Extending CPL with context ontology

Alisa Devlic
Appear Networks
Kista Science Tower
16451 Kista, Sweden
alisa.devlic@appearnetworks.com

ABSTRACT

Communication has always been an essential part of people's everyday life. Nowadays most of people would like to be reachable on multiple devices at anytime, anyplace. As a consequence, there has been a need to know and exploit a user's availability for communication (so called presence information), so that he/she can control incoming calls and make the decision to accept this call or not based on the user's current context. The appearance and acceptance of Session Initiation Protocol (SIP) as a signalling protocol for next generation networks has opened the door for multiple services, such as Voice over IP (VoIP), chat, games, instant messaging, and other innovative communication services. Built on top of existing data communication networks, this has enabled easy integration of voice and data services. In this paper we present an idea on how to use the context information to enhance the power of existing SIP call control services, to enable users to have greater control over their incoming/outgoing calls. These services are implemented using Call Processing Language (CPL), a language to describe and control Internet Telephony Services. They are extended with context parameters to permit context-based decision making based on context ontology. We want to show how easy is to add new context parameters to the CPL and how complex criteria can be built using our solution.

Keywords

Context-aware, SIP, VoIP, CPL, call processing, ontology

1. INTRODUCTION

The paper describes a motivation and benefits for integrating contextual parameters into call processing capabilities of the existing VoIP system. It also identifies the essential components and concepts needed to realize this solution. By contextual information we mean any information that can characterize a user and his/her current situation, such as: the location, task, activity, time of the day, etc. We have tried to illustrate real life scenarios that would capture a need for context parameters and call processing possibilities an end user would use. From the scenarios we have identified the needed parameters and modelled them in the ontology to represent a user's current context.

In the paper we show how can this context information enhance the functionalities of existing SIP [11] call control services by offering a user the possibility to decide whether to accept an incoming call based on his/her current context. These services are implemented as Call Processing Language (CPL) [10] scripts, and their behavior is described using a set of rules. We have extended the CPL syntax to support rules based on this context information.

We have utilized a scalable and reliable open source SIP platform, called SIP Express Router (SER) [1], to upload and execute CPL scripts. It can act as a SIP registrar, proxy, or redirect server. We have extended its functionality to support our context-based CPL scripts.

The goal of our research work is to show the benefits of extending CPL scripts with context ontology, allowing the easy extensibility and enabling simple CPL to be more powerful.

The paper is organized as follows: first, we give a short introduction of CPL scripts, SER, and call processing capabilities that can be achieved with the existing syntax. Second, we try to illustrate real-life scenarios to indicate the need for context parameters and model them in the ontology. Third, we give a description of our prototype and its components. Finally, we conclude the paper including the plans for the future work.

2. CPL SCRIPTS

CPL scripts are XML-based documents. The Document Type Definition (DTD) is specified in the cpl.dtd file available at [2]. It consists of ancillary information about the script and call processing actions. Ancillary information is information which is necessary for a server to correctly process a script, but which does not directly describe any operations or decisions. A call processing action is a structured tree that describes operations and decisions a telephony signalling server performs upon a call setup event. There are two types of call processing actions: top-level actions and subactions. Top-level actions are actions that are triggered by signalling events that arrive at the server. Two top-level actions are defined: "incoming", the action performed when a call arrives whose destination is the owner of the script,

IMAC Workshop 2006, Espoo, Finland

and "outgoing", the action performed when a call arrives whose originator is the owner of the script. Subactions are actions which can be called from other actions.

The graphical representation of a CPL action is shown in Fig. 1. An action is described by a collection of nodes that describe operations that can be performed or decisions that can be made. A node can have several parameters, which specify the behavior of the node. They usually have outputs, which depend on the result of a decision or action. Nodes are represented as boxes, and outputs as arrows. Nodes are arranged in a tree, starting at a root node. Outputs of nodes are connected to other nodes. When the action of the top-level node is invoked, based on the result of that node a server follows one of the node's outputs, and the subsequent node it points to is invoked. This procedure is repeated until the node with no outputs is reached.

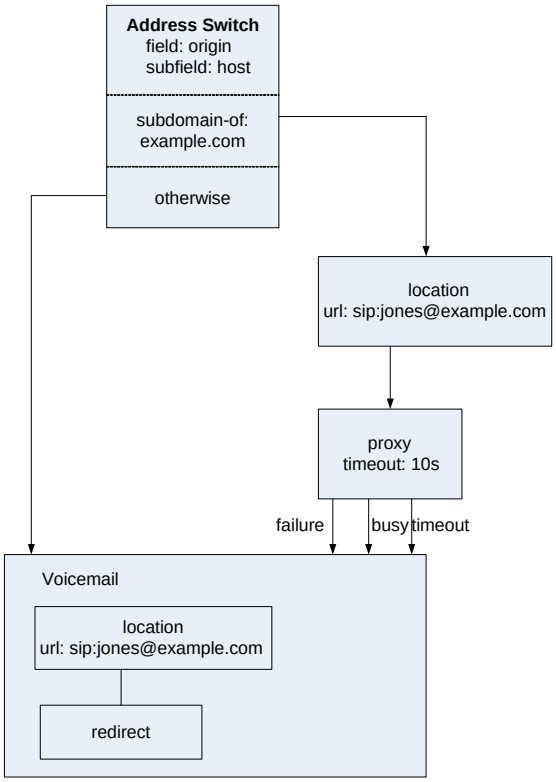

Figure 1: Graphical representation of a CPL script

There are four types of nodes: *switches*, which represent choices a CPL script can make, *location modifiers*, which add or remove locations from the location set, *signalling operations*, which cause signalling events in the underlying protocol, and *non-signalling operations*, which trigger behavior which does not effect the underlying protocol.

CPL scripts can reside on a SIP proxy server, an application server, or intelligent agent. In our case, we have uploaded CPL scripts to the SIP proxy server, SER (Fig. 2). When the SIP INVITE message comes (initiating incoming/outgoing call), SER executes the appropriate part of the user's CPL script that refers to an incoming/outgoing call and manages the call routing logic (accept and route the call to callee, reject the call, forward it to the voicemail, send an e-mail to, redirect, or proxy to some third party). CPL scripts can be uploaded using SIP's REGISTER method or with the aid of graphical programs, such as CPLEd [6].

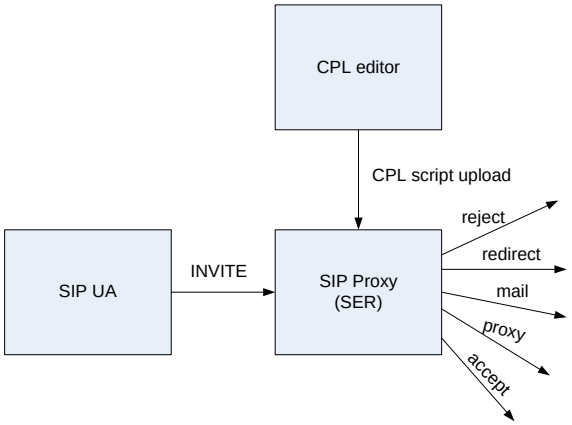

Figure 2: Call processing logic

A CPL script is parsed after uploading to SER. It is stored in an external MySQL database and is loaded and executed upon receiving incoming/outgoing call requests delivered by SIP INVITE messages. The CPL script then processes these calls.

3. APPLICATION SCENARIOS

We have tried to identify the need for context parameters by specifying scenarios that would be applicable in the real life situations. We have also wanted to illustrate call processing possibilities that an end user would use.

1. Alice works for a company "example.com". When she is **in a meeting** and is **presenting** new solution to the management staff, she wants to forward all incoming calls to her voicemail. On the other hand, if she is **listening** to someone's presentation she may want to receive calls that are labeled with an urgent priority on her mobile phone.

2. When she is on **vacation**, Alice wants to reject all incoming calls from the company.

3. When Alice is in her **car**, she would like for safely reasons to set the policy to disable all outgoing calls from her mobile phone while **driving**.

4. When she is on the **business trip** in France, she prefers to get e-mails instead of receiving expensive roaming calls on her mobile phone. However, if the language settings are set to French and the call is coming from her customer's company "trade.com", these calls should be forwarded to her mobile phone.

From the scenarios above we can extract the following context parameters: the **context owner** (the person to whom the context parameters relate to; i.e. Alice), **location** (office, home, vacation, business trip, car), **task** (in a meeting, at lunch), and **activity** (presenting,listening, driving). We

modelled the context parameters in the Web Ontology Language (OWL) ontology, available at [7]. The reason for it is that we can model the parameters into higher-level concepts, and use these concepts in CPL scripts for decision making.

4. CPL EXTENSIONS FOR CONTEXT

CPL extensions for context were designed to describe call processing services related to context relevant to Internet Telephony. We have chosen to utilize context parameters identified in the previous section: context owner, his/her location, task, and activity. In these extensions, we define a **context-switch** to support the services whose decisions are based on the context information of an end user.

In CPL, switches represent choices a CPL script can make based on either attributes of the original call request or other items independent of a call. The existing switches are: address switch, string switch, time switch, priority switch, and language switch, and different screening services can be created based on any of the above switches or combinations. All switches have a list of conditions that can match a variable. When the CPL script is executed, the conditions are checked in the order they are presented in the script. The output of the first matching node is taken. The information affecting the choice is carried in the SIP message.

Adding a context switch allows an end user to make decisions based on the current context parameters of a context owner. The context owner can be the user himself/herself or the user can specify context for some other person. However, we will not consider this later case further in this paper. Values of context parameters are specified in the user's ontology document. The user's context determines which script will be uploaded to the SER. When the context-switch node is invoked, it will match the context parameters set by ontology with context values in the CPL script and return the decision of how to process an incoming/outgoing call (accept, reject, redirect, voicemail, etc.).

Node "context-switch" has one parameter "owner", that identifies a context owner. Node "context" is the output of the "context-switch" node. It specifies different context attributes, such as: "location", "task", and "activity" of a context owner. These attributes were identified after analyzing application scenarios for a typical business user. Syntax of the node "context-switch" and the "context" node is shown in the Table 1.

Table 1: Syntax of a context-switch

Node:	context-switch	context-switch node
Outputs:	context	context parameters
Parameters:	owner	context owner

Output:	context	context node
Parameters:	location	location of a context owner
	task	task status
	activity	activity status

The definition of CPL extensions for context is specified in the file "context.dtd" [8]. An example of CPL script based on this extended CPL is shown below. Jim's SIP proxy server will reject the incoming call if he is in the meeting room called Grimeton, in a meeting, and if he is presenting.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cpl SYSTEM
'file:C:/Programs/CPLEd/context.dtd'>
<cpl>
  <incoming>
    <context-switch owner="jim">
      <context location="grimeton" task="meeting"
               activity="presenting">
        <reject status="reject"
               reason="InMajorMeeting_And_Presenting"/>
      </context>
    </context-switch>
  </incoming>
</cpl>
```

5. CONTEXT-AWARE VOIP PROTOTYPE

The idea of context-aware VoIP prototype was to make call processing dependent on context parameters, so as to make it easier to specify a suitable action to be taken.

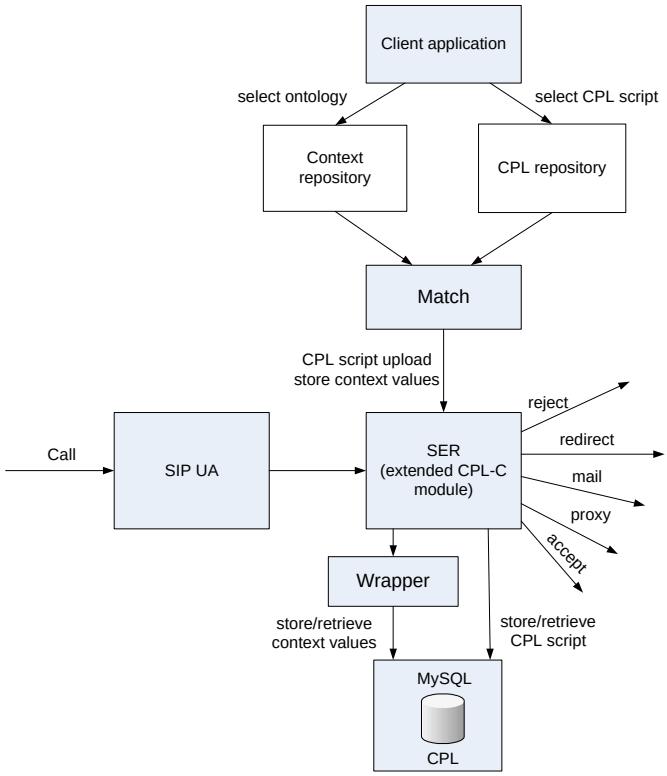

Figure 3: Context-aware VoIP prototype

When the user wants to upload a context-based CPL script (Fig. 3), he/she has to first upload the ontology to the match component, which first parses it, extracts the user's context parameter values, and stores them into the external MySQL database (that is also used by SER for storing users and CPL scripts). Second, the match component matches context values with the corresponding values in available CPL scripts to determine which script describes rules for the current user's context. Before they are uploaded to SER, these CPL scripts are stored in a CPL repository, while ontologies

reside in a context repository. Upon receiving a call or SIP INVITE message from a SIP User Agent (SIP UA), SER loads the user's current CPL script from the database and executes it. If the CPL script contains a context-switch, it will match values set in script rules with the corresponding context values, and if they match, take appropriate actions. The wrapper component is used by SER to retrieve context values from the database.

The prototype that we implemented in the lab consists of four components: a client application, match component, wrapper, and extensions to the CPL-C module [5] of SER.

5.1 Client application

A simple client application is used for uploading ontologies and CPL scripts (Fig. 4). CPL scripts that are not context-based can be uploaded directly, without the need to first upload the context ontology. The aplication was designed to be used from different machines and different locations, hence the preferable implementation is as an applet.

Figure 4: Client application

Note that this applet was built as a proof of concept only. The alternative solution is to have two clients (applets), one for uploading context(ontology) and another for uploading scripts. The applet opens the file chooser dialog (Fig. 5) to browse for a file to open (i.e. in this case ontology).

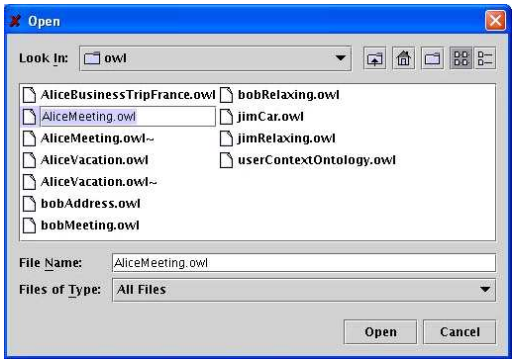

Figure 5: File chooser dialog

5.2 Match component

The match component is responsible for parsing the selected ontology to get context values, determine the appropriate CPL script, and upload that script via SIP (or HTTP(S)) protocol to SER. Both choices are available, but we mainly focused on SIP in this prototype. SER will, upon receiving the script, store it in the database under the supplied user's credentials.

5.3 Wrapper

The wrapper was created to pass context values client application, match component, and SER. The context parameters are stored in the database when the ontology is parsed, and retrieved by the wrapper program when the script is executed.

5.4 CPL-C module extensions

We had to modify the cpl-c module of the SER source code to support adding of our context-switch and context node. This is explained in more detail in [9].

6. CONCLUSIONS

We described in the paper the solution on how to extend CPL with context parameters to make call decision-making process more powerful. In the initial measurements, we saw that the SER server with CPL module is very scalable - it can register 4000 users in 10 seconds [12]. When we add a context-switch at a time to the script, the response time increases about 5%-24%, what we expected, because of added reasoning process and storage of values in the database [9]. In the future work we plan to: a) build a proxy proxy between SIP UA and SIP proxy to be able to make "advanced" calls based on the call priority, language that caller has set, and a free form string set, b) add sensor services like positioning systems (e.g. Cell-ID, GPS) and Calendar service to set the available context parameters: location, task, and activity in the ontology, c) find the way how to dynamically plug-in new sensor services from the environment and use them as context providers, d) make an ontology and CPL script uploaded automatically by the system, and not explicitly by the user, and e) do more measurements of initiating call requests from multiple users in the time to determine the SER's average and peak call processing capability.

7. BIOGRAPHY

I work as research engineer and industrial PhD student in the company Appear Networks, which is involved in two EU FP6 projects: MIDAS (Middleware Platform for Developing and Deploying Advanced Mobile Services) [3] and SIMS (Semantic Interfaces for Mobile Services) [4]. Our research group is looking the way to design a middleware which will simplify and speed up the task of developing and deploying mobile applications and services, taking into account large number of users, their context information, limited connectivity infrastructure, and short notice communication setup. We also investigate the usage of semantic interfaces for rapid development, dynamic discovery, and composition of mobile services.

This research work was carried out in the Wireless center of the Royal Institute of Technology and I would like to thank my advisor, Prof. Gerald Q. Maguire Jr., for his help and guidance.

8. REFERENCES

[1] CPL XML DTD draft. http://www.iptel.org/ser/, April 2006.

[2] SIP Express Router (SER). http://xml.coverpages.org/CPL-DTD-200201.txt, January 2002.

[3] MIDAS (Middleware Platform for Developing and Deploying Advanced Mobile Services) project. http://www.ist-midas.org, January 2006.

[4] SIMS (Semantic Interfaces for Mobile Services) project. http://www.ist-sims.org/, January 2006.

[5] CPL-C module of SER. http://www.voip-info.org/wiki-SIP+Express+Router, June 2006.

[6] CPLEd, a free graphical CPL editor. http://www.iptel.org/products/cpled/, September 2002.

[7] A. Devlic. Context ontology. http://web.it.kth.se/ devlic/userContextOntology.owl, June 2006.

[8] A. Devlic. CPL extensions for context DTD. http://web.it.kth.se/ devlic/context.dtd, June 2006.

[9] A. Devlic. CPL extensions. Report for the Practical VoIP course, http://web.it.kth.se/ devlic/CPL

[10] J.Lennox, X.Wu, and H.Schulzrinne. Call Processing Language: A Language for User Control of Internet Telephony Services. RFC 3880, http://www.ietf.org/rfc/rfc3880.txt, October 2004.

[11] J.Rosenberg, H.Schulzrinne, G.Camarillo, A.Johnston, J.Peterson, R.Sparks, M.Handley, and E.Schooler. SIP: Session Initiation Protocol. RFC 3261, http://www.ietf.org/rfc/rfc3261.txt, June 2002.

[12] Y. Oukhay. Context-aware services. In *Master of Science Thesis*. Royal Institute of Technology, March 2006.