Detecting Semantic Ambiguity in Service Description of Internet Telephony

Pattara Leelaprute[1], Masahide Nakamura[2], Tohru Kikuno[1]
[1]Department of Informatics and Mathematical Science,
Graduate School of Engineering Science Osaka University, Japan
[2]Graduate School of Information Science
Nara Institute of Science and Technology, Japan
pattara@ics.es.osaka-u.ac.jp

Abstract

The Call Processing Language (CPL, in short), recommended in RFC 2824 of IETF, is a service description language for the Internet Telephony. The CPL allows users to define their own services, which dramatically improves the choice and flexibility in service creation by the users. However, there are enough rooms for non-expert users to make semantic mistakes in the service logic. In this paper, we propose six classes of semantic warnings for the CPL service description. These warnings are not necessarily errors, but will help users to find ambiguity, redundancy and inconsistency in their own service description. We also present a tool, called CPL semantic checker for detecting the semantic warnings. The tool performs not only detection of the proposed semantic warnings, but also checking syntax well-formedness and DTD conformance for the given CPL script.

Key-Words: Internet telephony, semantic warnings, VoIP, CPL, Feature Interaction

1. Introduction

As the Internet is widely spread in society, high-quality services with the Internet are required. Among the various Internet services, this paper especially focuses on the *Internet telephony* [3], which is also called *Voice over IP*, (VoIP, in short). The Internet telephony has been widely studied and standardized at the protocol level (i.e., H323[5] by ITU-T, SIP[4] by IETF). Now, the concern is shifting to the service level; how to provide supplementary services (e.g., call forwarding, voice mail, etc.) on the Internet telephony.

One of the major issues is the *programmable service*, which allows users to define and create their own supplementary services. The *Call Processing Language* [2] (CPL, in short), based on XML, is recommended as a service description language in RFC2824 of the Internet Engineering Task Force (IETF) [1]. Users can deploy their own service just by putting the CPL scripts in the local VoIP server (called *signaling server*). This improves the range of user's choice and flexibility in service creation, significantly.

There is, however, a drawback of the programmable service. The service description of non-experts cannot always achieve the high quality. Also, users might make faults in the CPL scripts that lead to serious system down.

To cope with this problem, this paper tries to characterize *semantic warnings* of service description written in the CPL. As seen in many programming languages, the warnings are not necessarily errors. However, they could cause ambiguity, redundancy and inconsistency, which are often the major source of errors. We believe that the proposed warnings will help users to improve the quality of the CPL scripts.

2. Describing services with CPL

Let us define a new service based on the following requirements, using CPL:

- I (`pattara@example.com`) want to receive incoming calls only from domain `example.com`.
- I want to reject all calls from malicious crackers (belonging to `crackers.org`).
- I want to redirect any other calls to my voice mail (`pattara@voicemail.example.com`).

Figures 1 and 2 depict the requirement and an implementation of the service, respectively. The first two lines are for the declaration of XML and DTD (Document Type Definition). The tag `<cpl>` means the start of a body of the CPL script. The portion surrounded by `<subaction> </subaction>` defines a subaction, which is a sub-routine called from the main-routine. `<incoming>` tag specifies actions activated when an incoming call is received.

Next, <address-switch> allows the CPL to have a conditional branch with respect to the addresses. In this example, the condition is extracted from the host address of the caller (field= "origin" subfield=host). If the host's domain matches example.com (<address subdomain-of= "example.com">), then the location is set to sip:pattara@example.com, and the call is proxied there (<proxy />). If the domain matches crackers.org, the call is rejected (<reject status="reject" />). Otherwise, the subaction voicemail is called. In the subaction voicemail, the location is set to pattara@voicemail.example.com, and the call is redirected there. That is, the caller places the call again to the new address. For the detailed definition of CPL, please refer to the full specification [2].

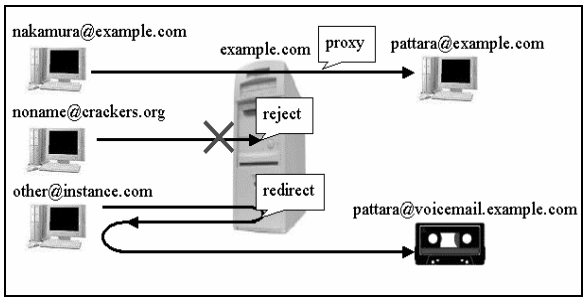

Figure 1. Behavior of the requirement

```
1: <?xml version="1.0" ?>
2: <!DOCTYPE cpl PUBLIC "-//IETF//DTD
3:                   RFCxxxx CPL 1.0//EN" "cpl.dtd">
4: <cpl>
5:  <subaction id="voicemail">
6:    <location url="sip:pattara@voicemail.example.com">
7:      <redirect/>
8:    </location>
9:  </subaction>
10:  <incoming>
11:    <address-switch field="origin" subfield="host">
12:      <address subdomain-of="example.com">
13:        <location url="sip:pattara@example.com">
14:          <proxy />
15:        </location>
16:      </address>
17:      <address subdomain-of="crackers.org">
18:        <reject status="reject" />
19:      </address>
20:      <otherwise>
21:        <sub ref="voicemail"/>
22:      </otherwise>
23:    </address-switch>
24:  </incoming>
25: </cpl>
```

Figure 2. Example of a CPL script

3. Characterizing Semantic Warnings

The CPL is a relatively simple language, as it has no variables, loops, or ability to run external programs. This allows simple but strict syntax definition by the DTD, and minimizes such complex semantic errors as the ones in the general programming languages [2]. However, compliance with the DTD is not a sufficient condition for correctness of a CPL script. There are enough rooms for *non-expert users* to make various mistakes, which make the CPL scripts complex, ambiguous and inconsistent.

Here we propose six classes to be considered as the semantic warnings. These might not be necessarily errors, but should be avoided. In the following subsections, we present a definition and its effects. To help the comprehensions, we also give an example for each semantic warning. Due to the limited pages, we reuse the script in Figure 2, and modify the script so that it contains a typical semantic warning. These examples are also available at http://www-kiku.ics.es.osaka-u.ac.jp/~pattara/CPL/ .

3.1 Multiple forwarding addresses (MF)

Definition: After multiple addresses are set by <location> tags, <proxy> or <redirect> comes.

Effects: The CPL allows calls to be proxied (or redirected) to multiple address locations by cascading <location> tags. However, if the call is redirected to multiple locations, the caller would confuse to which address the next call should be placed. Or, if the call is proxied, a race condition might occur depending on the configuration of the proxied terminals. As a typical example, if a user simultaneously sets the forwarding address to his handy phone and voice mail that immediately answers the call. Then the call never reaches his handy phone.

Example: Figure 3 shows an example. The call from anybody@example.com is proxied to the terminal and voicemail simultaneously. If the voice mail is configured so as to immediately answer the call, then the call never reaches the terminal.

3.2 Identical switches with the same parameters (IS)

Definition: After a switch tag with a parameter, the same switch with the same parameter comes.

Effects: The CPL has no variables or no loop. So, a condition evaluated in the former switch tag never changes in the latter switch tag. Hence, the conditional branch specified in the latter switch is in vain, since the condition must have been evaluated already. This would increase the ambiguity of the CPL script.

Example: Figure 4 shows an example. When a call is arrived, this script will check the originator's host domain. If it matches `example.com`, the call will be proxied to `pattara@example.com`. Otherwise, the call processing proceeds to `<otherwise>` block. However, here the originator's domain is checked again if it matches `example.com`. This condition has been already evaluated, and it never holds since it is in `<otherwise>` block. As a result, the subaction voicemail is never executed. Thus, the second switch is redundant and meaningless.

3.3 Call rejection in all paths (CR)

Definition: All execution paths terminate at `<reject>`.

Effects: No matter which path is selected, the call is rejected. No call processing is performed, and all executed actions and evaluated conditions are nullified. This is not a problem only when the user wants to reject all calls explicitly. However, complex conditional branches and deeply nested tags will make this problem difficult to be found, on the contrary to user's intention.

Example: Figure 5 shows an example. By this script, any incoming call is rejected, no matter who is the originator. All actions and evaluated conditions are meaningless after all.

3.4 Address set after address switch (AS)

Definition: When `<address>` and `<otherwise>` tags are specified as outputs of `<address-switch>`, the same address evaluated in the `<address>` is set in the `<otherwise>` block.

Effects: The `<otherwise>` block is executed when the current address does not match the one specified in `<address>`. If the address is set as a new current address in `<otherwise>` block, then a violation of the conditional branch might occur. A typical example is that, after screening a specific address by `<addressswitch>`, the call is proxied to the address, although any call to the address must have been filtered.

Example: Figure 6 shows an example. When the user makes an (outgoing) call, this script will check the destination of the call. The call should be rejected if the destination address is `pattara@example.com`, according to the condition specified in `<address>`. However, in the `<otherwise>` block, the call is proxied to `pattara@example.com`, which must have been rejected.

```
1: <?xml version="1.0" ?>
2: <!DOCTYPE cpl PUBLIC "-//IETF//DTD
3:                   RFCxxxx CPL 1.0//EN" "cpl.dtd">
4: <cpl>
5:  <subaction id="voicemail">
6:   <location url="sip:pattara@voicemail.example.com">
7:    <redirect/>
8:   </location>
9:  </subaction>
10:  <incoming>
11:   <address-switch field="origin" subfield="host">
12:    <address subdomain-of="example.com">
13:     <location url="sip:pattara@example.com">
14:      <proxy />
15:     </location>
16:    </address>
17:    <address subdomain-of="crackers.org">
18:     <reject status="reject" />
19:    </address>
20:    <otherwise>
21:     <address-switch field="origin" subfield="host">
22:      <address subdomain-of="example.com">
23:       <sub ref="voicemail"/>
24:      </address>
25:     </address-switch>
26:    </otherwise>
27:   </address-switch>
28:  </incoming>
29: </cpl>
```

Figure 3. Example of MF

```
1: <?xml version="1.0" ?>
2: <!DOCTYPE cpl PUBLIC "-//IETF//DTD
3:                   RFCxxxx CPL 1.0//EN" "cpl.dtd">
4: <cpl>
5:  <subaction id="voicemail">
6:   <location url="sip:pattara@voicemail.example.com">
7:    <redirect/>
8:   </location>
9:  </subaction>
10:  <incoming>
11:   <address-switch field="origin" subfield="host">
12:    <address subdomain-of="example.com">
13:     <location url="sip:pattara@example.com">
14:      <location url=
15:            "sip:pattara@voicemail.example.com">
16:       <proxy />
17:      </location>
18:     </location>
19:    </address>
20:    <address subdomain-of="crackers.org">
21:     <reject status="reject" />
22:    </address>
23:    <otherwise>
24:     <sub ref="voicemail"/>
25:    </otherwise>
26:   </address-switch>
27:  </incoming>
28: </cpl>
```

Figure 4. Example of IS

```
1: <?xml version="1.0" ?>
2: <!DOCTYPE cpl PUBLIC "-//IETF//DTD
3:                  RFCxxxx CPL 1.0//EN" "cpl.dtd">
4: <cpl>
5:   <incoming>
6:     <address-switch field="origin" subfield="host">
7:       <address subdomain-of="example.com">
8:         <location url="sip:pattara@example.com">
9:           <reject status="reject" />
10:        </location>
11:      </address>
12:      <address subdomain-of="crackers.org">
13:        <reject status="reject" />
14:      </address>
15:      <otherwise>
16:        <reject status="reject" />
22:      </otherwise>
23:    </address-switch>
24:  </incoming>
25: </cpl>
```

Figure 5. Example of CR

```
1: <?xml version="1.0" ?>
2: <!DOCTYPE cpl PUBLIC "-//IETF//DTD
3:        RFCxxxx CPL 1.0//EN" "cpl.dtd">
4: <cpl>
5:   <outgoing>
6:     <address-switch field="destination">
7:       <address is="sip:pattara@example.com">
8:         <reject status="reject"
9:            reason="I don't call Pattara" />
10:      </address>
11:      <otherwise>
12:        <location url="sip:pattara@example.com">
13:          <proxy/>14:        </location>
15:      </otherwise>
16:    </address-switch>
17:  </outgoing>
18: </cpl>
```

Figure 6. Example of AS

3.5 Unused Subactions (US)

Definition: Subaction <subaction id= "foo" > exists, but <subaction ref= "foo" > does not.

Effects: The subaction is defined but not used. The defined subaction is completely redundant, and should be removed to decrease server's overhead for parsing the CPL script.

Example: Figure 7 shows an example. In this script, a subaction "`voicemail`" that was declared in the subsection part is not used in the body of the script. So, the unused subaction "voicemail" is redundant and should be removed.

3.6 Overlapped Conditions in Switches (OS)

Definition: The condition is overlapped among the multiple output tags of a switch.

Effects: According to the CPL specification, if there exist multiple output tags for a switch, then the condition is evaluated in the order the tags are presented, and the first tag to match is taken. If the conditions specified in the outputs are overlapped (or identical), then the former tag is always taken. In extreme cases, the latter tag is never executed, which is a redundant description.

Example: Figure 8 shows an example. When a call reaches the user, this script will check the destination of the call. If the destination address contains `pattara`, the call will be proxied to his home address. If the destination address is `pattaraleelaprute`, this script tries to proxy the call to his mobile address. But in fact, the call is never proxied to `pattaraleelaprute`, since `pattara` is a substring of `pattaraleelaprute`, thus the first branch is always taken.

```
1: <?xml version="1.0" ?>
2: <!DOCTYPE cpl PUBLIC "-//IETF//DTD
3:                  RFCxxxx CPL 1.0//EN" "cpl.dtd">
4: <cpl>
5:  <subaction id="voicemail">
6:    <location url="sip:pattara@voicemail.example.com">
7:      <redirect/>
8:    </location>
9:  </subaction>
10:  <incoming>
11:    <address-switch field="origin" subfield="host">
12:      <address subdomain-of="example.com">
13:        <location url="sip:pattara@example.com">
14:          <proxy />
15:        </location>
16:      </address>
17:      <address subdomain-of="crackers.org">
18:        <reject status="reject" />
19:      </address>
20:    </address-switch>
21:  </incoming>
22: </cpl>
```

Figure 7. Example of US

```
1: <?xml version="1.0" ?>
2: <!DOCTYPE cpl PUBLIC "-//IETF//DTD
3:        RFCxxxx CPL 1.0//EN" "cpl.dtd">
4: <cpl>
5:   <incoming>
6:     <address-switch field="destination" >
7:       <address contains="pattara">
8:         <location url="sip:pattara@home.example.com">
9:           <proxy />
10:        </location>
11:      </address>
12:      <address is="pattaraleelaprute">
13:        <location url=
14:          "sip:pattaraleelaprute@home.example.com">
15:          <proxy />
16:        </location>
17:      </address>
18:    </address-switch>
19:  </incoming>
20: </cpl>
```

Figure 8. Example of OS

4. Developing CPL semantic checker

Based on the proposed definitions, we have developed software, called *CPL semantic checker*, to detect the semantic warnings. For a given CPL script, it can be used not only to detect the semantic warnings, but also to perform the syntax checking. That is, the CPL semantic checker performs (1) checking well-formedness of XML syntax, (2) validation of the CPL against the DTD, and finally (3) detection of the semantic warnings.

The CPL semantic checker is a CGI program implemented by the Perl language. By extensively utilizing open-source modules XML::Parser and XML::DOM::Parser, the CPL semantic checker itself is a very light-weight program consisting of 518 lines of codes. It is freely available at the URL http://www-kiku.ics.es.osaka-u.ac.jp/~pattara/CPL/ .

When you access the URL by web browsers, then an input interface, as shown in Figure 9, will appear. The interface contains a text box for putting a CPL script in. You can either directly write the CPL script or paste it from clipboard. To reset the input CPL script, click the "Reset" button. To validate the CPL script, click "Validate" button below the CPL input windows. Then the CPL semantic checker reports syntax checking and semantic warnings described in Section 3. The reports are shown sequentially below "Validate" and "Reset" buttons. If input CPL script is free from errors, a message "No error found." would be shown. If the given CPL script has semantic errors, the following messages would be shown:

!!Error=MF, found after proxy *(or redirect)* in line *(line number)*.

!!Error=IS, parameter in line *(line number)* same as parameter in line *(line number)*.

!!Error=CR, call rejected in all paths.

!!Error=AS, address is="address" in line *(line number)* same as location url="address" in line *(line number)*

!!Error=US, declared "subaction name" in line *(line number)* is unused.

!!Error=OS, "first condition" in line *(line number)* contain "second condition" in line *(line number)*

Figure 5 shows an example of the execution, showing that the input script contains semantic warning MF and US.

The CPL semantic checker also can be used as a module for CPL feature server and/or SIP Proxy server [6], in order to check the validity of the CPL script at run time. We are currently extending its interface.

5. Conclusion

In this paper, we have proposed the six classes of semantic warnings in CPL service description. Although there might exist other types of semantic warnings (thus, some quantitative evaluation is needed), we believe that the proposed warnings would contribute to describing consistent service logic in the CPL. We have also presented a tool, CPL semantic checker, to check the semantic warnings as well as the syntax error and DTD conformance.

Once each individual service is guaranteed to be consistent, then we have to tackle the next tough problem: Feature Interaction [7]. The Feature Interaction is known as a kind of inconsistent conflict between multiple services. It refers to situations where a combination of different services behaves differently than expected from the single services' behaviors. One of our future work is to establish a framework to detect the *Feature Interaction* in the CPL scripts. As a first step, we need to extend the definition of semantic warnings so as to cover the inconsistency over multiple scripts. As described in [1], there has been no effective solution for the Feature Interaction problem in the Internet Telephony, since the CPL scripts can be updated anytime by users. Therefore, we need to examine the architecture for detecting and resolving the Feature Interaction at run-time.

Acknowledgments

This work is partly supported by Grant-in-Aid for Encouragement of Young Scientists (No.13780234), from Japan Society for the Promotion of Scienc

12. References

[1] J.Lennox and H.Schulzrinne,"Call processing language framework and requirements," Request for Comments 2824, Internet Engineering Task Force,May 2000.
http://www.ietf.org/rfc/rfc2824.txt?number=2824

[2] J.Lennox and H.Schulzrinne,"CPL:A Language for User Control of Internet Telephony Service," Internet Engineering Task Force,Jan 2002. http://www.ietf.org/internet-drafts/draft-ietf-iptel-cpl-06.txt

[3] H.Schulzrinne and J.Rosenberg, "Internet Telephony: Architecture and protocols - an IETF perspective," Computer Networks and ISDN Systems, vol.31, pp.237-255, Feb 1999.

[4] M.Handley, H.Schulzrinne, E.Schooler, and J.Rosenberg, "SIP: session initiation protocol," Request for Comments 2543, Internet Engineering Task Force, Feb 2002.
http://www.ietf.org/internet-drafts/draft-ietfsip-rfc2543bis-09.txt

[5] ITU-T Recommendation H.323, "Packet-Based Multimedia Communications Systems," February 1998.

[6] VOVIDA.ORG, “Vovida Open Communication Application Library (VOCAL)”, http://www.vovida.org
[7] Keck, D.O. and Kuehn, P.J., “The feature interaction problem in telecommunications systems: A survey,” IEEE Trans. on Software Engineering, Vol.24, No.10, pp.779-796, 1998.

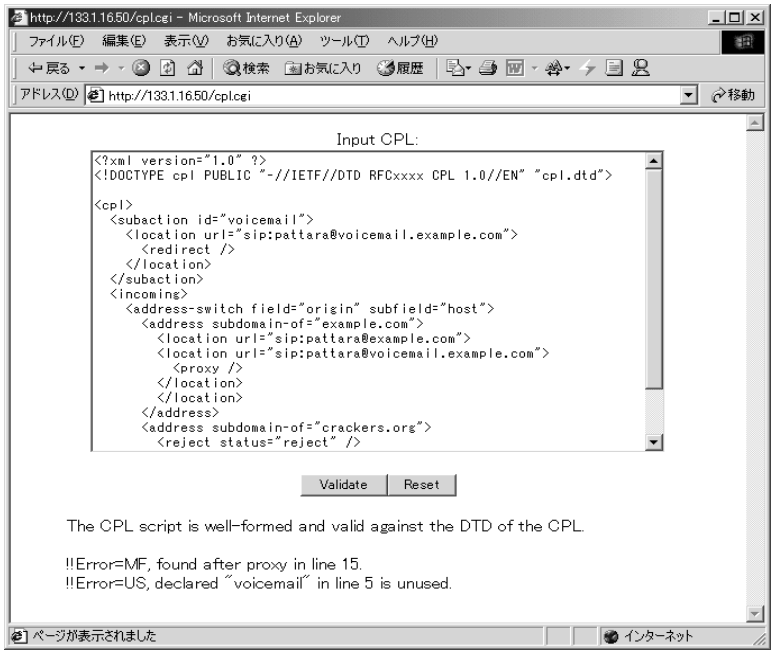

Figure 9. Execution example