Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 51 (2007) 536–557
Computer Networks
www.elsevier.com/locate/comnet

Managing feature interactions between distributed SIP call control services

Mario Kolberg *, Evan H. Magill

Department of Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, United Kingdom

Available online 20 September 2006

Responsible Editor: H. Rudin

Abstract

The Session Initiation Protocol (SIP) is widely used 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 technique is particularly strong in handling interactions between distributed services – a key characteristic of SIP-based services. Moreover, the approach is fully distributed without any centralised components, and includes detection *and* resolution of feature interactions.

Keywords: Telecommunications services; Voice over IP; Session initiation protocol; SIP; Feature interaction; Runtime approach

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 termed feature or service interactions [2].

1.1. Basic terms

Within feature interactions in 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

* Corresponding author. Tel.: +44 178646447; fax: +44 1786464551.
E-mail addresses: mko@cs.stir.ac.uk (M. Kolberg), ehm@cs.stir.ac.uk (E.H. Magill).

1389-1286/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2006.08.006

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 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.

As examples of interactions, consider the following scenario during setting up a voice over IP session using SIP. If a user who subscribes to *call waiting* (CW) and *call forward when busy* (CFB) is busy, then what will happen when there is an invitation to another session? If the invitation is forwarded, then the CW service is clearly compromised, and vice versa. In either case, the user will not have their expectations met.

More subtle interactions can occur when more than one user is involved. For example, consider the scenario where user A subscribes to a originating call screening (OCS) service, with user C on the screening list, and user B subscribes to CFB to user C. If A invites B, and the invitation is forwarded to C, as prescribed by B's service CFB, then A's service OCS is compromised. Clearly, if the invitation is not forwarded, then the CFB service is compromised. These kinds of interactions can be very difficult to handle, particularly since different services may be activated at different stages of the session setup, and may be deployed on different components in the SIP network.

As can be seen, feature interactions do not occur because of badly coded services, but because of broken assumptions and conflicting goals of the services involved. For instance, OCS assumes that the party the session is aimed at, will be the party the session is actually established with. However, CFB breaks this assumption.

Services with comparable functionality are already known from the traditional telephony network, however, they are also being deployed on SIP networks. SIP has some fundamental differences to traditional telephony networks and hence it is necessary to revisit these problems 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 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. Service examples

Throughout the paper reference is made to a number of common call control services. 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 which exhibits the same functionality as a forking proxy. Briefly, the services exhibit the following behaviour:

- *Call Forwarding Unconditional* (CFU) redirects all incoming calls to a predefined 3rd Party.
- *Call Forwarding on Busy* (CFB) redirects all incoming calls to a predefined 3rd Party when the subscriber is busy.
- *Originating Call Screening* (OCS) screens all directory numbers of outgoing calls against a database. If a number matches an entry in the database the call is blocked and the subscriber is redirected to an announcement. This is strictly Originating Dial Screening.
- *Terminating Call Screening* (TCS) for all incoming calls, it screens the directory numbers of the originating party against a database. If a number matches an entry in the database the call is blocked and the originator is redirected to an announcement.
- *Voice Mail System* (VMS) redirects the caller to voice mail.
- *Automatic Ringback* (AR) is activated when the subscriber is busy. It returns the call to the caller after the subscriber is idle again.
- *Do Not Disturb* (DND) redirects all incoming calls to an announcement.
- *Group Ringing* (GR) allows for an incoming call to ring on an additional phone(s). The phone which goes offhook first is connected to the originator. The remaining phone will be idle again. In a SIP environment this functionality is known as *Forking*.
- *Hotline* (HL) connects the subscriber to a preconfigured party without dialling their number.

1.3. Basic approaches

A substantial body of work [3–9] 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 [10].

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 issue 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 [2 and 11].

This paper builds on a pragmatic off-line approach [12]. Previously, that work had already been successfully applied at runtime in a traditional telephony setting [10]. It has also formed the base for a successful runtime *detection* approach in a SIP environment [13]. Here, the latter work is extended to include *resolution* of detected interactions. Resolution is crucial for runtime approaches, as simply detecting an interaction does not improve the situation for the users.

The remainder of this paper is structured as follows: The next section presents an introduction to the Session Initiation Protocol. Section 3 suggests a pragmatic approach which can be applied at runtime to SIP services. Section 4 discusses how the approach can be implemented in SIP and Section 5 presents experiences made with a test implementation. Section 6 discusses experimental results. The final section summarises the paper.

2. SIP

2.1. Architecture and components

Voice over IP uses a number of different protocols. SIP [14] 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 (streaming) as such. Other protocols are used in combination with SIP to allow for different media to be transmitted. 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 to the originator 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 on which 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. Therefore, SIP will allow a degree of programmability which is unknown in the PSTN.

2.2. SIP messages

SIP distinguishes two message types: *requests* and *responses*. Requests are messages sent from the user agent client to the user agent server. Responses are messages sent from the server to the client. SIP supports six request messages:

REGISTER is used to register contact information with a register server.
INVITE is used to establish a SIP session.
ACK is used in a three-way handshake after an INVITE to confirm the establishment of a session.
CANCEL is used to cancel the previous request.
BYE is used to terminate an existing session.
OPTIONS is used to query a server on its capabilities, e.g., supported SIP extensions.

SIP responses are based on HTTP responses and are distinguished by a three-digit status code. The first digit specifies the class of the response.

1** Provisional: The server has received the request and continues to process it.
2** Success: The request was successfully received, understood and accepted.
3** Redirection: A further action needs to be taken to complete the request. Usually the reply contains an address to which the client should direct subsequent requests to. This reply is commonly used by redirect servers.
4** Client Error: The request contains bad syntax or cannot be fulfilled at this server.
5** Server Error: The server failed in fulfilling an apparently valid request.
6** Global Failure: The request cannot be fulfilled at any server.

Responses of the 1** class are said to be *provisional*, whereas all other responses are *final*. Provisional responses need to be followed by a further final response.

Furthermore, SIP defines a *transaction*, which comprises all messages from the first request sent to the server up to the final response sent back to the client. INVITE requests use a three-way-handshake which consists of the INVITE message, possibly provisional responses, a final response and an ACK request. All other requests are answered by a final response which may be preceded by provisional responses.

SIP messages contain a number of *headers* providing more detail on the request, response and transaction. While there are only a few SIP messages, SIP is quite complex. This complexity is largely introduced by headers. There are a number of headers specified in the SIP standard, and furthermore, additional headers may be defined. If a server or user agent does not understand a header it simply ignores it. The OPTIONS request can help to establish what extensions the other party supports. The list below provides an overview of the most common headers (Table 1).

However, the specific use of headers is best explained with the aid of an example. The following SIP messages establish and terminate a two party session. Please note that any payload of messages has been omitted for brevity. Ref. [15] provides more details and example call flows.

Table 1
SIP headers

Header	Meaning
To	Destination address
From	Originator address
Call-Id	Session ID
Cseq	Sequential Number of request within a session
Contact	User Agent Client address
Via	Addresses of nodes message has passed through
Contents-Type	Type of payload in the message
Contents-Length	Length payload in the message
Allow	Requests understood by client
Supported	SIP extensions supported by client

```
INVITE sip:alice@d254203.cs.stir.ac.uk SIP/2.0
From: sip:chris@discus.cs.stir.ac.uk;tag=1c17644
To: sip:alice@d254203.cs.stir.ac.uk
Call-Id: call-1058523015-27@139.153.254.222
Cseq: 1 INVITE
Contact: <sip:chris@139.153.254.222>
Via: SIP/2.0/UDP 139.153.254.222
Content-Type: application/sdp
Accept-Language: en
Allow: INVITE, ACK, CANCEL, BYE, REFER, OPTIONS, REGISTER
Supported: sip-cc, sip-cc-01, timer, replaces
User-Agent: Pingtel/2.1.8 (VxWorks)
Date: Fri, 18 Jul 2003 10:10:15 GMT
```

The INVITE message is sent from the user agent client to the user agent server. The first line in the message provides the message name (INVITE), and the address where the message is to be sent to next. It is important to note that this is not necessarily the final destination, but only the next hop. At the end, the version of the SIP protocol is specified. The next two headers **From** and **To** specify who originated the transaction and its destination. These addresses are public addresses and do not specify the actual location of the users. Furthermore, if the request gets forwarded to a different destination, the To field stays unchanged. It is only the first line of the message which takes the new address.

The header, **Call-Id** specifies an ID for the session. All subsequent messages belonging to this session will be assigned this ID. The following header, **Cseq** provides the number of the request within that session. Replies to a particular requests carry the same Cseq. However, the Cseq is not always incremented for the next request. ACK request carry the same Cseq as the corresponding INVITE and CANCEL request carry the same Cseq as the requests which is to be cancelled.

The **Contact** specifies the address of the user agent client. This is the physical location of the user, and thus not identical to the public address specified in the From field. The Contact field is provided to allow for the media connection to be established directly between the user agent server and client. The **Via** header contain the addresses of all nodes the message has passed on its way, i.e., the user agent client and all proxies include their address as a Via header. In this example there is only one; however, as the request is getting closer to its destination there may be a number of Via headers in the message. This allows response messages to take the same path as the corresponding request.

The headers **Contents-Type** and **Contents-Length** are concerned with the body or payload of the request. SDP is a protocol to specify the media to be exchanged after establishment of the session. Commonly INVITE requests and 200OK response messages carry a SDP payload. These two messages are used to negotiate the media for the session. Basically, INVITE contains an offer and the 200OK message either accepts the offer (identical payload) or makes a counter proposal which is usually one of the options offered in the INVITE. Since call control is usually not concerned with the payload it has been omitted in this paper.

The header **Allow** specifies SIP requests the client understands, and the header **Supported** specifies SIP extensions which are supported by the client. The **User-Agent** header contains a string providing more detail on the user agent used. Finally, the **Date** header specifies the date and time when the request was issued.

```
SIP/2.0 100 trying – your call is important to us
From: sip:chris@discus.cs.stir.ac.uk;tag=1c17644
To: sip:alice@d254203.cs.stir.ac.uk
Call-Id: call-1058523015-27@139.153.254.222
Cseq: 1 INVITE
Via: SIP/2.0/UDP 139.153.254.222
Server: Sip EXpress router (0.8.11pre36 (i386/linux))
```

The second message in the transaction is a **100 Trying**. This is usually the first reply sent by a proxy server to the user agent client. It means that the proxy has received the request and is processing it. There are no new header fields in this message, except the **Server** header. This is similar to the User-Agent header discussed above.

Note that the From and To fields have not changed, they stay the same during the transaction. Similarly, Call-Id and Cseq are identical to the INVITE, i.e., this response is referring to the INVITE request. The address in the Via header is also the same as before, this means that the response should go via the address specified. Also, since there is only

one Via header, there is only one final hop remaining, which is the user agent client. The Content-Length header is set to 0, as there is no payload with provisional responses.

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 139.153.254.222
From: <sip:chris@discus.cs.stir.ac.uk>;
tag=1c17644
CSeq: 1 INVITE
Call-ID: call-1058523015-27@139.153.
254.222
To: <sip:alice@d254203.cs.stir.ac.uk>;
tag=2FFCD6CF
User-Agent: KPhone/3.11
Contact: "root" <sip:root@139.153.254.
203:5062;transport=udp>
```

The next message which is sent is a **180 Ringing** response. This is sent after the proxy server has contacted the user agent server (terminating UA) and the user is being alerted. Again, the From, To, Via, Cseq, Call-Id, and Content-Length headers are identical to the previous response. However, this message has a Contact field which specifies the location of the invited user.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 139.153.254.222
CSeq: 1 INVITE
Content-Type: application/sdp
Call-ID: call-1058523015-27@139.153.
254.222
To: <sip:alice@d254203.cs.stir.ac.uk>;
tag=2FFCD6CF
User-Agent: KPhone/3.11
Contact: "root" <sip:root@139.153.254.
203:5062;transport=udp>
```

The next message is a **200OK** message which signals success. This is a final response and is sent after the user accepts the call. In traditional telephony the user would have gone off-hook. The headers are as discussed before.

```
ACK sip:root@139.153.254.203:5062;
transport=udp SIP/2.0
Contact: sip:chris@139.153.254.222
From: <sip:chris@discus.cs.stir.ac.uk>;
tag=1c17644
To: <sip:alice@d254203.cs.stir.ac.uk>;
tag=2FFCD6CF
Call-Id: call-1058523015-27@139.153.
254.222
Cseq: 1 ACK
User-Agent: Pingtel/2.1.8 (VxWorks)
Date: Fri, 18 Jul 2003 10:10:22 GMT
Via: SIP/2.0/UDP 139.153.254.222
```

The next message, an ACK request, establishes the session. It is sent from the user agent client (originating UA) to the user agent server (terminating UA). The media stream will be established between the Contact addresses of the two parties chris@139.153.254.222 and root@139.153.254.203:5062. Note that the Cseq counter has not increased even though this is a new request.

```
BYE sip:root@139.153.254.203:5062;
transport=udp SIP/2.0
From: sip:chris@discus.cs.stir.ac.uk;
tag=1c17644
To: sip:alice@d254203.cs.stir.ac.uk;
tag=2FFCD6CF
Call-Id: call-1058523015-27@139.153.
254.222
Cseq: 2 BYE
Supported: sip-cc, sip-cc-01, timer,
replaces
User-Agent: Pingtel/2.1.8 (VxWorks)
Date: Fri, 18 Jul 2003 10:10:23 GMT
Via: SIP/2.0/UDP 139.153.254.222
```

The session is terminated by sending an BYE request. All header fields are like the previous request messages, except Cseq which has now been increased by 1. Note that the BYE request can be sent from either party.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 139.153.254.222
From: <sip:chris@discus.cs.stir.ac.uk>;
tag=1c17644
CSeq: 2 BYE
Call-ID: call-1058523015-27@139.153.
254.222
To: <sip:alice@d254203.cs.stir.ac.uk>;
tag=2FFCD6CF
User-Agent: KPhone/3.11
Contact: "root" <sip:root@139.153.254.
203: 5062;transport=udp>
```

The BYE request is confirmed with a 200OK response from the other party.

2.3. 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 may compete with each other. Thus they are not inclined to share detailed information on their services to avoid interactions. Also, a large degree of programmability allows end users to 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 from a range of independent stakeholders. Any approach to feature interactions for SIP needs to take this into account.

The fact that media packets travel end-to-end (except if B2BUA (Back to Back User Agent) are used), 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 to which they should send the media packets [1]. This is clearly an advantage for feature interaction as it prevents some broken assumptions.

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 this approach.

3. The applied approach

The algorithm builds upon the pragmatic approach presented in [12]. The approach concentrates on the establishment of connections and does not require detailed information about the involved services, but operates at the connection 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. It is adapted to operate at runtime in PSTN [10] and SIP environments [13]. Here the latter is extended to include resolution of interactions.

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.

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.: B indicates that B is the triggering party, as CFU is triggered at the terminating end of a call. In the second part, notation (A, B) → (A, C) indicates the connection type. (A, B) is the originally proposed connection (called original connection) and (A, C) is the connection after activating the service (called resulting connection). For a pair, such as (A, B), A is the source and B the destination.

In this example, the call starts with A attempting to connect to B. However, because of CFU, A is connected to C instead. This leads to the connection type (A, B) → (A, C).

The originally intended connection and the connection set up after activation of the service do not need to be part of the same call. While in the above CFU example the resulting connection is set up immediately after triggering the service, this might not be the case with some other services. For instance Automatic Ringback which is triggered by an incoming call to the subscribers line while they are engaged in another call, only returns the call to the caller after the subscriber is idle again. Hence a connection type may describe either a single or two calls. However, the order in which the calls are established is maintained, i.e., the resulting connection is only established *after* the original connection was attempted to be set up. As an example, Fig. 2 contains the specification for Automatic Ringback.

Treatments are an important aspect of this approach. Treatments are announcements or tones

TP.: B; (A, B) → (A, C)

Fig. 1. Description of Call Forwarding Unconditional.

TP.: B; (A, B) → (B, A)

Fig. 2. Specification of Automatic Ringback.

triggered by the network to handle certain conditions during a call, for example when a call is screened or blocked. Potentially, there may be multiple treatments involved in a call. For example, consider two services invoked during one call. One service might connect a party to a busy treatment, whereas the other service connects the (same or the) other party to a network unavailable treatment.

Clearly, this way of describing services abstracts from a considerable amount of service behaviour. However, the presented approach was aimed to be as abstract as possible while remaining useful. 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. Importantly, the order of the services within a pair does not have any influence on the result. Thus changing the order of services does not affect the functioning of the approach.

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. Note that, even if the services aim at setting up the same connection, the services may clash as they may be triggered simultaneously. Examples are given in Fig. 3. The shaded portions indicate the key parts of the descriptions.

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. Fig. 4). Again, the shaded portions indicate the identical connections.

a

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

b

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

Fig. 3. (a) Call Forwarding Busy & Call Waiting and (b) Automatic Ringback & Hotline.

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

Fig. 4. Call Forwarding Busy and Call Forwarding Unconditional revisited.

Rule 3 – Redirection and Treatment: This type of interaction is detected if the resulting connection of one service matches the original connection of the other service which connects to a treatment. Furthermore, either the originating parties for the original and resulting connections of one service need to be identical and the terminating parties be different, e.g., (A,C) → (A,B), or alternatively, 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, e.g., (A,B) → (B,A).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. Fig. 5 provides two examples.

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

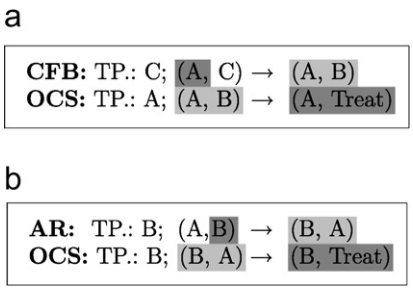

Fig. 5. (a) Call Forwarding Busy & Originating Call Screening and (b) Automatic Ringback & Originating Call Screening.

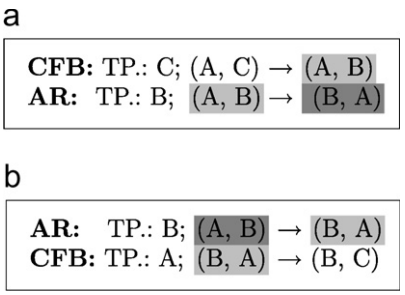

Fig. 6. (a) Call Forwarding Busy & Automatic Ringback and (b) Automatic Ringback & Call Forwarding Busy.

be the terminating part 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.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. Fig. 6 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 and for one service, the triggering party is also the originating party of original connection and the terminating party of the resulting connection is treatment. 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 Fig. 7.

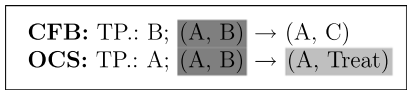

Fig. 7. Call Forwarding on Busy and Originating Call Screening revisited.

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. Applying the approach to SIP

4.1. SIP services

SIP allows for much more expressive messages to be exchanged between components than was possible in the PSTN. Headers as discussed in Section 2.2, convey a lot of information about the call to be set up. However, interactions are due to broken assumptions and conflicting goals between the services, and cannot be prevented by rich messages.

In SIP, services can be provided in a number of different ways, for instance, SIP CPL [16], SIP LESS [17], SIP CGI [18] and SIP-Servlets [19]. Additionally, high-level APIs such as JAIN and Parlay can be used. SIP CPL and LESS are XML-based languages and are deliberately restricted in their functionality. The other approaches offer 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 these "full" services. In fact, the approach can be applied to any service which can be surrounded by a cocoon (as discussed in Section 4.2.1). The case study in Section 5 has been carried out on SIP CGI-type services.

Below, it is discussed how some important call control services are reflected in SIP messages. Most call control services are active while the session is

being set up. Consequently, INVITE messages are the most important ones as services are triggered by them.

4.1.1. Call Forwarding Unconditional

According to work carried out within the IETF on SIP call control services [15] call forwarding of a request is achieved by rewriting the address in the first line of the message. The **To** header is not changed. The To header is supposed to show the address the call started off with. The example below shows relevant parts of two INVITE messages. The first is received by a proxy server and the second is sent by the proxy server. Note the first line in each message. In the first message, the INVITE is sent to bob@d254203.cs.stir.ac.uk.

```
INVITE sip:bob@d254203.cs.stir.ac.uk SIP/2.0
From:sip:chris@discus.cs.stir.ac.uk;tag=1c28023
To: sip:bob@d254203.cs.stir.ac.uk
Call-Id: call-1058520667-25@139.153.254.222
Cseq: 1 INVITE
Contact: <sip:chris@139.153.254.222>
Via: SIP/2.0/UDP 139.153.254.222
```

However, in the message below the INVITE is forwarded to alice@d254203.cs.stir.ac.uk. Note that the To header still contains Bob's address.

```
INVITE sip:alice@d254203.cs.stir.ac.uk SIP/2.0
From: sip:chris@discus.cs.stir.ac.uk;tag=1c28023
To: sip:bob@d254203.cs.stir.ac.uk
Call-Id: call-1058520667-25@139.153.254.222
Cseq: 1 INVITE
Contact: <sip:chris@139.153.254.222>
Via: SIP/2.0/UDP 139.153.254.50;branch=z9hG4bKa3fd.cd0ac2e7.0
Via: SIP/2.0/UDP 139.153.254.222
```

The service description for this example is constructed as follows. The Triggering Party is set to the address specified in the first line of the message. As this address is the next destination of a message, if a proxy (or any other SIP component) receives an INVITE request, the address in the first line is the address of the proxy. For this example it would be bob@d254203.cs.stir.ac.uk. For the original connection, the originating party is the address in the From header, and the terminating party is the same as the triggering party. It is important to note that the To header field cannot be used as the message may already have been forwarded. The originating party for the actual connection is also the address in the From header, and the terminating party is the address where the call is to be forwarded to (Fig. 8).

The above example assumes that the service is deployed on a SIP proxy server. However, it may also be present on the user agent server or a redirect server. When deployed on a user agent server, the messages are not very different, the second INVITE is simply sent by a different component. However, if the service deployed on a redirect server, the messaging is different in that the server does not forward the request, but sends a 301 Moved temporarily or 302 Moved permanently response with the new address in the Contact header back to the UAC. This will then trigger a new INVITE to the specified address. As this does not change the fundamental behaviour of the service (a different party gets the invitation), it has no effect on the description of the service.

4.1.2. Call Forwarding on Busy

In traditional telephony the switch is aware if a user is busy. Thus services which handle the busy state of a terminal, such as Call Forwarding on Busy or Call Waiting, can be triggered on the alert message. However, in SIP, a proxy server is not aware whether the user agent is busy. This occurs for two reasons. Firstly, the proxy may operate in the stateless mode. In this case the proxy only forwards messages and is not aware at what point in a session these messages are sent and indeed if they are legal. The second reason is that messages may be sent to the user agent bypassing the proxy server altogether. Thus services handling the busy state of a user agent need to operate differently from equivalent services in the PSTN.

In SIP, the busy condition is only known after the INVITE has been forwarded to the user agent

CFU: TP.: bob@d254203.cs.stir.ac.uk;
(chris@discus.cs.stir.ac.uk, bob@d254203.cs.stir.ac.uk) →
(chris@discus.cs.stir.ac.uk, alice@d254203.cs.stir.ac.uk)

Fig. 8. Description of CFU.

and the user agent has replied with a response message **486 Busy**. Hence, services like call forwarding on busy and call waiting will only be triggered on the busy message, not the original INVITE. Using the example of call forwarding on busy, this is shown in Fig. 9.

In the example, Alice tries to INVITE Bob. Bob can be reached at two locations Bob1 and Bob2 and Bob wants all calls to Bob1 forwarded to Bob2 if Bob1 is busy. The busy message shown by a dashed arrow triggers Bob1's CFB service. Dropping the busy message and issuing the INVITE shown dashed is a result of the CFB service.

As with CFU, CFB may also be provided on a user agent server. Again, this makes little difference to the messaging. The user agent server, simply forwards the invitation onto the new destination. Deploying CFB on a redirect server is not possible, as the server is not aware of the busy condition, as it does not see the response from the user agent server.

Even though Call Forwarding on Busy is triggered in a different way that Call Forwarding Unconditional, CFB and CFU services are described in the same way. Thus for the example shown in Fig. 9 the description for CFB is as shown in Fig. 10.

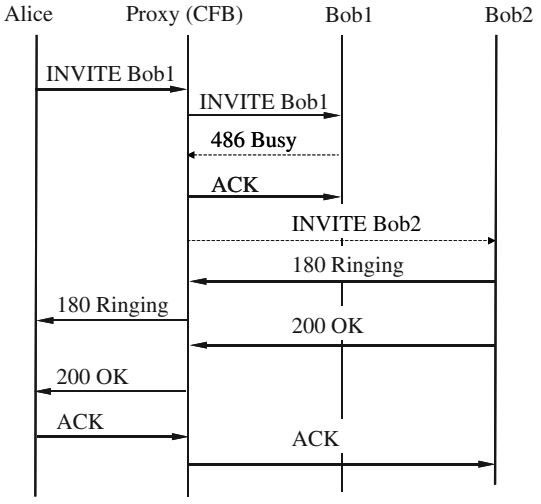

Fig. 9. Call Forwarding Busy in SIP.

CFB: TP.: Bob1; (Alice, Bob1) → (Alice, Bob2)

Fig. 10. Description of CFB.

4.1.3. Terminating call screening

A screening service in SIP can be implemented by screening relevant header fields or by asking for explicit authentication. While the authentication option is suggested in [15], at least for terminating call screening it has the disadvantage that calls can only be received from parties which are known and been granted explicit permission. This is a positive list, rather that the negative list used in traditional telephony. This appears to be very restrictive and not practical for most purposes. To allow for more flexible screening, the approach of checking header fields is adopted.

Terminating call screening checks the address in the From header field. If that address matches any address in the screening list, the call is screened. The triggering party again is the address specified in the first line in the message (address of current location). The address in the From header field is used for the originating parties in the connection type. Considering the second message in Section 4.1.1, and assuming that alice@d254203.cs.stir.ac.uk does not wish to be called from chris@discus.cs.stir.ac.uk, the description as shown in Fig. 11 can be derived.

TCS may be deployed on proxy servers, redirect servers and user agent servers alike. The message flow is largely identical for all three components. On receiving an invitation from a screened source, an error response message is returned (see Fig. 12).

4.1.4. Originating call screening

Originating call screening checks the Request-URI (address in the first line of an INVITE message). If that address is also in the screening list, the call is not allowed. The triggering party is the user name given in the From field and the address of the current location. As OCS is associated with the calling party it will usually be deployed on the

TCS: TP.: alice@d254203.cs.stir.ac.uk;
(chris@discus.cs.stir.ac.uk, alice@d254203.cs.stir.ac.uk) →
(chris@discus.cs.stir.ac.uk, treatment)

Fig. 11. Model of SIP TCS.

OCS: TP.: chris@discus.cs.stir.ac.uk;
(chris@discus.cs.stir.ac.uk, bob@d254203.cs.stir.ac.uk) →
(chris@discus.cs.stir.ac.uk, treatment)

Fig. 12. Model of SIP OCS.

local proxy server. In this case, the address for the triggering party will be the public address of the user agent client, and thus will match the address given in the From header. However, OCS may also be deployed directly in the user agent client. Then the address in the From header will not match the actual triggering location.

Using the example of the first INVITE message in Section 4.1.1, and assuming the current location is the local proxy server (discus.cs.stir.ac.uk), the following connection type for OCS can be derived:

4.2. Implementation of the approach in SIP

4.2.1. The ConType header

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 gains from being distributed as well.

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 distributed nature of the approach is one of its core features. It makes the approach highly scalable and resilient to a single component failure. Furthermore it cuts down on additional network messages.

The approach can be implemented, 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. Cocoons have been successfully adopted in the PSTN before by Marples et al. [20–22]. 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 then 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 Fig. 13.

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 triggering 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.

4.2.2. Interaction resolution

If a service interaction is detected, it needs to be resolved also at run-time. There appears no other

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

Fig. 13. SIP header for CFU.

solution to the problem in the literature, than to disable one service completely. Having a joint solution including some aspects of one service and some aspects of the other is not practical because services have a well defined, rather focussed task which can usually not be split. Hence, as the interaction will be between a pair of services, any interaction can hence be resolved by disabling one of the two services.

There are already a number of different approaches presented in the literature which help to decide which service should be disabled to resolve the interaction. The most flexible one appears to be the use of policies [23]. However, other approaches [24,25] may also be applicable.

These papers show that the decision on which of the two services should be disabled is very much context specific and as such beyond the scope of this paper. In this paper the focus is providing a mechanism to execute the decision.

Above resolution approaches can be tuned to various preferences, 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 effect all these approaches are implementing a priority system.

Essentially, the cocoon would communicate with any of these resolution techniques (the precise communication will depend on the chosen resolution technique). The received result would then be executed by the cocoon. Fig. 14 depicts this scenario. However, if a rather straightforward resolution approach is selected, this could be integrated with the cocoon itself.

The approach taken to support the decision has two alternatives depending on which of the two services is to be disabled.

If the second service (the current one) was chosen to be disabled, its outcome is simply discarded, and the call proceeds as if that service was not activated at all.

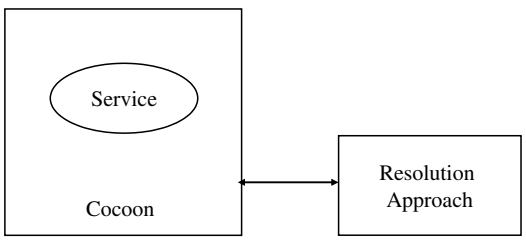

Fig. 14. Link between Cocoon and Resolution Approach.

If, however, the first service was chosen to be disabled, the resolution is slightly more complicated. The session setup attempt needs to be repeated, but without activation of the first service. To achieve this, the current session setup attempt is stopped and an appropriate response message passed back to the originator. As this process is similar to the 3xx responses in SIP used for redirection, the SIP Message 380 Alternative Service may be appropriate for this. To communicate the selected service down the chain, the ConType header for this service is copied into this message and extended with a field 'Status' set to 'disabled'. An example is shown in Fig. 15.

The originating user agent will receive this message, and initiate a new INVITE request. This will again include the ConType header copied from the 380 response (including the Status field). When a service gets triggered with this INVITE, the cocoon will check the data in the ConType header. If the triggered service matches the ID and triggering party in this header, the actions of that service will be discarded and the ConType header be removed from the message. Thus the session setup will progress without that service being activated.

Clearly, because a service is disabled the path the INVITE takes subsequently may be different to the first session setup attempt. Consequently, further pairs of incompatible services may be encountered. This is unavoidable and repeated session attempts may be required to resolve the situation. However, this is not uncommon in SIP; especially redirection works in a similar fashion. Furthermore, it is expected that re-attempts caused by this approach will be rare.

4.2.3. Services triggered on responses

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

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;
Status=disabled

Fig. 15. Disabling a CFU Service.

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 Fig. 16.

In this 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. The descriptions of the two services are shown in Fig. 17.

Resolving such an interaction follows the same pattern as discussed above. If the second service is to be disabled (CFB in the example), its actions are disregarded and the 486 Busy message is

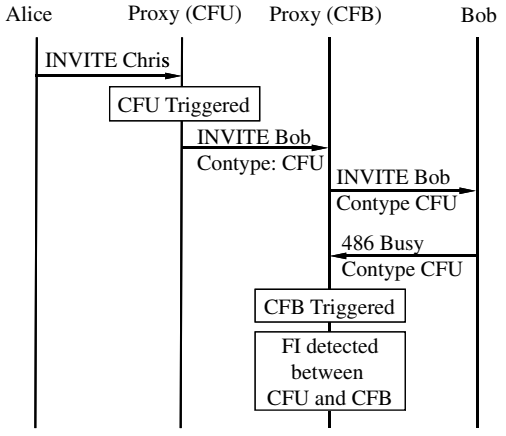

Fig. 16. Message flow with CFU and CFB.

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

Fig. 17. Call Forwarding Loop.

forwarded back to Alice. If, however, the first service (CFU in the example) is to be disabled, a 380 message as discussed above is returned to Alice. It will have one ConType header with the 'Status' set to 'disabled'. Consequently, a new INVITE will be initiated as above.

4.2.4. Impact on SIP

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.

An investigation was carried out if an already defined SIP header could be used, rather than defining the ConType header. Strong candidates were the History-Info (RFC 4244 [26]) and Reason headers (RFC 3326 [27]). However, both headers have been designed for a particular purpose which is defined rather narrowly. This is also reflected in the defined format for these headers. Hence the decision was made to define a new header. In time, it may be possible to integrate the information with the History-Info header.

The use of the 380 Response message to indicate a resolution to a feature interaction needs to be agreed within the related IETF working groups. In the current RFC [14] the meaning of this message is left largely undefined and hence may be available for the purpose described here. However, the used message code does not affect the working of the approach.

Our experiments showed that beside the ConType header, a second additional header is

required to allow this approach to operate in practise. This is discussed in Section 5.2.

5. Experimental case study

The approach has been implemented in a SIP testbed. SER [28] was chosen as proxy server, and a Pingtel SIP phone [29] and kphone [30] were used as user agents.

Currently, it is difficult to get access to implementations which support the provisioning of services generally, and even more to implementations which support the CGI interface. However, SER offers a proprietary interface to provide 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 adhering to the functionality described in the relevant IETF RFC [15] 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 of 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 Fig. 18.

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
```

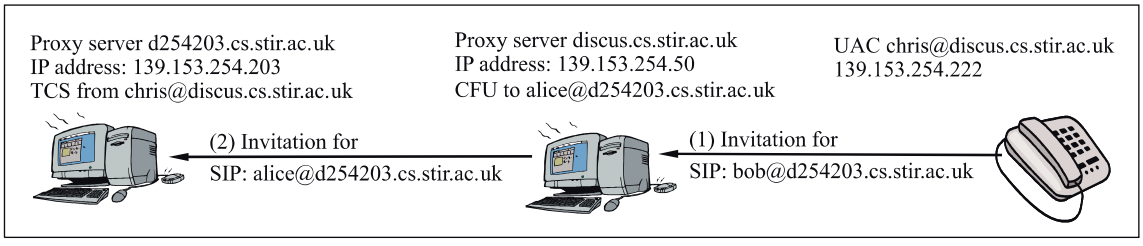

Fig. 18. Applying the Algorithm to the Interaction between CFU and TCS.

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.

As discussed in the previous Section, a decision has to be made which of the two services will be disabled. For this example it is assumed that the first service, CFU, was chosen to be disabled. Consequently, a 380 response is sent back to the originating user agent. For this example, the following 380 response message will be sent to Chris' user agent. Note the Status field in the ConType header. The proxy servers on the way will simply forward the message in standard SIP fashion.

```
SIP/2.0 380 Alternative Service
From: sip:chris@discus.cs.stir.ac.uk;
tag=lcl8932
To: sip:bob@d254203.cs.stir.ac.uk
Via: SIP/2.0/UDP l39.153.254.50;
branch=z9hG4bK6d0d.a5da393.0
Via: SIP/2.0/UDP l39.153.254.222
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;
Status=disabled
```

On reception of the 380 message, Chris' user agent will initiate a new INVITE message to be sent to the first proxy server. The message is depicted below.

```
INVITE sip:bob@d254203.cs.stir.ac.uk
SIP/2.0
From: sip:chris@discus.cs.stir.ac.uk;
tag=lcl8932
To: sip:bob@d254203.cs.stir.ac.uk
Contact: <sip:chris@l39.153.254.222>
Via: SIP/2.0/UDP l39.153.254.222
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;
Status=disabled
```

The only difference to the original INVITE is the inclusion of the ConType header, again with the Status field. The message is received by the first proxy server and will trigger the CFU service again. However, the cocoon discovers the existing ConType header with the Status field. The ID and TP fields of that ConType header are checked against the ones from the CFU service. As they match, the actions of the CFU service are discarded, and the ConType header removed from the message. Subsequently, the INVITE message will be forwarded to Bob. The approach has successfully detected and resolved 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. Fig. 19 shows the setup.

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 (see Fig. 20).

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 took no actions and hence the cocoon does not add 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

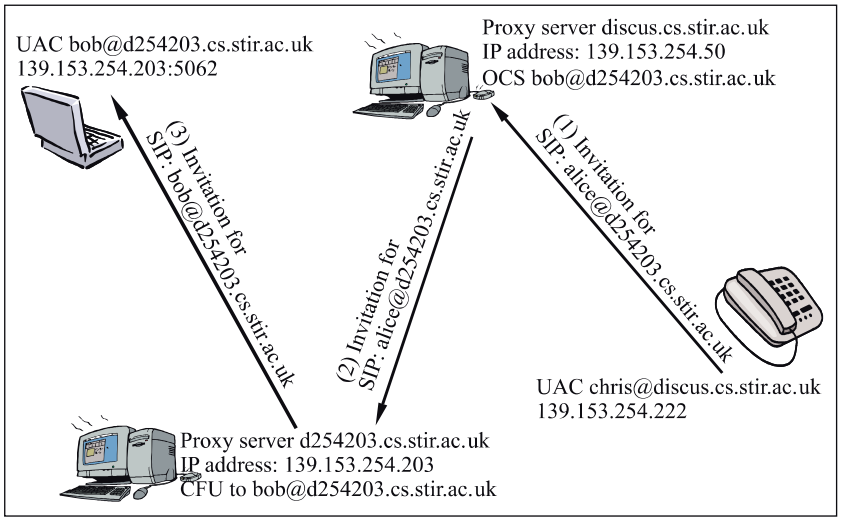

Fig. 19. Service Interaction between OCS and CFU.

	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									√ √

Fig. 20. Results of SIP case study.

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=1A80l9B3
Contact: "root" <sip:root@l39.l53.254.203: 5062;transport=udp>
Record-Route: <sip:alice@l39.153.254.203;ftag=1c8060;lr=lr>, <sip:alice@l39.153.254.50;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, experiments showed that 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 were 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=1cl7644;lr= lr>,
<sip:alice@139.153.  254.50;ftag=1cl7
644;lr=lr>
<sip:bob@139.153.254.23;ftag=1cl7644;
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 accepted 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 with the previous example, a decision needs to be made which of the two services should be disabled for resolution. For this scenario it is assumed that the second service, OCS, is chosen to be disabled. As discussed in Section 4.2, in this case the algorithm prescribes that the actions of the services will simply be disregarded. Hence the session between Chris and Bob will be setup in this example. Again, the approach has successfully detected and resolved the interaction.

If the other service (CFU) is to be disabled, the approach as discussed in the previous section is adopted.

6. Results

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. A short description of their behaviour is provided in Section 1.2.

Following the modelling approach from Section 3.1 the connection type for each of the services was developed. Table 2 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.

Furthermore, some rather different services have identical descriptions. For instance, TCS, DND and

Table 2
Specifications of the case study services

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

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.

6.2. Discussion

Fig. 20 provides details of the performance of the approach. 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. These scenarios involved both SIP servers and SIP user agents. All detected interactions were also successfully resolved.

All the 'x'-entries in the table refer 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. If the service dropping the INVITE is triggered first, the second service will not be triggered at all. Hence the detection algorithm in the cocoon of the second service is not activated. This also applies to scenarios where both services drop the triggering message. In a way, such interactions which in the PSTN could be classified as Shared Trigger Interactions (according to Marples' taxonomy) [22,10] have essentially become Missed Trigger Interactions in SIP. Missed Trigger Interactions are notoriously difficult to detect and cannot be handled by this approach.

On the other hand, it could be argued that interactions between services deployed on the same server (or user agent) are in fact not interactions in SIP components. This is because deployed services are usually configured in a list in SIP, 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 observed in two different scenarios. Firstly, the arguably more traditional setup where both services are deployed at the same location. However, the interaction can also successfully be verified and resolved when HL and OCS are deployed at different locations, i.e., Hotline at the user agent client and OCS at 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 is 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 its 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 exist in SIP. This applies to interactions which involve a service which handles a busy condition and another service which is triggered regardless of the party being busy. The message flow for the classic example between TCS and CFB is depicted in Fig. 21.

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

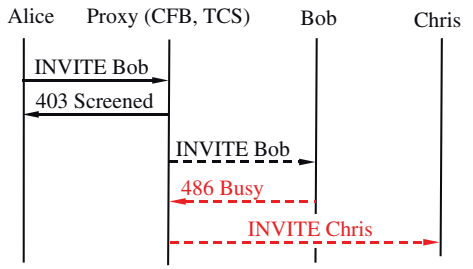

Fig. 21. SIP Call Scenario between CFB and TCS.

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 to operate at runtime. 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.

Conversely, the approach does not expose any service information beyond what users can observe from normal session setup behaviour anyway. For instance, redirecting a call or sending an invitation to voice mail is clearly visible to users. In the examples in this paper, the name of the service is used as the ID within the ConType header. If revealing which service triggered the action is of concern, the ID can be constructed differently. It just has to be unique between services on a particular SIP component. Hence privacy is not an issue with the approach.

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 set up 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. Furthermore, the approach is fully distributed, and does not need any central component. This makes the approach very scalable.

Once an interaction is detected it also needs to be resolved. In this paper the resolution approach is based on disabling one of the services involved in the interaction. The decision which one of the two services involved in the interaction should be disabled is user specific. A strong candidate to implement this are policies. Policies are very flexible and can be adjusted to particular user preferences. The approach in this paper provides support to implement the decision by the external algorithm. The approach incurs minimal overhead and integrates well into the SIP protocol.

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).

One potential issue with the distributed nature of the approach is the existence of rouge components in the signalling chain. Such components may abuse the additional header included in the message or do not implement the approach correctly. However, this problem also applies to standard SIP components without the extension suggested here. As such, dealing with malicious components is beyond the scope of this paper.

Finally, this paper concentrates on core SIP functionality. However, there are many extensions to SIP, such as *presence* and *event notification*. These additions may have an impact on the message flows and hence on interactions. It will be interesting to investigate common SIP extensions in light of feature interactions.

In summary, the presented approach closes a major gap by providing a runtime resolution mechanism distributed across the SIP components. This will be necessary if distributed call control architectures, such as SIP, are to be successful. It will be interesting to investigate the applicability of this approach to the emerging Peer-to-Peer SIP (P2P-SIP) architecture currently being discussed within IETF.

References

[1] J. Lennox, H. Schulzrinne, Feature interaction in internet telephony, in: [7], May 2000, pp. 38–50.

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

[3] L.G. Bouma, H. Velthuijsen (Eds.), Feature Interactions in Telecommunications Systems, IOS Press, Amsterdam, 1994.

[4] K.E. Cheng, T. Ohta (Eds.), Feature Interactions in Telecommunications Systems III, IOS Press, Amsterdam, 1995.

[5] P. Dini, R. Boutaba, L. Logrippo (Eds.), Feature Interactions in Telecommunication Networks IV, IOS Press, Amsterdam, 1997.

[6] K. Kimbler, L.G. Bouma (Eds.), Feature Interactions in Telecommunications and Software Systems V, IOS Press, Amsterdam, 1998.

[7] M. Calder, E. Magill (Eds.), Feature Interactions in Telecommunications and Software Systems VI, IOS Press, Amsterdam, 2000.

[8] D. Amyot, L. Logrippo (Eds.), Feature Interactions in Telecommunications and Software Systems VII, IOS Press, Amsterdam, 2003.

[9] S. Reiff-Marganiec, M. Ryan (Eds.), Feature Interactions in Telecommunications and Software Systems VIII, IOS Press, Amsterdam, 2005.

[10] M. Calder, M. Kolberg, E.H. Magill, S. Reiff-Marganiec, Hybrid solutions to the feature interaction problem, in: [8], June 2003, pp. 295–312.

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

[12] M. Kolberg, 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) (2002) 591–602.

[13] M. Kolberg, E.H. Magill, Detecting feature interactions between sip call control services, in: [9], June 2005, pp. 147–162.

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

[15] A. Johnston, S. Donovan, R. Sparks, C. Cunningham, K. Summers, Session Initiation Protocol (SIP) Basic Call Flow Examples, IETF RFC 3665, December 2003.

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

[17] X. Wu, H. Schulzrinne, LESS: Language for End System Services in Internet Telephony, Internet Engineering Task Force, 2005, in preparation.

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

[19] Java Community Process, SIP Servlet API, Java Specification Request 116, 2003.

[20] D. Marples, E.H. Magill, The use of rollback to prevent incorrect operation of features in intelligent network based systems, in: [6], September 1998, pp. 115–134.

[21] M. Calder, E. Magill, D. Marples, A hybrid approach to software interworking problems: managing interactions between legacy and evolving telecommunications software, IEE Proceedings—Software 146 (3) (1999) 167–180, June.

[22] D. Marples, Detection and Resolution in of Feature Interactions in Telecommunications Systems at Runtime, Ph.D. Thesis, Communications Division, Department of Electrical and Electronic Engineering, University of Strathclyde, 2000.

[23] K.J. Turner, S. Reiff-Marganiec, L. Blair, J. Pang, T. Gray, P. Perry, J. Ireland, Policy support for call control, Computer Standards and Interfaces, Elsevier Science, in press, available online.

[24] 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.

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

[26] M. Barnes, An Extension to the Session Initiation Protocol (SIP) for Request History Information, RFC 4244, Internet Engineering Task Force, 2005.

[27] H. Schulzrinne, D. Oran, G. Camarillo, The Reason Header Field for the Session Initiation Protocol, RFC 3326, Internet Engineering Task Force, 2002.

[28] SER SIP Express Router. Available from: <http://www.iptel.org/ser>.

[29] Pingtel SIP Phone. Available from: <http://www.pingtel.com>.

[30] Kphone SIP User Agent. Available from: <http://www.wirlab.net/kphone>.

Mario Kolberg studied computer science at the HTWS Zittau/Goerlitz in Germany. After spending one year in the computer science department at Humboldt University in Berlin he joined the Communications Division in the Department of Electronic and Electrical Engineering at the University of Strathclyde in June 1997. In September 2000 he moved to the Department of Computing Science and Mathematics at Stirling University.

He is a member of the Communications and Services research group within the department. His research interests include Home Networks, Feature Interaction, Service Creation, and IP Telephony. He has extensive experience of home networking protocols and of implementing services on the OSGi platform. He is leading a project funded by Panasonic (USA) investigating service discovery for networked appliances. He is also involved with the MATCH project, focusing on integrating different technologies for care in the home. He was leading an effort providing a proof-of-concept demonstrating the integration of digital pen and paper with networked appliances. This is being exhibited in the trial home of The Home Application Initiative (TAHI), a DTI funded UK consortium on home automation.

Previously he has worked on an EU funded project (TOSCA) on issues related to the rapid development of distributed telecommunications services. On invitation by Telcordia Technologies, a leading presence in home automation, he has visited their laboratories in the USA for an extended period. He holds a Ph.D. in Electrical Engineering from the University of Strathclyde.

Evan H. Magill leads the Communications and Services research group at Stirling. His main research focus is on the creation and interworking of network services and applications with heterogeneous sets of devices and protocols. In particular he has sought ways to automate these processes to allow self-management.

He has published widely on these topics through scientific papers, conferences and books. He has served on a number of programme committees, and is currently on the IEEE ICC-2007 local organizing committee. He has been institutional lead on both European and UK funded projects. He was a founding member of the UK-wide FORCES project to conduct collaborative research on service creation. He is currently co-investigator on PROSEN; a project across 4 UK universities investigating wireless sensor networks. He is also closely involved with MATCH; a project across 4 Scottish universities investigating technologies for care in the home. He has been awarded external funding for research on communications systems from companies such as British Telecom, Edinburgh Network Technologies, GPT, Panasonic, and SysNet. His interests include data, voice, multimedia, and home networks. His career spans both university and industry, and, Computing Science and Electrical Engineering.