Online Detection of Feature Interactions of CPL Services

Lixiang Wang[a], Jiuyun Xu[a,1], Stephan Reiff-Marganiec[b]
[a] *School of Computer and Communication Engineering, China University of Petroleum, email：jiuyun.xu@googlemail.com*
[b] *Department of Computer Science, University of Leicester, srm13@le.ac.uk*

Abstract. The Call Processing Language (CPL) is one of the best known approaches developed for creating telephony services in Internet telephony. Its XML-based structure makes it a flexible approach to create services not only by service providers, but also by end-users, which increases the possibility of feature interactions. In this paper we present an online approach for detecting feature interactions of CPL. A feature interaction manager is introduced to handle the problem. The manager detects feature interaction through comparing actions indicated by real-time SIP messages and actions indicated by CPL scripts which we call intentions. We evaluated the proposed approach using the VOCAL (Vovida Open Communication Application Library) VoIP system. Among the ready-made features, a feature interaction has been detected between Call Screening and Call Forward All Calls.

Keywords. Feature Interaction, Online Approach, CPL, SIP, VoIP

1. Introduction

A feature is a component of additional functionality added to a base system[1], which has the purpose of fulfilling certain user intentions – in call processing systems these intentions usually change the call behaviour. Features are expected to work correctly on the base of some assumptions. Features may have to interwork to share a resource. When features interwork, some may break the assumption of others, or the goals of the different features conflict with each other, and thus feature interactions happen. For example in a call processing system, user A has an Originating Call Screening (OCS) feature, with user C on its screening list, and user B has a Call Forward Unconditionally (CFU) feature to forward all incoming call to C. If user A calls user B, a connection between A and C is established according to CFU, but this conflicts with the goal of OCS – hence we say that features OCS and CFU interact. While this is a simple example, the problem is getting worse with an increasing number of features, higher complexity of each feature and if non-experts (that is end-users) can define their own features.

The feature interaction problem has been extensively studied in traditional telephone system[2], and a substantial work has been done to deal with feature interactions. Generally, approaches can be divided into two categories: off-line and on-

[1] Corresponding Author: Jiuyun Xu, xujiuyun@ieee.org

line. Off-line approaches are applied at design-time and are often based on formal methods. Off-line approaches have a number of shortcomings, which are especially apparent in an open and competitive market. On one hand, detailed information of the feature is not always available for commercial reasons as features are provided by a number of providers (which might also be independent of the provider of the interaction detection solution). On the other hand, the rapidly increasing number of service makes it an arduous, if not impossible, task to analyze pair-wise feature interactions. However, these approaches do have significant value when applied to a group of services provided by the same party.

On-line approaches are applied at run-time. While this still does not provide detailed implementation information about features, it provides similar detail, namely that observable from the features behavior, about features from all providers. More crucially, it allows to focus on features involved in the current call rather than requiring an analysis of all possible combinations. These approaches have clear benefits in open market environments.

The Call Processing Language (CPL)[3], standardized by IETF, is one of the best known approaches for creating telephony services for Internet telephony. There are other approaches including SIP CGI (Common Gateway Interface), SIP Servlets, JAIN APIs. However, CPL is one of the most popular methods because it is signaling independent and designed to ensure safety. Another important reason for the popularity is that CPL is XML-based, and hence it is easy to read and write by non-professionals, and hence end-users can write (and subsequently deploy) their own scripts. However, the potential for interactions increases the larger the number of deployed features becomes and also increases significantly if lay-users build their own features as they tend to be less aware of the consequences of their wishes on the overall system than a telecommunications engineer. Feature interaction in CPL has been studied in [4-6], where the authors adopt off-line approaches. In this paper, we propose an on-line method to detect feature interaction which should be more suitable for CPL due to the general benefits of on-line approaches discussed above. The approach is based on a feature interaction manager, which detects feature interactions by analysing actions indicated by real-time SIP messages and actions (referred to as intentions) indicated in CPL scripts. The approach can detect Single-Component[7] feature interactions and we have verified this approach on an open-source VoIP system, VOCAL (Vovida Open Communication Application Library). While the approach has been developed for CPL, it is not limited to it. It can also be applied to detect feature interactions implemented by other approaches where the system actions and the intended behavior of features can be observed.

The remainder of this paper is structured as follows: The next section presents an introduction to CPL and related basic concepts. Section 2 proposes an on-line feature interaction detecting approach. Section 3 discusses how the approach works and section 4 illustrates the implementation of the approach with VOCAL and presents a case study. Section 5 discusses related work. Finally, we conclude and provide an outline of further research.

2. Background

2.1 Call Processing Language

CPL defines a scripting language, where the scripts describe the desired behavior for occurring actions and certain conditions. Crucially, loops and recursions are not allowed in CPL scripts, as these can easily lead to potential errors (that is a single CPL script not working correctly). CPL scripts can either be deployed on network servers or user agents.

CPL defines to types of call processing actions: incoming and outgoing which are actions performed when a call arrives whose destination is the owner of the script or when a call is made where the script owner is the origin respectively. These two actions are seen as top-level actions, with subactions being actions which can be called from other actions. In general each top action is root to a tree of nodes and outputs.

There are four categories of CPL nodes: *switches*, including address-switch and string switch, which represent choices a CPL script can make; *location modifiers*, which add or remove locations from the location set; *signaling operations*, which cause signaling events in the underlying protocol; and *nonsignaling operations,* which trigger behavior that does not affect the underlying protocol.

Semantically, CPL scripts can be considered as a decision tree. The tree describes the decisions and operations a telephony signaling server performs on a call set-up event. Switches and location modifiers compose conditions to describe whether the following operations should be taken or not, and the operations describe what the manager or user wants to do with the call depending on the condition(s).

2.2 How CPL Controls SIP Message Flow

CPL was designed to describe Internet telephone services. It works only when parsed and used in conjunction with a signaling architecture or protocol. Here we introduce how signaling operations affect SIP message flow, other uses of CPL are introduced in RFC3880 [3].

As we discussed in section 2.1, switches and location modifiers help make decisions, and finally signaling operators complete the call control by “proxy”, “redirect”, or “reject” actions. “proxy” means to forward the call to the currently specified location; “redirect” causes the server to direct the calling party to place its call to the specified set of locations; “reject” tells the calling party that the call has been rejected. A SIP system operates enforces the suggested behaviour:

(1) If the signaling operation is “proxy”, the feature server will change the request URL with the URL obtained from the switch condition or the result of a lookup to generate a new request, and forward it accordingly. A response will be received to the call request according to the system condition or other service.

(2) If the signaling operation is “redirect”, the feature server will generate a 3xx response related to the INVITE message with the new contact address in the CONTACT field – it is then up to the user agent to re-attempt the call.

(3) If the signaling operation in the script is “reject”, the feature server will return a response in the 4xx, 5xx, 6xx range to the originator to tell them the call is rejected, and indicate the reason is whether busy, not found, reject, or error.

2.3 Basic Concepts

Several basic concepts are important in the study of feature interaction in CPL, such as policy, feature and intention. In [8], policy has been defined as "high-level statements to support personal, organizational, or system goals". An intention is one of these goals. A feature is a basic functionality offered by a system and is specified by means of policies

In this paper, we represent intention by a potential connection as indicated jointly by the SIP message and the applicable CPL scripts. Potential connection is used here to represent the relationship of two users that may be connected together in the call set up stage. The features involved and the SIP messages suggest potential connections: For example, a SIP INVITE message, defines a potential connection between the user in the From header and the user in the request line; if a SIP INVITE message from A to B arrives at a server and B has subscribed a feature CFU with all incoming calls forwarded to C, a potential connection between A and C is indicated. The word "potential" is used because the relationship may be changed by a feature or a redirect operation, and hence any connection suggested by the basic call or any individual feature is not necessarily the finally established connection.

In the example above, the potential connection between A and C is the intention of the feature CFU. It is obtained by composing information from the SIP INVITE message and the feature.

If a feature is triggered, its intention must be embodied in the new SIP message generated after the feature is executed. We use intentions of features and potential connections of SIP messages, instead of only SIP messages, considering the condition that some feature may not be triggered, but the resulting connection of the other feature may conflict with its intention. A typical example is the feature interaction between OCS and CFU. We will discuss this in detail in section 3.2.

3. The Approach

Feature interactions result in conflicts of potential connections in the call set up stage. Potential connections can be separated into two categories: allowed connections and forbidden ones. If one feature intends to forbid a connection of two users, while another feature tries to establish the same connection, feature interaction occurs between the two features. Our approach is based on this understanding. Furthermore, the two categories of potential connection can be represented by certain string items, so feature interaction can be detected simply by string comparison without complex logic decisions. A feature interaction manager is used in our approach to detect feature interactions. Single-Component interactions can be detected by this method, and this has been implemented with VOCAL as a testbed. The approach can be extended to Multi-Component interactions, and we will come back to this.

3.1 Representations of Call States

Let commu(x, y) represent a connection from user x to user y, and we refer to this as a communication item. Two lists are used to carry the items: the connection list contains connections or feature intentions that are permitted, and the no-connection list connections that are forbidden.

An example will illustrate this. A SIP INVITE message arrives at a server with Alice in the From header and Bob's URL in the request line. The condition can be described as Figure 1.

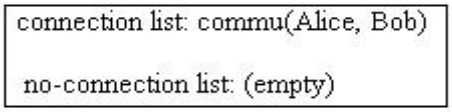

Figure 1. Representation of a SIP INVITE message

Consider a more complex example in terms of a CPL feature. User Alice has a service which forbids all outgoing calls to Carl and proxies all other outgoing calls to Bob. The CPL script is shown in Figure 2, and its intention is represented as shown in Figure 3.

```
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">
<cpl>
<outgoing>
      <address-switch field="destination">
          <address is="sip:Carl@example.com">
               <reject status="reject" reason="I don't call Carl" />
          </address>
     <otherwise>
          <location url="sip:Bob@example.com">
               <proxy/>
          </location>
     </otherwise>
    </address-switch>
</outgoing>
</cpl>
```

Figure 2. An example CPL script

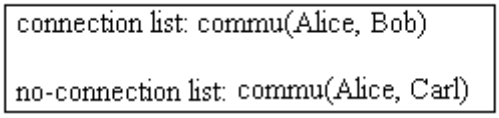

Figure 3. Representation of CPL script in Figure 3-2

Clearly the lists are only listing connections that have been defined clearly by the SIP messages or the call script. Connections that are not included in the description are allowed by default. For example, if user A has an OCS service with only B on his screening list, connection from A to all the other users are automatically allowed (at least they are not forbidden by the OCS feature, though some other feature might forbid

them). Hence, we assume that connections which are not in the connection or no-connection lists are allowed.

The representation used in this approach is inspired by Richard E. Fikes and Nils J. Nilsson [9]. They once used the similar representation method in the artificial intelligence area to represent operator results, however we find it is also helpful here for the feature interaction detection problem.

3.2 Feature Interaction Analysis

Feature interaction means either conflicts between intentions expressed by different features or that the resulting connection created by the features is logically unreasonable. In the first case, the intention of at least one of the features will be changed or not executed as expected (this is usually the intention of the first feature to be invoked). To detect this interaction, the connection indicated by the first feature's intention and the resulting connection of the second feature should be compared to detect feature interaction. In the latter case, each features' intention is satisfied, but the final results is logically conflict.

Feature interaction can be divided into three categories based on this understanding. Note that in this paper we consider only the situation of two features interacting with each other based on the assumption that the community has not really identified much need for three-way (or more-way) detection.

Category 1, Conflict between two features' intentions. Feature interaction happens only after the second feature is triggered, so this kind of feature interaction can be detected by comparing the first feature's intention and the connection set up after activation of the second feature. This kind of interaction has two cases.

(a) During the call setup process, the first feature which is invoked by one of the subscribers involved in the call is not triggered by the INVITE message, because the service specific data is not satisfied. For example, subscriber A has an OCS service with C on its screening list. Subscriber B has a CFU service which redirects all incoming calls to C. While A calls B, OCS is not triggered, and CFU redirects the call to C so a connection between A and C is established. This clearly conflicts with the intention of OCS. Using our representation method, the intention of the first feature and the resulting connection of the second feature are captured as shown in Figure 4. This behavior is similar to sequential action interactions (SAI) defined in[10].

(b) Both features are triggered by the INVITE message, but the second feature changes the intention of the first. This kind of interaction is the most prevalent and a typical example is the interaction between CFB and TCS. Subscriber B redirects an incoming call from A to C according to its CFB feature. But A happens to be on the screening list of C. Interaction happens, and the intention of CFB is destroyed by TSC service of C. The connection state of the intention of the CFB and the resulting connection of TCS are as shown in Figure 5. This behavior is similar to shared trigger interactions (STI) defined in[10] .

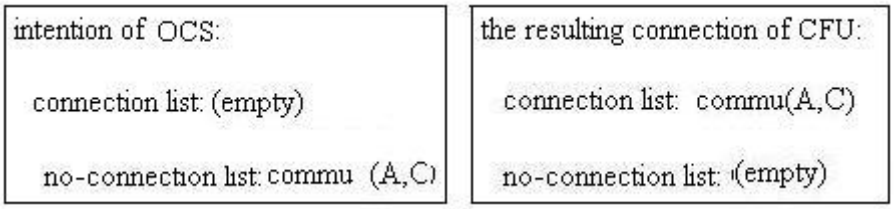

Figure 4. State representation of OCS and CFU

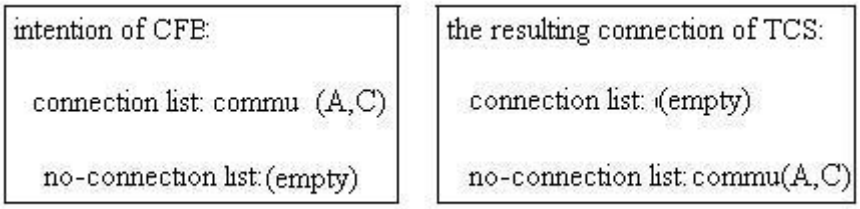

Figure 5. State representation of CFB and TCS

Category 2, Forwarding loops. If both features are forwarding features, the resulting connection of the second feature is identical to the originating connection and the resulting connection of the first feature is identical to the originating connection of the second feature we encounter a forwarding loop. [10] mentions these as a typical type.

In this case, the initial connection state needs to be compared to the resulting connection, and if identical a forwarding loop has been detected. A typical example is a user B forwarding all incoming calls to C, while C forwards incoming calls to B when she is busy. Figure 6 represents the call states.

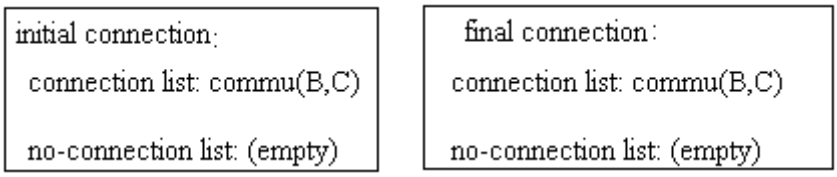

Figure 6. Example of forwarding loop

Category 3, Connection to an unrelated person. If the initial originator of the call finally receives a call request from a third party or if the initial destination of the call is finally connected to an unknown third party users will be astonished. This type of interaction has been called user intention violation in[11]. Combination of a CF service and a CR service easily leads to this kind of feature interaction. Suppose subscriber B forwards all the incoming call to C, while C has a CR service. When user A calls B, the call would be forwarded to C, but C is busy now, so C calls back later. An unexpected connection is built. It is shown in Figure 7.

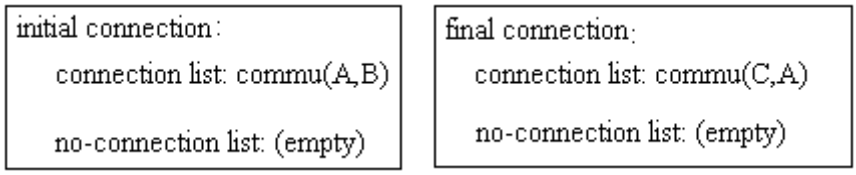

Figure 7. Example of Connection to an unrelated person

3.3 Summary of Interaction Analysis

Among the three categories, the intention of the first feature and the resulting connection are needed for interaction detection in the first case, while the last two need to retrieve the initial connection and the resulting connection. However, as in a typical call situation we don't know which interaction will be encountered a-priori sufficient data needs to be collected to ensure we can detect all situations. Hence, three connection states should be recorded and represented: the initial potential connection, the intention of the first feature and the resulting connection. The information of the initial connection and the resulting connection, such as From info and To info can retrieve directly, so only one message will be recorded in the connection list or the no-connection list. Since more than one intention may be described in a CPL script, multiple items may be recorded in the relating lists.

4. Applying our Approach to CPL Features in a SIP Environment

As mentioned before, a feature interaction manager forms the core of our method. The manager stores the initial potential connection, the first feature's intention, and resulting connection when related SIP messages are observed, and judges whether a feature interaction exists. We have so far implemented the manager on the feature server, so it can detect Single-Component feature interactions.

The feature intention and the potential connection are obtained in the way discussed in section 2.3. This method aims to realize on-line detection of CPL, so it also needs data from the CPL script to learn the feature's intention, hence this needs to be obtained from the CPL script, or from the data structure which stores data retrieved from the CPL script. But CPL script data on its own is insufficient. We also need real time data too to learn about some identities of the involved parties. For example, subscriber B has a CFB service, from the script we can only know where B wants to redirect the call when he/she is busy, but the originator of the call must be obtained from the actual incoming SIP message that triggered the feature. Hence, the feature intention of the first feature is learned from the CPL data and the related SIP messages. The originator of the potential call is the user in the From header, and the potential destination is the user specified in the CPL script in case of "proxy" or "redirect", or a treatment in case of "reject". The initial and resulting connection can be obtained from SIP messages without considering scripts. Usually the connection will be built between the user in the From header and the one in the request line (the first line of a INVITE message).

The above applies to conditions that the service is deployed on a SIP proxy server, or a user agent server. However, it can also be present on a redirect server. In that case, the server does not simply change the request line of the INVITE message and forward it, but sends a "302 Moved temporarily" or "301 Moved permanently" response with the new address in the Contact header. Then potential connection can then be obtained from the Contact header of the 301 or 302 message instead of the INVITE message.

A flag is used to mark the number of the features the call flow goes through. In our research, there are at most two features that will be triggered. The judgment is triggered by the second feature. It doesn't matter whether the first feature was actually triggered by the INVITE or not, because we use its intention to detect feature interaction.

We represent every connection state in the form that was introduced in Section 2 with allowed connections in the connection list and forbidden connections in the no-connection list. In case of the first category of interaction, comparison should be made between the first feature's intention and the resulting connection. If the same item appears in lists with different names, interaction happens. In the example illustrated in figure 4, commu(A,C) appears both at the first intention's no-communication list and the resulting connection's communication list. It indicates that the first feature intends to establish a forbidden connection between A and C, with that connection established finally. Clearly, the second feature changes the call opposed to the first one's intention. Feature interaction happens between OCS and CFU. This kind of interaction mainly occurs between proxy and reject actions. In case of the last two categories of interaction, comparison should be made between the initial potential connection and the resulting connection. Comparison should be made between the initial state's connection list and the resulting connection's connection list.

We will now describe the detection algorithm in detail:

Input: A duplicate of the initial INVITE message, information of the first CPL feature (i.e. the signaling operation and the specified location), and a duplicate of the newly generated SIP messages after the activation of the second feature.

Output: Notification of whether feature interaction occurred.

The feature interaction manager performs the following algorithm:

1. Retrieve the initial potential connection from the duplicate of the initial INVITE message and place it on the initial connection state's *connectlist*; retrieve the first feature's action, which includes the initial INVITE message, signaling operation (e.g. proxy, redirect, etc) and address information and store them into *connectlist* or *no-connectlist* according to the information of the feature action.

2. Check the flag to see if the second feature is triggered. If so, retrieve the resulting connection from the duplicate of the newly generated SIP message, put the item on the resulting connection state's *connectlist* and go to Step 3; else, delete the saved data and the feature interaction detection is finished (there were less than two features, so no interaction is detected).

3. Detect feature interaction by string matching.

   i. Get string from the resulting connection state's *connectlist*. Let us call this *S*.

   ii. Search for *S* in the first action's *no-connectlist*. If *S* is not found, go to step iii; if found, a feature interaction of the first category has been found. A notification will be given to the user.

   iii. Search for S in the initial connection's *connectlist*. If not found, go to step iv; if found, feature interaction of the second category has been found. A notification will be given to the user.

iv. Check the user part of the string to see if the resulting connection is built from a third party to the initial originator or from the initial destination to a third party. If so, feature interaction of the third category has been found and a notification will be given to the user; else, no feature interaction occurred and the feature interaction detection is completed.

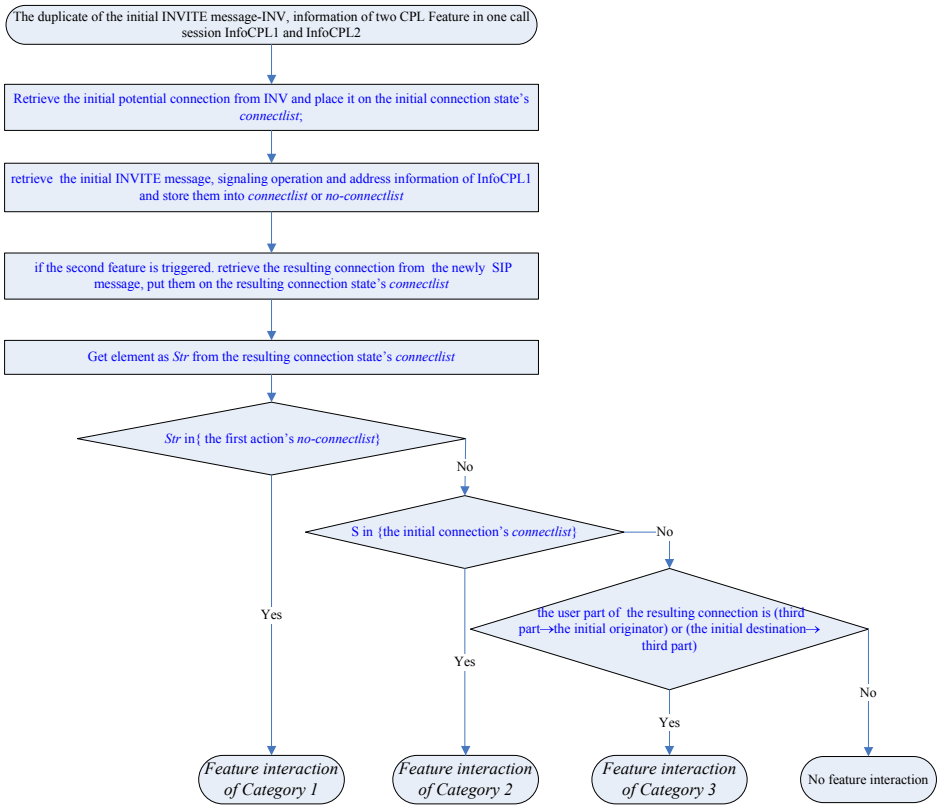

Figure 8. The algorithm of detecting feature interaction

5. Implementation

The approach has been implemented with the VOCAL system, which is developed by an open-source community vovida.org[12]. VOCAL takes SIP as its signaling protocol, and provides a SIP stack, SIP servers and a SIP user agent. The VOCAL architecture includes a special server called feature server to supply services programmed in CPL, thus providing the ideal platform for the validation of our methods. Conveniently, the platform includes 5 ready–made CPL features as well as managers to activate, deactivate and configure features through the provisioning GUI (a built-in interface). Users can deploy their own additional features by placing new CPL scripts in a designated directory and restarting a feature server's process to enable the new script.

The experimental setup includes a computer with Linux operating system with VOCAL deployed on it, and several other computers running x-lite, a free software telephone.

The goal of the experiment is to validate the application of the proposed method to a practical situation. We can detect feature interactions between the ready-made features and also with custom features.

5.1 Implementation of our Approach

The feature interaction manager is deployed overarching different feature servers, as shown in Figure 8. The detection procedure is as follows:

(1) The first feature server through which the call goes sends the initial INVITE message to the manager and the manager extracts the initial potential connection.

(2) Being centralized, the manager can also get data from the first related feature subscribed to by the initial originator or destination from the feature server. This information can include location sets and the signaling operators to get feature intentions. Also, we can obtain information whether the feature is triggered or not at this time.

(3) If a second involved feature is triggered in the call, new INVITE messages, and response messages (such as 301, 302, 403, 404, 486, 450 messages) generated by the feature server will also be send to the feature interaction manager. This allows to determine the resulting connection. If no features are involved any more, the feature server skips this call and applies its action to the next call.

(4) The manager checks the information stored to see if the call has gone through two features, if so detects feature interaction using the algorithm discussed earlier.

Note that, although features are executed by different feature servers, since the whole VOCAL system is deployed on one computer, feature interactions in this environment can be considered as Single-Component ones.

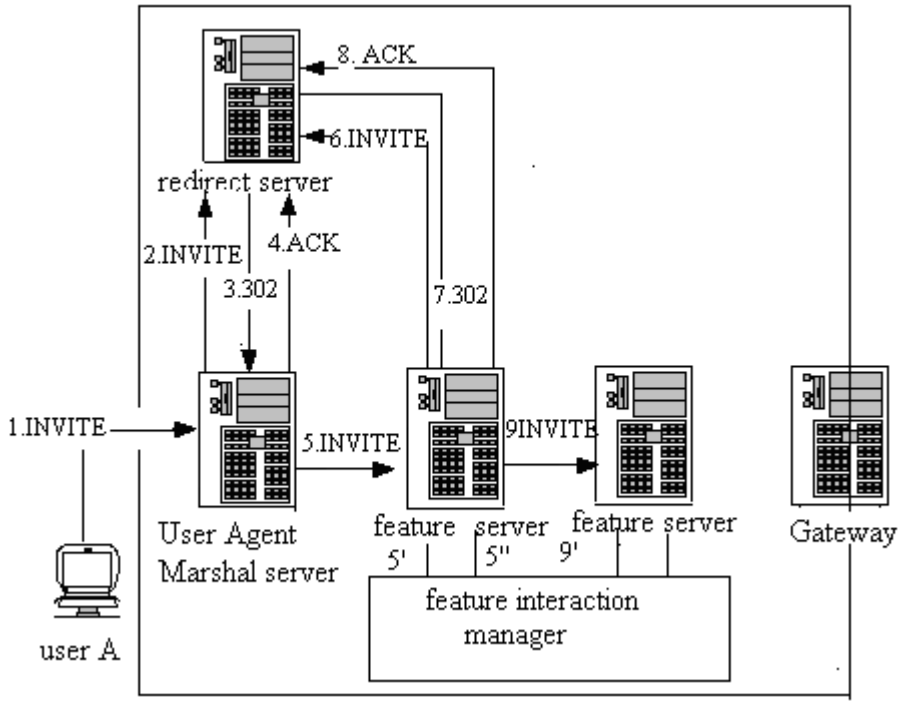

Figure 8. Location of the feature interaction manager

5.2 A Case Study

In the following the approach is applied to the interaction between Originating Call Screening and Call Forwarding Unconditionally.

Though the VOCAL system is deployed on one computer, it has two Marshal Servers which use different ports of the computer. The public address of the computer is telcom.upc.edu.cn. The public address of the user agent client is Alice@ telcom.upc.edu.cn:5060, and Carl@ telcom.upc.edu.cn:5065. User Alice subscribes to the OCS feature with Carl on the screening list and user Bob subscribes to CFU to forward all incoming calls to Carl. In the scenario, Alice attempts to invite Bob to a session. The relevant part of the INVITE message is as follows:

INVITE sip: bob@ telcom.upc.edu.cn:5065
From: sip: alice@ telcom.upc.edu.cn:5060; tag=3f5428
To: sip bob@ telcom.upc.edu.cn:5065

The INVITE message is captured by the feature interaction manager and the original connection state is stored with commu(alice, bob) on the connection list. Then the manager gets the OCS's information from the feature server and together with the INVITE message, the manager retrieves the feature's intention and adds commu(alice, carl) to the (currently empty) no-connection list. At this time OCS is not triggered. The

call set up continues, and a new INVITE message is generated according to Bob's CFU feature.

INVITE sip: carl@ telcom.upc.edu.cn:5065
From: sip: alice@ telcom.upc.edu.cn:5060; tag=3f5428
To: sip bob@ telcom.upc.edu.cn:5065

The manager captures this INVITE message and obtains the final connection. The item commu(alice, carl) appears on the final connection state's connection list. The manager compares the three connection states using its algorithm and detects a feature interaction between OCS and CFU.

5.3 Discussion about the Example

We choose OCS and CFU as our example, because it is quite a typical example in the area of telecommunication features and hence does not need much explaining. In this case, the OCS feature is not triggered by the INVITE message, and the feature interaction can still be detected, because its intention is used.

6. Related Work

There is a large body of FI work already in existence, and some approaches are targeted towards SIP and CPL.

Nakamura et al. [6] addressed eight kinds of possible semantic warnings in individual CPL scripts. Their method can detect semantic warning in a single CPL script directly according to their definition. For pairs of scripts, they present a combination operator to first compose the individual behaviours into one script on which they can use the semantic warnings for detection. Their approach is based on single script, and concentrates on semantic warnings – we assume that each script is free of semantic and compilation problems. Another significant difference is that their method is off-line while ours is on-line. Xu et al. [4] present very similar ideas, but they concentrate on logic inconsistence. They use the Simple Formal Specification Language (SFSL) to express the intentions of CPL scripts in logic. Their method can detect logic interaction both in single scripts as well as in pairs of script. Incompatibility of reject and proxy as well as forwarding loops can be detected by their method. However they cannot detect connections between unrelated parties and again their method is off-line.

Kolberg et al. [9] propose an on-line technique to detect and resolve feature interaction between distributed SIP call services. Their method is based on six rules, allowing them to detect connection looping, redirection and treatment and diversion and reversing similar to our method. In fact, we don't consider their rule 5 Treatment and Subsequent Missed call Handling as a kind of interaction. In their example, the originating connection is from A to B, and A has OCS service with B on the screening list, while B has CFB service. In this case, CFB will not be triggered, because the call is forbidden on the first server where OCS is deployed. In our opinion this should not be considered as an interaction, for the service OCS would exhibit similar behaviour in the absence of CFB. A more fundamental difference lies in the fact that their approach is based on an extension to SIP, while we do not require such an extension. Chentouf et al. [13, 14] proposes a solution for Multi-Component FI situations. A manager called

FIMA (Feature Interaction Management Agent) is introduced in their work. However, they just describe the framework, and do not consider detection rules. However, this work might be of interest to us in the future when considering extensions to Multi-Component FI.

7. Summary and Future Work

We have implemented an on-line method to detect Single-Component feature interactions, through comparing one feature's intention and the resulting connection, or the initial potential connection and the resulting connection using a feature manager. This detection approach is consistent with the INVITE message flow. This approach allows to detect interactions independent of whether the feature is actually triggered or not. In this way, interactions can be detected, and we do not need to check response messages any more.

Feature intentions can be detect without knowing detailed implementation information about the service. What we need is only some key data about the service, much of which is often configuration data (who is being screened, who subscribes the feature) and this information is being collected at runtime, so the approach can be applied in environments where detailed information of services is not available. More detailed evaluation will be given in the future work.

Multi-Component feature interaction is an important category in VoIP, because of the inherent distributed environment underlying VoIP telephony servers. Multi-Component feature interaction is left as future work. One option is to maintain a centralized feature manager which interacts with all feature servers, similar to Chentouf's work – however there are alternatives that involve communication between a number of decentralized feature managers.

References

[1] M. K. Muffy Calder, Evan H. Magill, Stephan Reiff-Marganiec, "Feature interaction: a critical review and considered forecast," *Computer Networks,* vol. 41, pp. 115-141, 2003/1/15 Jan., 2003.

[2] Y. Xu, L. Logrippo, and J. Sincennes, "Detecting feature interactions in CPL," *Journal of Network and Computer Applications,* vol. 30, pp. 775-799, April 2007 2007.

[3] J. Lennox and H. Schulzrinn, "CPL: A Language for User Control of Internet Telephony Services," 2002.

[4] Y. Xu, L. Logrippo, and J. Sincennes, "Detecting feature interactions in CPL," *Network and Computer Applications,* 2005.

[5] N. Gorse, L. Logrippo, and J. sincennes, "Detecting Feature Interaction in CPL," *Software&System Modeling,* vol. 5, pp. 121-134, 2005/11/16 2006.

[6] P. Leelaprute, M. Nakamura, K.-i. Matsumoto, and T. Kikuno, "Evaluating Semantic Warnings in VoIP Programmable Services with Open Source Environment," in *The Tenth Asia-Pacific Software Engineering Conference(APSEC'03)*, 2003, p. 10.

[7] E. J. Cameron, N. Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen, "A feature-interaction benchmark for IN and beyond," *IEEE Communication Magazine,* vol. 31, pp. 64-69, March, 1993 1993.

[8] S. Reiff-Marganiec, Turner K., "Feature interactions in policies," *Computer Networks,* vol. 45, pp. 569-584, 2004.

[9] R. E. F. N. J. NHsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *2nd IJCAI, Imperial College, London, England, September 1-3, 1971.,* 1971.

[10] Muffy Calder, Mario Kolberg, Evan Magill, Dave Marples, and S. Reiff-Marganiec, "Hybrid Solutions to the Feature Interaction Problem," in *The 7th International Workshop on Feature Interactions in Telecommunications and Software Systems*, Ottawa, Canada, 2003, pp. 295-312.

[11] S. Reiff-Marganiec, "Runtime Resolution of Feature Interactions in Evolving Telecommunications Systems," in *Department of Computing Science*. vol. Ph.D Glasgow, UK: University of Glasgow, 2002, p. 160.

[12] VOCAL, VOCAL: The Vovida Open Communication Application Library, *Available at http://www.vovida.org,*2007

[13] Z. C. Chentouf, S. Khoumsi, A, "Implementing online feature interaction detection in SIP environment: early results," *ICT 2003,* vol. 1, pp. 515-521, 2003.

[14] Z. Chentouf, S. Cherkaoui, and A. Khoumsi, "Experimenting with Feature Interaction Management in SIP Environment," *Journal of Telecommunication Systems,* vol. 24, pp. 251-274, 2003.