SECE: Making Services Programmable (Again)
Overview

- Motivation and (brief) history
- Programming in-network services $\rightarrow$ NetServ
- Combining web and telecom services $\rightarrow$ SECE
A Pyramid of programmers

Clocks at home blink 12:00
Can program a DVR
Writes Excel formulas
Creates simple HTML
Writes script in PHP or Ruby
Java programmer
C & kernel code
Firmware

JAIN
Levels of programming

Combination of services

Services (CPL)

Application protocols (Java JSR xxx)

Transport protocols, with names (Java)

Transport protocols (sockets)
SECE: Sense Everything, Control Everything
Omer Boyaci, Victoria Beltran and Henning Schulzrinne
SECE allows non-technical users to create services that combine communication, calendaring, location, and devices in the physical world. SECE: event-driven system uses high-level event languages to trigger action scripts, written in Tcl and other languages in the future.
Events & actions

**Events**
- Presence updates
- Incoming calls
- Email
- Calendar entries
- Sensor inputs
- Location updates

**Actions**
- Control the delivery of email
- Route phone calls
- Update social network status
- Control actuators such as lights
- Reminders (email, voice call, SMS)
- Interact with Internet services
Event language syntax

every sunset {
    homelights on;
}

every week on WE at 6:00 PM{
    email irt_list “Pizza talk at 6:00 PM today.”;
}

if my stock.google > 14 {
    sms me "google stock:"+[stock google];
}
Event rules: more examples

- **Time**
  - Single: on February 16, 2010 at 6:00 PM
  - Recurring: every day at 12:00 until April

- **Location**
  - Tom within 5 miles of me

- **Context**
  - if my office.temperature > 80

- **Communication requests**
  - incoming call
SECE: The glue for Internet applications

GW: PUBLISH PIDF-LO

SUB/NOT PIDF-LO, RPID, others

update SNs, email...

B2BUA: call state

Call events, VM, SMS

next appt.

geocoding travel time

monitor energy usage

control appliances

Alice → a@b.com, +1 212 555 1234

RFID GW

Monitor energy usage

GW: control appliances

Tuesday 9
Software Modules

- Email
- SMS
- IM
- Calls
- Geocoding
- User State
- Geo cache
- Scripts
- ANTLR
- JACL
- Calendar R
- Time R
- Location R
- Context R
- Request R
- Sensor
- Sensors & Actuators Gateway
- LoST
- LoST Server
- Mobicents
- Mobicents
- SER
- SER
- C
- C
Software architecture
Every day at 12:00 from 01/01/2010 until 04/01/2010 { email employees “lunch time” “Location: 5th floor Dinning Room, Time: 12:30” }
on Anne's birthday at 12:00 am in anne.location {sms anne "Happy Birthday!"}

e.g Europe/Zurich

sunrise
sunset
evening twilight
morning twilight
first working hour
last working hour
lunch break

2011-12-31, 12/31/2011
December 31, 2011
31th day of December, 2011
300th day, 2011
2th MO of May, 2011
May 2th Monday, 2001
Last Sunday in 52th week, 2011
Christmas Day, 2011
Thanksgiving Day, 2011
1 day before Thanksgiving Day, 2011
Time: recurrences

```
every freq (on dateExpr) (at timeExpr) (in timezone) (from dateExpr) (until dateExpr) (for num times| timeUnits) (during period) (except dateExprList) (including dateExprList) { body }
```

every sunset { homelights on; } → “every day at sunset”

every year on last Friday of January, March, June at last working hour except August { backup; }

every day at 10 minutes before lunch break from September 1, 2010 until Dec 24, 2010 { sms group-list "Lunch time! Let’s meet in 5 minutes"; }

every last monthly day { email me "Reminder: Check the students’ monthly reports"; }

every WE at 6:00 PM from 10/1/09 until May 12, 2010 except 3th WE of Feb, including first day of June, 2010 { email irt-list "reminder: weekly meeting today at 6:00 PM"; }
```
Calendar events

when meeting-name begins|finishes { body }

when time time-units before|after meeting-name { body }

when "weekly meeting" begins {
    status activity busy;
    sms [event participants] "Please, switch your cell phone off or set silent mode";
}
when 30 minutes before "weekly meeting" {
    email [event participants] ""[event title]" "The event [event title] will start is 30 minutes and will last [event duration] minutes. Description: [event description]. Start time: [event start]."
    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 events

user operator location { body }

bob near "Columbia University"
me near 40.807,-73.963

tom within 5 miles of me
me within 3 miles of ”2960 Broadway, New York, 10027”

tom in “Rockefeller center”
Me outside of “Manhattan”

bob moved 1.5 miles

- Place types and user-defined locations:
  me near a post office
  Anne in a museum
  me near my tennis club
Location-based events

(user, op, location)

Location rule

SECE Server

Presence Server

geo-cache

PUBLISH

SECE User

LoST Server

Geographical database

geo-cache

NOTIFY

(lost reply)

location=civic info

location=place type

Presence database

location=user

location=civic info

Location rule

Location rule
Handing location updates

- **User**
  - publishes his/her location periodically (e.g., every 5 min) to a presence server or to a location service such as Google Latitude

- **Presence server**
  - notifies changes in location to SECE server

- **Google Latitude (or similar service)**
  - SECE retrieves user’s location periodically

- **SECE server**
  - depending on user’s defined rules, queries LoST server
  - takes action based on rules and contextual location information

- **LoST server**
  - replies with current information on user’s surroundings

IBM Service Science Workshop May 2010
<table>
<thead>
<tr>
<th>Incoming</th>
<th>social_network</th>
<th>message_type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>facebook</td>
<td>wallmessage</td>
</tr>
<tr>
<td></td>
<td>twitter</td>
<td>newsmessage</td>
</tr>
<tr>
<td></td>
<td>linkedin</td>
<td>direct</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>social_network</th>
<th>status_update</th>
</tr>
</thead>
<tbody>
<tr>
<td>facebook</td>
<td></td>
</tr>
<tr>
<td>twitter</td>
<td></td>
</tr>
<tr>
<td>linkedin</td>
<td></td>
</tr>
</tbody>
</table>
What’s context?

- Presence information ➔ IETF SIMPLE specifications
- Sensor information

```plaintext
if bob's activity is idle { call bob }
If bob@domain.com’s status equals working { alarm me }
if my stock.google > 14 { sms bob “water the plants”; }
If me.office.temperature < 50 { ac off; }
If anne.location.civic changed { email me “Anne’s civic location:”+[status anne location.civic]}
```
Context as a tree

- bob
  - placeis
    - video
    - text
  - status
  - activity
  - office
    - temperature
  - phone
    - home
    - work

bob.placeis.audio
bob.activity
bob.phone.work
Automated Call Handling

- **Control**: Accept, reject, redirect, forward calls based on variety of SECE signals
- **Integration**: Calendar, address book, PSTN, Google Voice, SMS, location, Text-to-speech, voicemail
- **Simplicity**: Natural, easy to learn scripting language
- **Flexibility**: Input from a variety of SECE components involved in call handling
- **Automation**: Scripts for recurring tasks (setup a conf. call based on calendar)

“On mom's birthday, call mom when I am home and near phone.”
“Setup a conference call, enter password, invite people, ring desk phone.”
“If driving and incoming call, play “user driving” and redirect to voicemail.”
“If desk phone ringing and not in room, send SMS with caller's number.”
Communication-based events

 incoming|outgoing event from user|address to address { body }

 missed call from user|address to address { body }

 received call from user|address to address { body }

Event: call, im, sms*, voicemail*, email (*only incoming)

incoming call {
  if { [my activity] == "on-the-phone"} forward sip:bob@example.com
}
outgoing call {
  if {[outgoing destination] == "18003456789"} modify_call destination 12129397054
}
ingoing call from Anne {
  if {[my location] != "office"} auto_answer audio no_office.au –record
}
ingoing im {
  sms me [incoming from]+" sent an im:"+[incoming content]
}
ingoing call {
  schedule "call received from [incoming origin]"
}
Sensors and actuators

**Sensors**: smoke, light, humidity, motion, temperature and RFID Readers

**Actuators**: networked devices and actuators such as lights, cameras, sprinklers, heaters, and air conditioners.

**Sensor description**: basic RDF ontology (Resource Description Framework)

```java
if my warehouse.motion equals true {
    sms me "person in the warehouse."
}
if my office.smoke equals true {
    sprinklers on;
    call_tts fire-department "fire in the "+[get me.office.address];
    electrical-appliances off;
}
```
Next steps

- **Functionality:**
  - Integrate SER server
  - Integrate LoST server and geo database
  - Finish address book functionality
  - Finish request-based incoming rules
  - Implement request-based outgoing rules

- **Usability:**
  - guided & graphical interface
  - evaluation: can “normal” users create services?
Conclusion

Real world (location & sensors)

Web services (SNs, calendar, contacts, ..)

Communication (VoIP, SMS, email)

SECE