Detecting Feature Interactions between SIP Call Control Services

Mario Kolberg [a,1] and Evan H. Magill [a]
[a] *Department of Computing Science & Mathematics, University of Stirling, UK*

Abstract. The Session Initiation Protocol (SIP) is a strong contender as a call control protocol for Voice over IP (VoIP), and indeed commercial implementations are readily available off-the-shelf. SIP supports flexible service provisioning not only through third parties, but also end-users. Laboratory experience shows that as these services are interworking they are subject to the feature interaction problem. Feature interactions may considerably delay service deployment and hence are a threat to rapid service provisioning.

This paper investigates the feature interaction problem in SIP-based services and investigates the application of a pragmatic approach. This runtime approach does not require any detailed information about the services and hence can be applied in a competitive market. Furthermore, the approach is particularly strong in detecting interactions between distributed services - a key characteristic of SIP-based services.

Keywords. Feature Interactions, Runtime Approach, SIP, VoIP

1. Introduction

One of the main drivers for the success of SIP is the relatively easy provisioning of services. Third party service providers and even end users may provide services. Once fully tested and deployed, each service functions well on its own. However, as was discussed by Lennox and Schulzrinne [1], when SIP services interwork their combined behaviour may not be acceptable. This phenomenon already known from traditional telephony networks is known as feature interactions or service interactions [2].

1.1. Basic Terms

Within telecommunications the terms interworking and incompatibility have well understood meanings. Services must *interwork* to share a (communications) resource, for instance a session. The services may interwork *explicitly* through an exchange of information with each other, or *implicitly* through changing the session. In the second case, the services often have no knowledge that the other exists.

When services interwork to share communication resources, they are *compatible* if the joint behaviour of the resource is acceptable. However, if the joint behaviour is not

[1]Correspondence to: Mario Kolberg, Department of Computing Science & Mathematics, University of Stirling, Stirling FK9 4LA, United Kingdom; E-mail: mko@cs.stir.ac.uk

acceptable, i.e. the services are not compatible, the services are said to *interact*. Compatibility does *not* refer to coding errors, nor to the adherence of interfaces or protocols, but to the adequate behaviour of a resource under the joint control of interworking service.

Although services with comparable functionality are already known from traditional telephony network, there are some significant differences when deployed on SIP networks. SIP has some fundamental differences to traditional telephony networks and hence it is necessary to study these problems again in a SIP context.

Some authors distinguish the concepts of features and services. For this paper, the distinction between feature and service is not significant, what is crucial is the concept of *interaction*. The terms *service* and *feature* are used interchangeably.

Clearly, with the increased complexity and number of services, the problem gets worse. Neither manual inspection, nor simple testing, offer tractable solutions. More effective approaches addressing the special requirements of this domain are needed.

1.2. Basic Approaches

A substantial body of work [3,4] exists on dealing with feature interactions. Most approaches can be categorised as either off-line or on-line. Briefly, off-line approaches are applicable at design-time whereas on-line approaches are applied at run-time. The former being most useful at the early stages of the software lifecycle, the latter during testing and deployment [5].

Off-line approaches are often based on the application of formal methods, and as such require considerable information of each individual software increment. Increasingly, as the market becomes more competitive, this information may not be available. Also, as the number of services increases, the work in analysing pair-wise interactions increases with the square of the number of services. With a large number of services in an open market, this will quickly become untenable. However, off-line approaches still have a role to test services inside a single offering.

In contrast, on-line approaches carry out checks as required. Clearly there are computing resource issues, but the major issue is having sufficient information about the services available at runtime. A particular limitation with run-time approaches is the ability to detect interactions between services deployed on different components in the network. The approach presented in this paper is attempting to close this gap. A more detailed discussion on existing approaches can be found in [6] and [2].

2. SIP

2.1. Architecture and Components

Voice over IP uses a number of different protocols. SIP [7] is used for ’signalling’. SIP is concerned with user registration as well as session setup, modification and termination. Crucially, SIP does not deal with the media exchange as such. Other protocols are used in combination with SIP to allow for different media to be exchanged. Indeed SIP can be used to establish non-telephony sessions.

Within a SIP network there are two basic types of devices, end devices (user agents) and servers. User agents are the devices used by end users to place or receive calls. These may be SIP phones, or so called soft phones, which are software implementations to be run on a PC. Note that user agents do not necessarily interface directly with a user, an answering machine is also a user agent. User agents are distinguished according to their role in a call: the user agent client places the call, and the user agent server receives the call. User agents initiate and respond to signalling and send and receive media. User agents are aware of the call state. Unlike traditional telephony, user agents may provide a number of services, such as Call Waiting, Call Forwarding, or Call Screening.

Servers handle the application level control and routing of SIP messages. There are three different kinds of servers defined in SIP: Register, Redirect and Proxy servers. If a user is to be invited to join a session (call), there is the question of where the invitation should be sent to as users may be located at different IP addresses. Users are addressed by email-like addresses, e.g. sip:mko@cs.stir.ac.uk. This is a public address. However, at present, the user may in fact be located at a computer with the name d25.cs.stir.ac.uk and be logged on as user mk0123. The SIP address for this location would be sip:mk0123@d25.cs.stir.ac.uk. To link the two addresses users need to register with a register server. Register servers work very closely with redirect and proxy servers.

Invitations are sent from the user agent client via a number of redirect or proxy servers to the user agent server. If a redirect server receives an invitation for a user, it checks with the database of the local register server and returns the address where it believes the user to be invited can be found. If the user is not actually located at that address, more information on the user’s location may be available from that address.

Proxy servers are similar to redirect servers in that they help to find the location of a user. However, a proxy server does not return the found address but forwards the invitation on to that address. In the path an invitation is sent from the user agent client to the user agent server there may be number of redirect *and* proxy servers.

Both proxy and redirect servers can host and execute call control services in that they can direct, block, or alter call signalling messages. Consequently, SIP offers the potential for a truly distributed service provisioning. Services may be deployed on user agents and redirect and proxy servers. SIP will allow a degree of programmability which is unknown in the PSTN.

2.2. Consequences for Feature Interactions

Services working on the same call may be deployed in a number of different locations, which are controlled by separate organisations. These organisations may not be aware of each other or competing with each other. Thus they are not inclined to share detailed information on their services to avoid interactions. Also because of the large degree of possible programmability, even end users may design and deploy their own call control services, either on their user agent or by uploading them to the local proxy server. Consequently, SIP uses a heavily distributed architecture with services possibly being deployed on every component. Any approach to feature interactions for SIP needs to take this into account.

The fact that media packets travel end-to-end, without being interceptable by intermediate servers means that some services can no longer be implemented transparently. For instance, "pipe-bending" services, such as forwarding a call, cannot be performed without informing the other party of the new address where they should send the media packets to [1].

Further, the increased numbers of possible addresses can also complicate some services. In the traditional telephony network a phone number can be used to identify a party for call screening. In SIP this is much harder to achieve.

The next section provides details of an approach which is applicable to SIP. Section 4 discusses the application of the approach to SIP.

3. The Applied Approach

The algorithm is based on the pragmatic approach presented in [8]. Here it is adapted to operate in a SIP environment. The approach concentrates on the establishment of connections and does not require detailed information about the involved services, but operates at a higher level.

Thus it can be applied in a competitive business environment where no detailed technical information will be available. And secondly, the applicability of the approach is independent of the network architecture. The approach has already been employed in a PSTN setting [2] and also as a filtering approach [8].

3.1. Overview

The behaviour of a service is described in two parts: the triggering party and a connection type. The latter consists of two parts: the original connection to be set up before the service is activated and the connection set up after the service has been triggered.

$$\boxed{\text{TP.: B; (A, B)} \rightarrow \text{(A, C)}}$$

Figure 1. Description of Call Forwarding Unconditional

An example should illustrate this. Call Forwarding Unconditional (CFU), which redirects all incoming calls to a predefined third user, can be described as shown in Fig. 1. Assume party A is the originator, B the terminator, and C the party where the call is redirected to. The behaviour has two parts, separated by a semicolon. In the first part, the notation TP.: X indicates that X is the triggering party, in this case it is B because CFU is triggered at the terminating end of a call. In the second part, notation (X, Y) → (U, V) indicates the connection type. (X, Y) is called the original connection and (U, V) is the connection after activating the service. For each pair (A, B) A is the source and B the destination. The call starts with A attempting to connect to B. However, because of CFU, A is connected to C instead. So the connection type is (A, B) → (A, C).

Treatments are an important aspect of this approach. Treatments are announcements or tones triggered by the network to handle certain conditions during a call, for example when a call is screened or blocked.

Clearly, this way of describing services abstracts from a considerable amount of service behaviour. The presented results show that omitting these details is not an issue.

3.2. Interaction Analysis

Interaction cases are found by analysing *pairs* of services. Two service descriptions are compared according to five rules. If a service pair fulfills any of the five rules, then the pair is said to interact. In the following, each of the rules is considered in turn.

Rule 1 – Single User - Dual Service Control: If both services have the same triggering party and either the original connections or the resulting connections are identical, the pair interacts. Examples are given in Figure 2. The shaded portions indicate the key parts of the descriptions.

CFB: TP.: B; (A, B) → (A, C)
CW: TP.: B; (A, B) → (A, B)
(a)

AR: TP.: A; (B, A) → (A, B)
HL: TP.: A; (A, B) → (A, B)
(b)

Figure 2. (a) Call Forwarding Busy & Call Waiting and (b) Automatic Ringback & Hotline

Rule 2 – Connection Looping: If the original connection of the first service is identical to the resulting connection of the second service, and the original connection of the second service is identical to the resulting connection of the first service then a connection loop occurs. Further, the triggering parties need to be different. As both services are trying to divert the connection in a circular way, a loop occurs (ref. Figure 3). Again, the shaded portions indicate the identical connections.

CFB: TP.: B; (A, B) → (A, C)
CFU: TP.: C; (A, C) → (A, B)

Figure 3. Call Forwarding Busy and Call Forwarding Unconditional revisited

Rule 3 – Redirection and Treatment: This type of interaction is detected if the resulting connection of the first service matches the original connection of the second service and the second service connects to a treatment. Further, one of two conditions need to be met. Firstly, the originating party of the original connection and the originating party of the resulting connection of the first service need to be identical and the terminating parties be different (Fig. 4(a)). Or secondly, the originating party of the original connection needs to match the terminating party of the resulting connection and the terminating party of the original connection is the originating party of the resulting connection (Fig. 4(b)).
In other words, one service establishes a connection by either forwarding (not to a treatment) or reversing a connection. The resulting connection is the original one of the second service, which redirects the call to a treatment. This scenario is a potential problem as the connection which is set-up by the redirection service is prevented by the treatment service.

Rule 4 – Diversion and Reversing: This rule specifies that an interaction is detected if the resulting connection of the first service matches the original connection of the second service. Furthermore, for one of the two services, the originating party of

CFB: TP.: C; (A, C) → (A, B)
OCS: TP.: A; (A, B) → (A, Treat) (a)

AR: TP.: B; (A, B) → (B, A)
OCS: TP.: B; (B, A) → (B, Treat) (b)

Figure 4. (a) Call Forwarding Busy & Originating Call Screening and (b) Automatic Ringback & Originating Call Screening

the original connection is identical with the originating party of the resulting connection and the terminating parties of the two connections are different. For the other service, the originating party of the original connection needs to be the terminating party of the resulting connection and the terminating party of the original connection needs to be the originating party of the resulting connection.
Here one service forwards a call and the other reverses the call. This may happen in either order, i.e. one service forwards a call which is subsequently reversed to the originator by the other service. In this case the originator of the original connection will receive a reversed call from someone they never rung.

CFB: TP.: C; (A, C) → (A, B)
AR: TP.: B; (A, B) → (B, A) (a)

AR: TP.: B; (A, B) → (B, A)
CFB: TP.: A; (B, A) → (B, C) (b)

Figure 5. (a) Call Forwarding Busy & Automatic Ringback and (b) Automatic Ringback & Call Forwarding Busy

Alternatively, a reversed call is forwarded. In this case, a service reverses a call which is subsequently forwarded by the other service to a third party. Consequently, the returned call is reaching a person which never placed a call in the first place. Figure 5 contains two example interaction scenarios.

Rule 5 – Treatment and Subsequent Missed Call Handling: This type of interaction occurs if the original connections for both services are identical. For one service the triggering party must also be the originating party of original connection and the terminating party of the resulting connection is treatment. Finally, the triggering parties of both services need to be different.
This type of interaction is concerned with call control services which are prevented from functioning by another service which connects the call to a treatment. An example of this type of interaction is given in Figure 6.

CFB: TP.: B; (A, B) → (A, C)
OCS: TP.: A; (A, B) → (A, Treat)

Figure 6. Call Forwarding on Busy and Originating Call Screening revisited.

Finally, it is important to note that the descriptions are on a per service base, not per services pair. It is only the algorithm which combines two service descriptions to pairs. This greatly reduces the complexity, because when new services are introduced no existing descriptions need to be changed.

4. Using the Approach in SIP

4.1. SIP Services

Fundamentally, services can be provided in two ways: SIP CPL [9] and SIP CGI [10]. SIP CPL is a XML-based language and is restricted in its functionality, but safe. SIP CGI offers full access to SIP messages and also the use of external databases which are important for a number of services, such as forwarding and screening. Thus the discussion in this paper is targeted at SIP CGI services.

4.2. Application of the Approach in SIP

SIP is a distributed protocol, with services being located at various locations through which a SIP message travels. Hence, the approach and especially the application of the algorithm has to be distributed as well.

The basic idea is that each service which gets activated includes its Triggering Party and Connection Type into the message. If there is already one or more entries in the message, these are checked against the description of the current service. Thus the algorithm is executed wherever necessary and a central feature manager is not required. This is possible as SIP is an extensible protocol. Additional headers carrying additional information may be defined and included with messages. The newly defined header to carry the required information for this approach is called **Contype**.

For this approach, each service needs to be surrounded by a cocoon. The cocoon contains the connection type for the service and the logic for the interaction algorithm.

When a message arrives at a server (or user agent), the message gets passed to the deployed services. As the services are surrounded by cocoons, the message is actually sent to the cocoon, but from there it is sent immediately to the service proper. At this point, the cocoon does not check or alter the message.

Once the service proper is finished with execution the potentially altered message is passed back to the cocoon. With the message, the service sends an indication to the cocoon, whether the service actually was triggered by the message, i.e. it altered the message (e.g. CFU), or was armed to execute at some event in the future (e.g. AR).

This indication is important as the services may only get triggered by some messages, e.g. some calls are not forwarded or not screened. This often depends on service specific data, such as screening lists. Services which have not been triggered cannot cause any interactions. Hence the following algorithm does not need to be executed for such services. If the service was triggered by the message, the cocoon checks the message for a header called **Contype**. These headers contain the descriptions of services which have already been active in that transaction.

If such a header is not found, no other service has previously been active and hence a service interaction cannot have occurred. In this case, the cocoon inserts a Contype header into the message which contains the description of the related service. For instance, for the Call Forwarding Unconditional service discussed in Section 4.1 the header is depicted in Figure 7.

The header contains a number of fields: the ID field shows which service is represented by the header. Currently this is a simple string, but to avoid duplicate names unique identifiers can be used in future implementations. The TP field contains the trig-

ConType: ID=CFU; TP=sip:bob@d254203.cs.stir.ac.uk;
OrigFrom=chris@discus.cs.stir.ac.uk; OrigTo=bob@d254203.cs.stir.ac.uk;
FinalFrom=chris@discus.cs.stir.acuk; FinalTo=alice@d254203.cs.stir.ac.uk

Figure 7. SIP Header for CFU

gering party, and the remaining four fields correspond to the four fields used to construct the connection type.

If a Contype header is found in the message, the data from that header is extracted and together with the description of the local service fed into the service interaction algorithm. If no service interaction is detected, and no further Contype header is found in the message, the Contype header for the current service is inserted into the message and the message is sent on to its destination.

If, however, a service interaction is detected, it needs to be resolved also at run-time. Currently, a basic solution has been implemented: if an interaction occurs, the outcome of the second service is discarded, and the call proceeds as if that service was not activated at all. This is a basic resolution approach, and clearly more elaborated approaches, such as presented in [11,12] could be used. Also policies may be involved finding the best solution. Resolution approaches may be tuned to achieve various goals, such as finding the solution with the most executed services, or executing all services subscribed to by the party who pays for the call, or, in a private environment the services belonging to the organisation may have priority over services of the individual. In a way all these approaches implement a priority system. Our approach gives the highest priority to the service executed first.

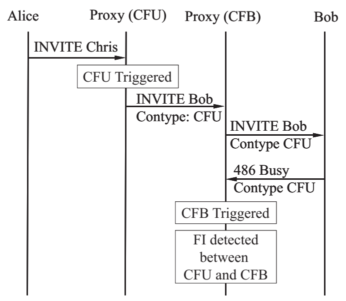


Figure 8. Message flow with CFU and CFB

However, not all services involved in a session setup are triggered by the INVITE message sent by the user agent client. For instance, with services triggered on busy responses there will be INVITE messages and also busy response messages involved in the call setup. For such cases, response messages also need to contain the Contype headers which were added to the corresponding INVITE. Thus if a user agent server receives an INVITE request with Contype headers, the Contype headers need to be copied into the response message. This way, cocoons for services triggered by response messages are also aware of services that previously have been operating on the INVITE. Clearly, if a service which is triggered by a response message issues a new INVITE request, then the cocoon needs to copy any Contype headers from the response to the new INVITE message. The approach of copying headers from the request to response message is a common approach taken in SIP. For instance, the standard VIA header is also used in this way. The interaction between CFU and CFB is shown as an example of such a message flow in Figure 8.

In the example, Alice tries to invite Chris, however, at the local proxy server the INVITE gets forwarded by a CFU service to Bob. A Contype header is attached to the INVITE message and set on. The message then passes through the second proxy and is sent to Bob's user agent. However, as Bob is busy, a 486 Busy response is returned. This response also contains the CFU Contype header. At the proxy, CFB is triggered by the 486 Busy message. The CFB service forwards all calls to Chris if Bob is busy. Thus the cocoon checks for an interaction and using the Contype header of the 486 response message, discovers an interaction and CFB is disregarded, i.e. the 486 Busy message is forwarded back to Alice. The descriptions of the two services is shown in Figure 9.

CFU: TP.: Chris; (Alice, Chris) → (Alice, Bob)
CFB: TP.: Bob; (Alice, Bob) → (Alice, Chris)

Figure 9. Call Forwarding Loop

Implementing the approach in SIP extends SIP with an additional header, Contype. This extension is in line with the SIP standard and follows SIP conventions. Clearly, in order for the approach to work, implementations of SIP components, especially proxy servers and user agents, need to be aware of the new header to make use of the information provided. However, if a message with a Contype header passes through a SIP component which does not support the header, it simply ignores it. This is one of the fundamental principles of SIP to allow for extensions. Thus, a component which does not support this extension does not fail to work properly, and does not prevent other component which support the header from using the information.

5. Experimentation

The approach has been implemented in a SIP testbed. SER [13] was chosen as proxy server, and a Pingtel SIP phone [14] and kphone [15] were used as user agents.

Currently, it is difficult to get access to implementations which support the provisioning of services generally, and even more so implementations which support the CGI interface. However, SER offers a proprietary interface for providing services. This interface provides access to full SIP messages and in principle this is the same as a CGI interface would offer. Thus the implementation discussed here is using the proprietary interface offered by SER and not the CGI interface.

The approach was tried with a number of service combinations. Unfortunately, there are no call control services available off-the-shelf. Thus simple service prototypes were developed to carry out the experimentation.

In the next two sections two example interactions are described and it is demonstrated how the approach can deal with them. Section 6 presents a summary of all scenarios tried and a discussion on the results.

5.1. A Simple Example

In the following the approach is applied to the interaction between Call Forwarding and Terminating Call Screening. An overview is depicted in Figure 10.

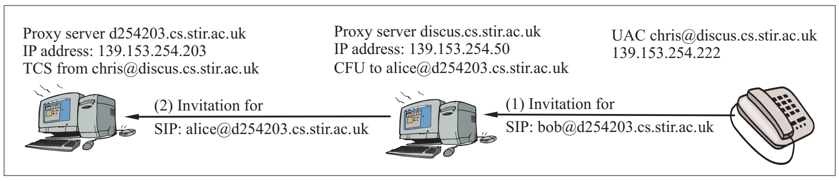

Proxy server d254203.cs.stir.ac.uk
IP address: 139.153.254.203
TCS from chris@discus.cs.stir.ac.uk
Proxy server discus.cs.stir.ac.uk
IP address: 139.153.254.50
CFU to alice@d254203.cs.stir.ac.uk
UAC chris@discus.cs.stir.ac.uk
139.153.254.222
(2) Invitation for
SIP: alice@d254203.cs.stir.ac.uk
(1) Invitation for
SIP: bob@d254203.cs.stir.ac.uk

Figure 10. Applying the Algorithm to the Interaction between CFU and TCS.

There are two proxy servers involved: discus.cs.stir.ac.uk and d254203.cs.stir.ac.uk. The public address of the user agent client (Pingtel phone) is chris@discus.cs.stir.ac.uk and is thus associated with the first proxy server. In the scenario, Chris attempts to invite Bob to a session, however, due to a forwarding service on the first proxy server, the invitation is redirected to Alice at the second proxy server. Alice has a terminating call screening service deployed on the second proxy server with Chris on the screening list.

Initially, the user agent client sends a INVITE message to invite to the first proxy server (payload and some unrelated headers have been omitted).

```
INVITE sip:bob@d254203.cs.stir.ac.uk SIP/2.0
From: sip:chris@discus.cs.stir.ac.uk;tag=1c18932
To: sip:bob@d254203.cs.stir.ac.uk
Contact: <sip:chris@139.153.254.222>
Via: SIP/2.0/UDP 139.153.254.222
```

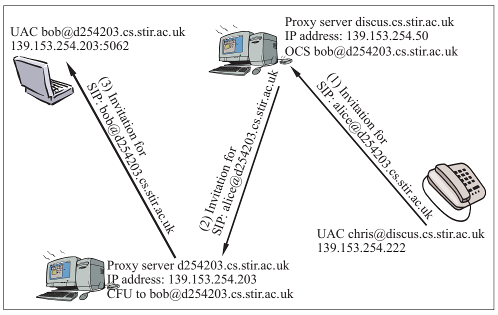
At this proxy server, the CFU service is invoked. This changes the message in a way that it is sent to Alice rather than Bob. Furthermore, as the service changed the message, the cocoon, inserts a Contype header into the INVITE request (payload and unrelated headers have been omitted).

```
INVITE sip:alice@d254203.cs.stir.ac.uk SIP/2.0
From: sip:chris@discus.cs.stir.ac.uk;tag=1c18932
To: sip:bob@d254203.cs.stir.ac.uk
Contact: <sip:chris@139.153.254.222>
Via: SIP/2.0/UDP 139.153.254.50;branch=z9hG4bK6d0d.a5da393.0
Via: SIP/2.0/UDP 139.153.254.222
ConType: ID=CFU; TP=sip:bob@d254203.cs.stir.ac.uk;
  OrigFrom=chris@discus.cs.stir.ac.uk;
  OrigTo=bob@d254203.cs.stir.ac.uk;
  FinalFrom=chris@discus.cs.stir.ac.uk;
  FinalTo=alice@d254203.cs.stir.ac.uk
```

At the second proxy server, the TCS service is called. After execution, the service notifies the cocoon that it was triggered and the cocoon checks the INVITE message for an existing Contype header. The message contains the CFU Contype header and thus the cocoon applies the service interaction algorithm to the CFU and TCS descriptions. As a result an interaction was detected by rule 3. Thus the action of the TCS service is discarded and the INVITE is forwarded to Alice. The approach has successfully handled the interaction.

5.2. A more complex Example

This section provides an example where the second service is triggered by a response message. The example used is the interaction between Originating Call Screening and Call Forwarding. Figure 11 shows the setup.



UAC bob@d254203.cs.stir.ac.uk
139.153.254.203:5062
Proxy server discus.cs.stir.ac.uk
IP address: 139.153.254.50
OCS bob@d254203.cs.stir.ac.uk
(3) Invitation for SIP: bob@d254203.cs.stir.ac.uk
(2) Invitation for SIP: alice@d254203.cs.stir.ac.uk
(1) Invitation for SIP: alice@d254203.cs.stir.ac.uk
UAC chris@discus.cs.stir.ac.uk
139.153.254.222
Proxy server d254203.cs.stir.ac.uk
IP address: 139.153.254.203
CFU to bob@d254203.cs.stir.ac.uk

Figure 11. Service Interaction between OCS and CFU.

When considering service interactions, one issue which is often discussed with Originating Call Screening is its actual purpose. That is, should it only prevent Chris from *dialing* Bob's number (perhaps because of additional costs being involved), or is it intended that Chris is not *connected* to Bob. If the aim is the former, then checking INVITE messages is sufficient and no interaction between the two services exists. In this paper, this service is then referred to as Originating Dial Screening (ODS). However, if the goal is the latter, 200 OK responses sent in reply to INVITE requests are checked by the service. This functionality is assumed in this section.

In the scenario, OCS is called on the first proxy server with no calls to Bob being allowed. OCS checks the destination of the INVITE request (address in the first line of the message) and since the request is directed towards Alice, the request is not screened. The cocoon is notified that OCS did not take any actions and hence the cocoon is not including a Contype header for the OCS service. Thus the INVITE message is sent on unchanged.

At the second proxy, the call forwarding service is called and it redirects the INVITE towards Bob. The cocoon inserts a Contype header for the CFU service. This INVITE message is then delivered to Bob's user agent server which responds with a *200 OK* message signalling that he accepts the invitation. This 200 OK response message also contains the CFU Contype header from the INVITE request. The 200 OK message is then sent back to the first proxy where the 200 OK response message triggers the OCS service again.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 139.153.254.50;branch=z9hG4bKafde.6e4854d4.0
Via: SIP/2.0/UDP 139.153.254.222
ConType: ID=CFU; TP=sip:alice@d254203.cs.stir.ac.uk;
  OrigFrom=chris@discus.cs.stir.ac.uk;
  OrigTo=alice@d254203.cs.stir.ac.uk;
  FinalFrom=chris@discus.cs.stir.ac.uk;
  FinalTo=bob@d254203.cs.stir.ac.uk
Forwarded-To: <sip:bob@d254203.cs.stir.ac.uk>
From: <sip:chris@discus.cs.stir.ac.uk>;tag=1c8060
To: <sip:alice@d254203.cs.stir.ac.uk>;tag=1A8019B3
Contact: "root" <sip:root@139.153.254.203:5062;transport=udp>
Record-Route: <sip:alice@139.153.254.203;ftag=1c8060;lr=lr>,
<sip:alice@139.153.254.50;ftag=1c8060;lr=lr>
```

However, there is an issue with the response sent by Bob as it does not necessarily reveal that the response is sent from Bob and not from Alice. The To header in the request and response is not changed by the forwarding service. In the SIP standard, the To header is defined as the address of the invited user the session setup started off with. Indeed, if the To header is changed by the forwarding service, the user agent client does not recognise related responses.

In SIP, the party the invitation was finally delivered to is identified by the Contact header. However, from the address given in this header it is not possible to derive who the other party is. The address in the contact header reflects the system username at the address given, not the SIP username. The SIP username and the system username do not relate to each other. For instance, in the example above, the contact header contains the address root@139.153.254.203:5062.

The second option of finding out the SIP name of the user agent server is checking the Record-Route header. Proxy servers put the SIP address in the Record-Route which they have been dealing with. In the above response the header only shows entries for Alice. This is because both proxy servers received an INVITE for Alice. However, if there was a third proxy server (IP address 139.153.254.23) in the chain, the Record-Route would look like:

```
Record-Route: <sip:alice@139.153.254.203;ftag=1c17644;lr=lr>,
<sip:alice@139.153.254.50;ftag=1c17644;lr=lr>
<sip:bob@139.153.254.23;ftag=1c17644;lr=lr>
```

Thus if the forwarding happens on any proxy server except the last in the chain, the Record-Route header can be used by the OCS service. However, because of the limitation this cannot be used as a general solution.

There does not appear to be any header defined in SIP which contains the SIP address of the invited party. However, for the screening service to work, this information is required. To overcome this issue another header was defined which contains the SIP address of the user agent server. The header is called **Forwarded-To** and is inserted into a message whenever a service redirects an INVITE request. As discussed above for the Contype header, the Forwarded-To header is also copied from the INVITE request to responses generated by the user agent server. The response shown above contains the Forwarded-To header.

When the OCS service checks the reply, it will find the Forwarded-To header with Bob's address. Because calls to Bob are not allowed, the cocoon is notified that OCS was active on the message. The cocoon will apply the service interaction algorithm to the data in the Contype header for the CFU service and the data for the OCS service and an interaction will be detected by rule 3. As OCS is the latest service to execute, its actions will be disregarded, i.e. the call will be setup in this example.

6. Case Study

6.1. Selected Services

For the case study nine common services were selected. These are Call Forwarding Unconditional, Call Forwarding Busy, Originating Call Screening, Terminating Call Screening, Voice Mail System, Automatic Ringback, Do Not Disturb, Hotline, and Group Ringing.

Following the modelling approach from Section 3.1 the connection type for each of the services was developed. Table 12 contains the specifications of the services. As can be seen from the table, some services which are actually quite different are modelled in a rather similar way. For instance, the services OCS and TCS differ only in their triggering party.

Service	**Triggering Party**	**Connection Type**
CFU	B	(A, B) $\rightarrow$ (A, C)
CFB	B	(A, B) $\rightarrow$ (A, C)
OCS	A	(A, B) $\rightarrow$ (A, Treat)
TCS	B	(A, B) $\rightarrow$ (A, Treat)
VMS	B	(A, B) $\rightarrow$ (A, Treat)
AR	B	(A, B) $\rightarrow$ (B, A)
DND	B	(A, B) $\rightarrow$ (A, Treat)
GR	B	(A, B) $\rightarrow$ (A, C)
HL	A	(A, B) $\rightarrow$ (A, B)

Figure 12. Specifications of the case study services

Furthermore, some rather different services have identical descriptions. For instance, TCS, DND and VMS are very different from another. However, even though the three services are quite different, they have some commonalities - these are captured by the notation of the approach. For instance, with all three services it is not the intended party who answers the call, but a network treatment. Thus at the chosen level of abstraction, the services exhibit the same interaction. For instance, CFU will interact with all three services because a forwarded call is not answered by the intended party, but rather by a network treatment. Clearly, at a lower level of abstraction, perhaps when considering network messages the interactions may be different.

Table 13 provides details of the detected interactions. Ticks show a successfully handled interaction, an ’x’ symbolises an interaction which is not detected by the approach (see below). Double entries in the table refer to multiple call scenarios between two services exhibiting interactions.

	CFU	CFB	OCS	TCS	VMS	AR	DND	HL	GR
CFU	√ √	√	√	√ x	√ x	√ √	√ x		√ √
CFB		√ √	√	√	√	√ √	√		√
OCS						√		√	√ √
TCS				√	√ x	√	√ x	√	√ x
VMS					√ x	√	√ x	√	√ x
AR						√	√		√
DND								√	√ x
HL									
GR									√ √

Figure 13. Results of SIP case study.

All the ’x’-entries in the table are referring to a single issue with the approach. This only concerns single component interactions, i.e. both services are deployed on the same proxy where the services are triggered by an INVITE request, but one service drops the INVITE and generates a response message (e.g. Terminating Call Screening). In this case an interaction can only be detected if the service dropping the INVITE is triggered second. This is because the cocoon of the first service needs to include the Contype header in the INVITE and the cocoon of the second service needs to see that header and execute the algorithm. However, if the service dropping the INVITE request is triggered first the second service will not be triggered at all and hence the cocoon stays inactive. This also applies to scenarios where both services drop the triggering message.

However, it could be argued that interactions between services deployed on the same server (or UA) are in fact not interactions in SIP. This is because deployed services are usually configured in a list, i.e. the first service is executed first followed by the next service and so on. It could be argued that this list represents implicit priorities. That is the order of the services in the list represent user preferences. Thus if one service prevents subsequent services from executing, it is likely to be in the interest of the user.

In SIP, interactions between two services may occur in different call scenarios. For instance, the interaction between HL and OCS can be simulated in two different scenarios. Firstly, the arguably more traditional setup where both services are deployed on the same location. However, the interaction can also successfully be verified when HL and OCS are deployed on different locations, i.e. Hotline on the user agent client and OCS on the local proxy server. The reason is that the approach works with public SIP addresses. The public SIP address of a user is identical regardless of whether the user agent or the local proxy are considered.

SIP offers some differences to PSTN services and poses some issues. The most important one is the possibility of identifying a single user by a number of different SIP addresses. This leads to difficulties in identifying a party, e.g. for screening purposes.

Related to this is the issue of identifying a party from a response message. This was rather surprising. The use of the To header appears redundant, as messages belonging to a session can also be identified by the Call-Id header. The introduction of the Forward-To header solved this problem. Arguing whether the usage of the SIP header is ideal as it is or even changing their meaning is beyond the scope of this paper.

Another issue is that in SIP services which handle the busy condition will be triggered differently than in the PSTN. Assuming that the services are deployed on a proxy server, they are not triggered by the INVITE message, but by the 486 Busy response sent by the user agent server. Due to this fact a number of Single Component interactions known from the PSTN do not exists in SIP. This applies to interactions which involved a service which handles a busy condition and a other service which is triggered regardless of the party being busy. The message flow for the classic example between TCS and CFB is depicted in Figure 14.

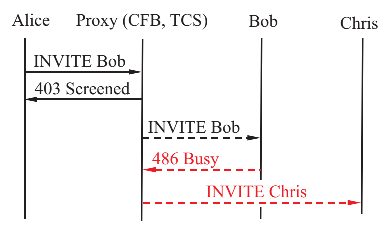


Figure 14. SIP Call Scenario between CFB and TCS

In this scenario Alice tries to call Bob. Bob has a CFB service which redirects all calls while he is busy to Chris. Furthermore, Bob has a TCS service which screens all calls from Alice. If Alice calls Bob while he is busy the call gets screened and there is no conflict with the CFB service. This is because the TCS service is triggered by the INVITE request. Thus the CFB service which waits for a 486 Busy response is never triggered. The CFB messages which are not sent because of TCS are shown by dashed lines. Thus in SIP, there is no interaction in this scenario.

7. Summary and Further Work

This paper has discussed the feature interaction problem between SIP call control services. A pragmatic approach has been implemented. The approach does not require detailed service knowledge. This is essential in an open and competitive market environment as expected for SIP services.

As was shown in the experimentation, the approach can easily be implemented on a SIP platform. The approach only requires service information at a very abstract level, which is readily available.

The approach requires extensions to the SIP protocol. However, the extensions are in line with the rules for SIP extensions as defined by the SIP standard. For the approach to work fully, all components involved in a session setup need to support these extensions. However, if some components do not support the extensions, sessions can still be setup but with limited feature interaction handling.

The particular strength of the approach is the detection of interactions between services deployed on *different* SIP components. As often there will be a number of SIP components involved in a session setup, distributed services will be common practise. As shown in the experiments, the presented approach is able to handle all these scenarios.

Clearly, the presented approach has some weaknesses detecting some Single Component interactions. However, as discussed above, single component interactions may not represent undesired behaviour as in PSTN. This is because the services are configured in a list which may represent users preferences (most important service first).

Once an interaction is detected it also needs to be resolved. In this paper a resolution approach based on the order of invocation has been adopted. In the event of an interaction, the service invoked last will be disregarded. This approach incurs minimal overhead and in principal constitutes a priority based approach. More elaborated priority schemes and also the use of policies to help select the service whose actions are disregarded, can be adopted in the future.

In summary, the presented approach closes a major gap as it allows to handle interactions between services deployed on different components at runtime. This will be necessary if distributed call control architectures, such as SIP, are to be successful.

References

[1] J. Lennox and H. Schulzrinne. Feature interaction in internet telephony. In *[3]*, pages 38–50, May 2000.

[2] M. Calder, M. Kolberg, E.H. Magill, and S. Reiff-Marganiec. Hybrid solutions to the feature interaction probelm. In *[4]*, pages 295–312, June 2003.

[3] M. Calder and E. Magill, editors. *Feature Interactions in Telecommunications and Software Systems VI*. IOS Press (Amsterdam), May 2000.

[4] D. Amyot and L. Logrippo, editors. *Feature Interactions in Telecommunications and Software Systems VII*. IOS Press (Amsterdam), June 2003.

[5] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature interaction: A critical review and considered forecast. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 41(1):115–141, 2003.

[6] D. O. Keck and P. J. Kuehn. The feature and service interaction problem in telecommunications systems: A survey. *IEEE Transactions on Software Engineering*, 24(10):779–796, October 1998. IEEE.

[7] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session initiation protocol. *Request for Comments (Standards Track) 3261*, 2002. Internet Engineering Task Force.

[8] M. Kolberg and E. H. Magill. A pragmatic approach to service interaction filtering between call control services. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 38(5):591–602, 2002.

[9] J. Lennox, X. Wu, and H. Schulzrinne. *Call Processing Language (CPL): A Language for User Control of Internet Telephony Services, RFC 3880*. Internet Engineering Task Force, 2004.

[10] J. Lennox, H. Schulzrinne, and J. Rosenberg. *Common Gateway Interface for SIP, RFC 3050*. Internet Engineering Task Force, 2001.

[11] M. Rizzo and A. Garyfalos. Using SIP to negotiate over user requirements in personalized Internet Telephony services. In *Proceedings of SIP 2000*. Upper Side Conferences, Paris, January 2000.

[12] M. Kolberg and E. Magill. Handling incompatibilities between services deployed on ip-based networks. In *IEEE Intelligent Networks Workshop 2001 (IN2001), Boston, USA.*, May 2001.

[13] SER SIP Express Router. `http://www.iptel.org/ser`.

[14] Pingtel SIP Phone. `http://www.pingtel.com`.

[15] Kphone SIP User Agent. `http://www.wirlab.net/kphone`.