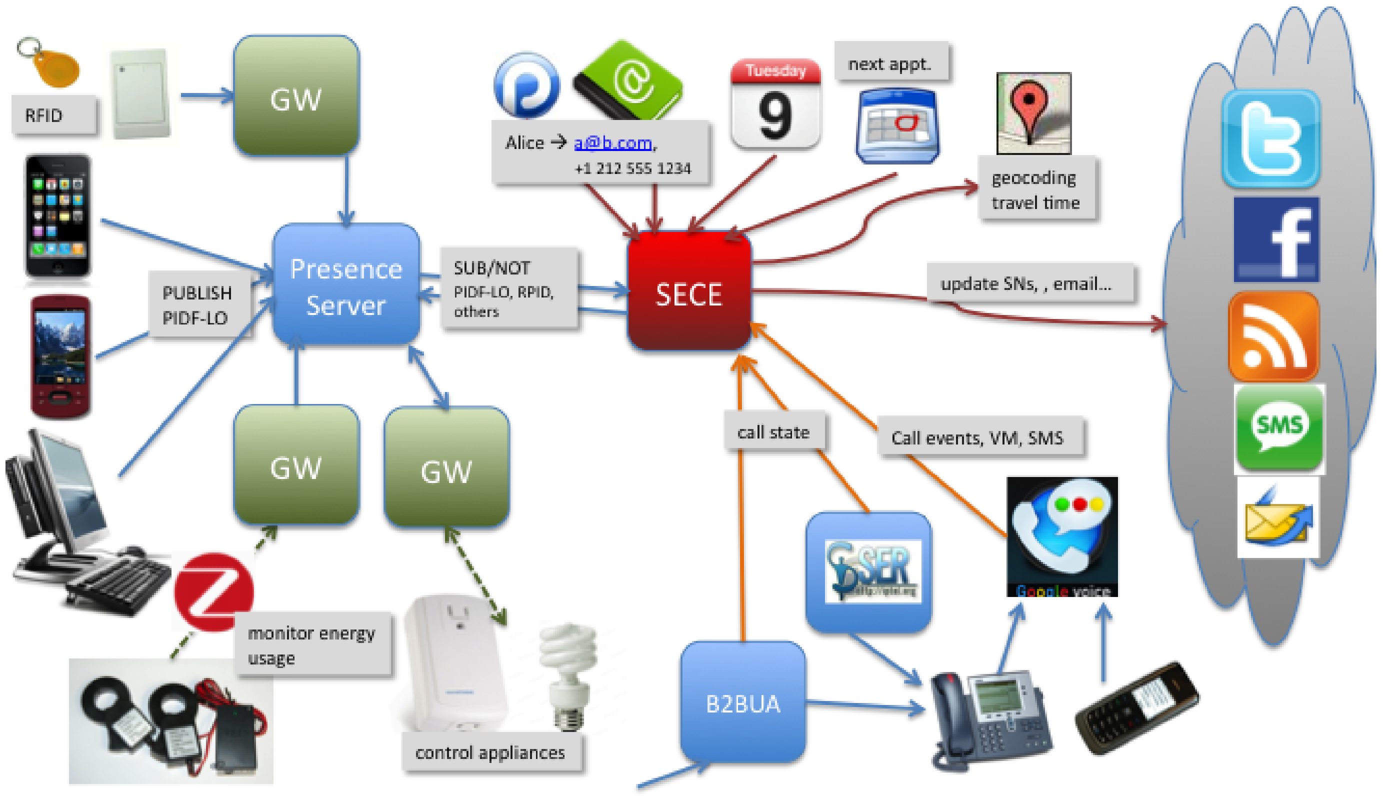

Fig. 1: The Architecture of SECE

APIs for geo-coding and supports polygon definitions both in the location sublanguage and also in the GUI.

Social Networks

SECE can receive direct and wall messages from Twitter and Facebook. It can also post tweets to Twitter and change Facebook status.

Presence and Instant Messaging (IM)

Integration of Presence and Instant Messaging (IM) networks to SECE allows users to monitor their friends presence states, to send instant messages to their friends programmatically, to update their presence states using other information sources such as their current location, activity (e.g., on the phone call) or calendar (e.g., in a meeting), to process incoming IM programmatically. SECE not only can monitor incoming instant messages but can also relay them via other communication methods like SMS in case users are away from their computer. IETF has standardized two presence and IM protocols namely SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) and The Extensible Messaging and Presence Protocol (XMPP). We integrated both of them into SECE.

VoIP Systems

SECE is integrated with the SER (SIP Express Router, http://www.iptel.org/ser/). There is a TCP-based communication link between SER and SECE. Using this link, SER and SECE can exchange messages to initiate new calls or to handle an incoming or outgoing call. New Tcl commands reject, forward, and accept are added to handle an incoming or outgoing call. SECE can initiate SIP calls using the new Tcl action command call. Another action command calltts is introduced to allow reading the given command parameters to the dialed number with the help of text-to-speech technology. The current SER implementation does not support call and calltts commands yet.

Context	Event	Action
Facebook	incoming wallmessage incoming newsmessage incoming direct	facebook
Twitter	incoming twitter direct incoming twitter wallmessage	tweet
Phone calls	incoming call incoming voicemail missed call outgoing call	call calltts accept reject forward
SMS	incoming SMS	sms
IM	incoming im outgoing im	im
Email	incoming email	email
Presence	if Bob is available	presence
Calendar	when [time] before [meeting] when [meeting] begins	schedule
Flickr		flickr
Translate		to_en, to_tr, . . .
Location	near [landmark] within [dist] of [landmark] in [landmark] outside of [landmark]	
Time	on [time] every [time]	
Contextual	if [variable] [operator]	status [variable] [value]

Sensors	if office.motion equals true
	if office.temperature > 250	

Actuators		status office.light true

TABLE 2: Summary of SECE events and actions

Extensibility and Graphical User Interface

While designing SECE we specifically considered the fact that SECE should be able to support new event types and action commands. Adding a new action command or a new event type is pretty easy, however recompilation and server restart are still required. SECE has a web interface to manage rules, registry, logs, location polygons and third party service subscriptions. Beltran et al. [14] propose integrating SECE with more advanced, easy-to-use front-end applications as well as discovering and composing web services automatically in SECE.

Deployment

The SECE server could be deployed in a home device or provided as a cloud service. The latter may make controlling in-home devices more challenging, given NATs. Also, in-home servers protect privacy and security by keeping the rules and details of sensors and actuators within home boundaries. However, such an arrangement makes it more difficult to update the rules from anywhere.

Implementation Details and Evaluation

SECE provides a simple way to express end-users requirements. Table 2 summarizes action commands and events supported by SECE. We compared the complexity of weekly tweeting a message using both SECE and Java. The SECE rules consists of two natural-English-like lines, whereas Java version required 125 lines of code [15]. Also, in order to write the same rule in Java, the user has to learn how to use iCal4j and Twitter4J third-party Java libraries. This comparison excludes the authentication part which is just a mouse click in SECE, while the Java one has to use an OAuth library.

In the future, more sophisticated GUIs may be built on top of SECE for providing users with suggestion systems that help in constructing meaningful rules. On the other hand, the SECE language is more resource-efficient and flexible for people with some skills on technology. An end-user evaluation study may help us to better understand the end-user friendliness of the SECE language.

SECE consists of almost 20,000 lines of Java source code and 2,000 lines of grammar definition to describe the sublanguages. Fifty different third-party Java libraries are used. Some events such as time, communication and sensors are handled in a push-based fashion, while others like facebook and email are handled in a pull-based fashion. SECE does not support boolean constructs for events, but allows responding to an event depending on other contextual information. For example, in Listing 11 for an incoming call event, the caller is forwarded to Bob if the user is on the phone.

Actions are triggered independently, so there is no checking that actions do not contradict each other. We also make no guarantees as to the order of execution if a single condition triggers multiple rules.

WE believe that intelligent, integrative, context-aware, human-oriented and proactive systems like SECE will make our communication more efficient and our life easier.

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," *IEEE Internet Computing*, vol. 3, no. 3, pp. 63–72, May/Jun 1999.

[2] X. Wu and H. Schulzrinne, "Programmable End System Services Using SIP," in *IEEE International Conference on Communications*, vol. 2, Anchorage, Alaska USA, May 2003, pp. 789–793 vol.2.

[3] L. Burgy, C. Consel, F. Latry, J. Lawall, N. Palix, and L. Réveillère, "Language Technology for Internet-Telephony Service Creation," in *IEEE International Conference on Communications*, vol. 4. Istanbul Turkey: IEEE Computer Society Press, 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 United States: 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 *2nd LNCS Conference on Principles, Systems and Applications of IP Telecommunications*, Heidelberg Germany, 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 8th International Conference on Generative Programming and Component Engineering*. Denver, CO États-Unis: 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*. Bristol, UK: Springer-Verlag, 2000, pp. 172–186.

[8] "task.fm Free SMS and Email Reminders," http://task.fm, 2011, [Online; accessed 18-April-2011].

[9] "Yahoo pipes," http://pipes.yahoo.com/pipes/, 2011, [Online; accessed 18-April-2011].

[10] "Timelimes web aggregator," http://www.timelimes.com/, Sep. 2010, [Online; accessed 18-April-2011].

[11] "Netvibes - Dashboard Everything," http://www.netvibes.com/, Sep. 2010, [Online; accessed 18-April-2011].

[12] "Ping.fm," http://www.ping.fm/, Sep. 2010, [Online; accessed 18-April-2011].

[13] J. K. Ousterhout and K. Jones, *Tcl and the Tk Toolkit; 2nd ed.*, ser. Addison-Wesley Professional Computing Series. Addison-Wesley, 2009.

[14] K. A. Victoria Beltran and Henning Schulzrinne, "Ontology-based User-defined Rules and Context-aware Service Composition System," in *Proceeding of the 4th International Workshop on Resource Discovery (RED 2011)*, Heraklion, Greece, May 2011.

[15] "Comparison of Java and SECE by example," http://www.cs.columbia.edu/~boyaci/sece/comparison/, 2011, [Online; accessed 22-November-2011].

PLACE PHOTO HERE

Omer Boyaci Omer Boyaci is a software engineer at Google. He holds a B.S. in computer science from Bilkent University (Turkey) and a PhD in computer science from Columbia University. His research interests are real-time networking, software development and multimedia networking. Contact him at boyaci@cs.columbia.edu.

PLACE PHOTO HERE

Victoria Beltrán Martínez received a degree in computer engineering from the University of Murcia, Spain, 2005. She is currently a PhD candidate in the Department of Telematics Engineering of the Technical University of Catalonia (UPC), Spain. Her research interests include context management, context-aware applications and presence-based systems. Contact her at vbeltran@entel.upc.edu.

PLACE PHOTO HERE

Henning Schulzrinne Prof. Henning Schulzrinne, Levi Professor of Computer Science at Columbia University, received his Ph.D. from the University of Massachusetts in Amherst, Massachusetts. He was an MTS at ATT Bell Laboratories and an associate department head at GMD-Fokus (Berlin), before joining the Computer Science and EE departments at Columbia University. He served as chair of Computer Science from 2004 to 2009.

Protocols co-developed by him, such as RTP, RTSP and SIP, are now Internet standards, used by almost all Internet telephony and multimedia applications. His research interests include Internet multimedia systems, ubiquitous computing, mobile systems. He is a Fellow of the IEEE. Contact him at hgs@cs.columbia.edu.