Handling Distributed Feature Interactions in Enterprise SIP Application Servers

M. Kolberg[1], J. F. Buford[2], K. Dhara[2], X. Wu[2], V. Krishnaswamy[2]

[1]University of Stirling, Stirling, Scotland, UK
[2]Avaya Labs Research, Lincroft, NJ 07738, USA
mko@cs.stir.ac.uk , {buford,dhara,xwu,venky}@avaya.com

Abstract

Several trends in SIP application server deployments exacerbate the classic problem of feature interaction in large enterprise telephony environments: use of distributed feature servers, mixing of legacy and green-field feature servers, and the co-existence of multiple third-party feature implementations provisioned in the same environment. Next-generation SIP application servers will include an application router (AR) to provide more control over feature sequencing. As we discuss here, the AR can be augmented to incorporate feature interaction detection and resolution logic.

We describe a novel design for run-time feature interaction detection and resolution in an environment of distributed feature servers using a SIP application server with application routing function, such as that defined in JSR 289. The approach is based on the algorithm of the Kolberg-Magill (K-M) method for feature interaction detection. Here we extend the notation of the algorithm to cover advanced call control services, enable the algorithm to work in topologies involving B2BUAs and SBCs, and test the approach with a substantial feature set of 32 features.

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 Wu and Schulzrinne [1], when SIP services interwork their combined behavior may not be acceptable. This phenomenon is known as feature interaction (FI) or service interaction [2].

1.1 Feature Interaction

When services interwork to share communication resources, they are *compatible* if the joint behavior of the resource is acceptable. However, if the joint behavior 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 behavior of a resource under the joint control of interworking services.

Some previous studies distinguish the concepts of features and services. However, for this paper, the distinction between feature and service is not significant, what is crucial is the concept of *interaction*. Here, the terms *service* and *feature* are used interchangeably.

In large enterprise deployments, multiple application servers are typically used both for scaling and geographic span. This makes it more difficult to use techniques which involve centralized monitoring and control of calls. Further, the mixing of legacy and green-field feature servers is a practical solution to system evolution without forklift replacement, but also limits the use of centralized methods. Finally, open platform deployments mean that multiple third-party feature implementations can be provisioned in the same environment. Competing vendors are not likely to disclose the workings of their implementations, a pre-requisite for offline methods. For these reasons we require a run-time approach which works in a distributed feature server environment.

Further, the method described here is suitable to IMS systems and highly distributed feature implementations that might be implemented in the future in peer-to-peer overlays. We describe such an architecture in [14]

1.2 Contributions

Very few existing FI approaches are suitable for distributed architectures, even fewer are suitable for Application Servers implementing features using SIP Servlets [3],[4]. Here we present the first approach which works successfully in distributed SIP Application Servers which also addresses a number of crucial topology issues in enterprise SIP.

This approach can be used in conjunction with techniques used inside a single Application Router, such as the DFC[5] based Application Router (AR) [8],[9]. While approaches focusing on a single AR cannot handle interactions between services deployed on separate Application Servers, our the approach uses SIP messages to deliver information on services between Application Servers. However, the approach can also be used to check services for feature interactions within an Application Router.

We extend the notation of existing Kolberg-Magill (K-M) [11][13] approach by introducing a notation for conference services and bridged appearances. We also show how the approach operates across SIP components commonly found in enterprise installations which prevent the passing on of privacy relevant information, such as B2BUAs and SBCs (Session Border Controller). Finally, the correct operation of the approach is verified with an extensive case study involving 32 services.

1.3 Related Work

1.3.1 Feature Interactions

A substantial body of work exists on dealing with feature interactions through the regular series of Feature Interaction Workshops. Approaches can be categorized as being either off-line or on-line. Off-line approaches are applicable prior to deployment of the services during requirements capturing or the design phase whereas on-line approaches are applied during testing or post-deployment at run-time of the services. Approaches are discussed in detail in [6] and [7].

The work in this paper is particularly related to approaches which order services for execution allowing for increased interworking and re-use. The Distributed Feature Composition (DFC) [5] approach is the basis for the JSR 289 Application Router [8],[9] design. DFC is not strictly a FI detection mechanism but rather a paradigm for feature modularity and sequencing which avoids feature interactions. Thus DFC is complementary to the method described here. The SIP Steplets approach by Kocan et al [10] has been applied to IMS. However, both approaches are essentially designed for a single domain, in which they have the knowledge of all the feature history information, but lack a way to carry feature history information across domains. Our approach can work across multiple domains.

This paper builds on work that had already been successfully applied in a traditional telephony setting [12] and a *basic* SIP environment [13]. These designs in turn were based on a pragmatic approach [10].

1.3.2 JSR 289

JSR 289 [4] is the next version of the SIP Servlet specification which revises JSR 116 [3]. The main architectural difference between JSR 289 and JSR 116 is a new application selection and composition model. The key component is a logical entity called *Application Router (AR).* Logically, the AR is separated from the application container. It only handles application selection and routing, but not application logic. Separating composition from applications allows the application developers to focus on the logic of applications. It promotes modularity and re-use, and allows application development and sequencing to be controlled by configuration and policy settings.

To compose applications, the application router should be aware of the intention of applications when it sends a request. The approaches defined in this paper can help an application router detect potential feature interactions based on the intention. To handle feature interaction for inter-container application routing, the first application router should pass feature information to the second application router. JSR 289 suggests passing call state information in SIP Route headers, however, the state information is not sufficient for detecting feature interactions.

This paper uses a private SIP header called *P-ConType*, which carries application-specific information from one application router to another and facilitates feature interaction detection in two or more application routers. Alternately, the necessary feature state can be propagated in the body of SIP messages using MIME encoding. The main considerations include compliance with SIP specifications, the ability for different implementations in different domains to interoperate, and increasing the likelihood that the information will be passed by various signaling elements.

We discuss how to implement our approach in the JSR 289 framework in detail in Section 2.

1.3.3 Feature Interaction Approach

The K-M approach uses high-level compact service descriptions which are inserted in the SIP message after a service has been active on the call. Before subsequent services are activated on a call, an algorithm using simple rules checks each service description carried in the SIP message against the description of the service to be activated. If no interaction is detected, the service is activated as usual. However, if an interaction is detected, only one of the involved services is allowed to be active on the call. This might be achieved by simply disallowing the second service, or by repeating the call such that the first service is prevented from being activated on the call.

The approach describes the behavior of a service with the triggering party and a connection type. The latter consists of the connection to be set up before the service is activated, and the connection set up after the service has been triggered. Call Forwarding Unconditional (CFU), which redirects all incoming calls to a predefined third user, can be described as follows. Assume party A is the originator, B the terminator, and C the party where the call is redirected to. 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 connection type, `(A, B)` $\rightarrow$ `(A, C)`, `(A, B)` is the 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. The call starts with A attempting to connect to B. However, due to CFU, A is connected to C instead.

Interaction cases are found by analyzing pairs of services. Two service descriptions are checked against five rules. If a service pair fulfills any of the five rules, then the pair is said to interact. The rules cover the following behaviors:

- Single User – Dual Feature Control
- Connection Looping
- Redirection and Treatment
- Diversion and Reversing
- Treatment and subsequent Missed Call Handling

For brevity the rules are not repeated here, but can be studied in [11][13].

1.3.4 The Approach in SIP

The P-ConType Header

The distributed nature of the approach helps its application to SIP. Each service which gets activated includes its description into the SIP 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 makes the approach highly scalable.

For SIP, additional headers carrying the required information have been defined and can be included with the SIP messages. Two private headers have been defined to carry the required information for this approach: **P-ConType** and **P-Forwarded-To**. The P-ConType header contains the descriptions of services which have been active on the current session. The P-Forwarded-To header contains the ID for an invited party when an INVITE request is redirected to another party. As is discussed in [12],[13] standard SIP headers do not provide sufficient detail for this.

During feature sequencing, the current SIP message is checked for the P-ConType header. If no such a header is found, no other service has previously been active and hence a service interaction cannot have occurred. In this case, a new P-ConType header is inserted into the message describing the current service. For instance, for a forwarding service the header is depicted below.

```
P-ConType: ID=Forward; TP=sip:bob@d254203.com;
OrigFrom=chris@discus.com;OrigTo=bob@d254203.com;
FinalFrom=chris@discus.com;FinalTo=alice@d254203.com
```

The header contains the ID field, triggering party and the connection type. The ID identifies the service described in the header. The TP contains the triggering party, and the remaining four fields correspond to the four fields of the connection type.

2. Implementation in a JSR 289 SIP AS

In the JSR 289 application selection and composition model, there are three routing regions, namely originating, terminating, and neutral. If an application serves the subscriber as caller, it is in the originating region. If it serves the subscriber as callee, it is in the terminating region. If it is invoked without a specific subscriber, it is in the neutral region. There are also three routing directives: *NEW*, *CONTINUE*, and *REVERSE*. The *NEW* directive handles initial requests, the *CONTINUE* directive handles relayed requests, and the *REVERSE* directive can change the routing region of applications. A more detailed description of routing regions and routing directives can be found in JSR 289 [4]. To be compatible with JSR 289, our notations and P-ConType header must be able to represent these concepts. Table 1 shows the mapping for a subscriber A with applications in different routing regions under different routing directives. As an application in neutral region does not have a specific subscriber, it is not listed in the table.

	NEW	CONTINUE	REVERSE
Originating	OrigFrom = A; OrigTo = B; FinalFrom = A; FinalTo = B;	OrigFrom = A; OrigTo = B; FinalFrom = A; FinalTo = B;	OrigFrom = A; OrigTo = B; FinalFrom = B; FinalTo = A;
Terminating	OrigFrom = B; OrigTo = A; FinalFrom = B; FinalTo = A;	OrigFrom = B; OrigTo = A; FinalFrom = B; FinalTo = A;	OrigFrom = A; OrigTo = B; FinalFrom = A; FinalTo = B;

For applications residing in the same container, the application router for that container can keep track of the invoked applications and check feature interactions based on the rules defined in Section 3. Under this circumstance, it is not necessary to use P-ConType header to carry service history information. This also allows the algorithm examine together all services deployed on a single Application Server which are selected for a call. As this will occur prior to service execution, it will help with the resolution of interaction and also with the detection of Missed Trigger Interactions [12] and can also influence the execution order decision made by the Application Router.

However, once a request goes to another container, it must carry all the service history in the P-ConType header. The P-ConType header may be composed by the application router when it handles the last application in its associated container. An alternative and more consistent way would be having each application compose the P-ConType header, as we described in Section 3.3.6. The application router then does not have to keep track of application invocation history, instead, it simply processes the P-ConType header of each request and detects feature interactions.

An architecture integrating the P-ConType header processing with the Application Router is depicted in Figure 1. This shows the P-ConType header processing being linked to the Application Router. Before the Application Router decides on a final sequence of services to be executed for a particular request, it consults the P-ConType header processing with a preliminary list of services. The P-ConType header processing engine will then detect interactions among that set of services and between services in the set and any previous services active on the call.

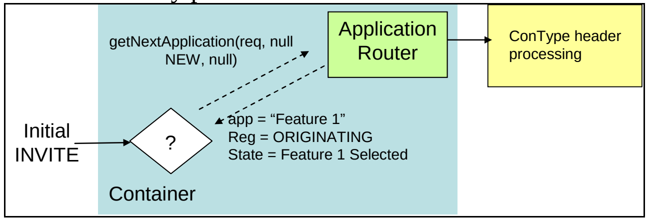

Figure 1: Integration of the P-ConType header processing into the SIP AS application router.

2.1 Topology Issues

As the K-M approach relies on information being transmitted in the SIP messages between SIP components, its operation with SIP servers which restrict the information passed through them (e.g. B2BUAs) require careful consideration.

In the following specific examples are examined including transparent B2BUA, monitoring B2BUA, and B2BUA acting as session controller.

2.1.1 Transparent B2BUA

There are two cases for Transparent B2BUAs: Firstly they may carry the P-ConType header forward as specified, and also are able to send back the disabling of a feature due to an interaction. There is no need altering information in headers.

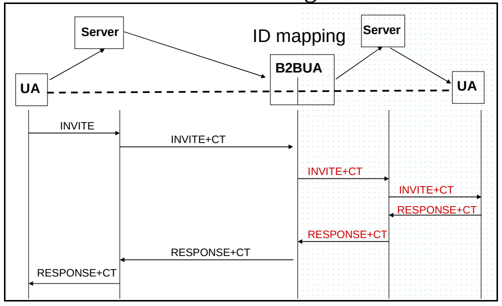

Figure 2: Identity mapping with transparent B2BUAs.

In the second case, B2BUAs modify information in some headers which could impact on the feature interaction approach. For instance, by changing the identity of the endpoints through changes in the From/To/RequestURI, the mapping between those headers and the information contained in the P-ConType header is broken. Furthermore, the P-ConType header may still reveal the 'previous' identity of the parties. Hence the B2BUA needs to perform the same address mapping on the values in the P-ConType header as in the altered SIP headers. This mapping should happen for both upstream and downstream messages. The mapping is illustrated in Figure 2.

In the case of a chain of transparent B2BUAs along a signaling path, the mapping occurs at each B2BUA. Hence the behavior if chained B2BUAs can be seen as a sequence of the single case.

2.1.2 Monitoring B2BUA

Monitoring of a session can either be invisible (e.g. through a feature such as Lawful Intercept) or visible (through a feature such as Session Recording). "Invisible" monitoring should not be detectable by other endpoints in the call. Hence the signaling from the monitoring endpoint needs to be hidden from the other endpoints. B2BUAs can be used to achieve this. But clearly there are privacy issues here which might be compromised by the P-ConType header. A number of different cases of monitoring can be distinguished:

1) Monitoring is invisible with higher priority than the monitored call. In this scenario, features such as Lawful Intercept or Supervisor Monitoring should have priority over any feature interaction issues. That is, the monitoring should stay invisible even though this means that some interactions due to the monitoring are not dealt with. An example of such a scenario is when the monitoring party is on the screening list of a party on the monitored call.

   In such scenarios, P-ConType headers from features for the monitoring party are not to be sent to other parties the call, and the call setup should *never* be repeated due to a feature interaction (disabling one of the features) as this can be detected at the other endpoints and reveal the monitoring. Instead, such an interaction is resolved by giving priority to the features of the monitoring party.
2) Monitoring party is invisible with equal or lower priority than the monitored call. An example for this scenario is if monitoring is active on call. Then the CEO with a feature disallowing monitoring of calls joins the call. In this case monitoring should then be disabled.
3) Monitor is visible. In this case, the privacy issues do not apply and the P-ConType header can be included into the messages as normal. Also feature interaction resolution operates as defined in Section 3. Interaction within call legs which have the B2BUA as origin or termination point need resolving at the B2BUA. However, there could be feature interactions across the call legs of a multiparty call that cannot be made consistent. Interactions in such a scenario are analyzed asymmetrically to the different legs.

2.1.3 B2BUA is Session Border Controller

A Session Border Controller (SBC) main purpose is to hide domain routing and endpoint identities from external endpoints and signaling elements. Clearly, the goals of the SBC and the feature interaction approach conflict. A SBC will not forward information in the P-ConType header as this might reveal identities and features used by those identities. An example topology involving a SBC is shown in Figure 3.

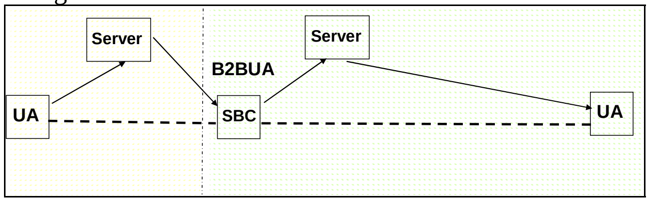

Figure 3: Topology with Session Border Controller.

However, feature interaction analysis within one domain is still possible by isolating the feature interaction logic within each domain. While this will resolve interactions between services used within one domain, it will not capture interactions involving services from different domains.
Alternatively, the SBC could map feature interaction feedback in a way that does not disclose the internal topology or signaling. For instance, there could be a list of hidden features that are filtered out of the P-ConType to prevent visibility outside the domain. Or only public endpoints are visible outside the domain. All these approaches will impact on the ability to handle some interactions, for the benefit of increased the privacy. The exact policy, e.g. are all P-ConType headers removed, or only some, and is a feature interaction handling within the local domain implemented, depends very much on the privacy requirements of a particular domain and should be configurable.

3. Experimentation and Results

Previously, we have experimented with the approach using nine common call control services. Here we have significantly extended the experimental services set to 32 services. Some services required extensions to the approach which are discussed below.

3.1 Extensions to the K-M Approach

So far the K-M approach did not cover multi-party calls and bridged appearances as well as priority calls. In the following notation for these features is introduced.

3.1.1 Multiparty Calls

To allow the approach to cover three-party calls, the notation has been extended. A multi-party call (join) is represented as:

```
ConfJoin: TP: A; {A,C} A,B → A,B,C
```

Here, party A is already in a call with party C and has currently placed that call on hold (curly brackets). A now calls B and after activating the join conference feature, the two calls are joined together to form a three-party conference.

With this extension of notation, the 5 rules are affected as follows. Rule 1 is applied unchanged. The held call (curly brackets) is not included in the check for an interaction.

Considering Rule 2 (loop), the final connection A,B,C does not conflict with any forwarding feature. Consider the following example:

```
ConfJoin: TP: A; {A,C} A,B → A,B,C
CFU: TP: C; A,C → A,B
```

Here, C has a forwarding feature for all calls to B. However, there is no loop being created during the conference join. Similarly, if B had a forwarding feature to C, no interaction would occur, A would simply get Busy tone as A is already connected to C.

```
CFU: TP: B; A,B → A,C
```

Hence, a multi-party call connection A,B,C does not cause a loop with other two party call features.

Rule 3 involves checking all possible pairwise connection against the other feature. Consider the following example:

```
ConfJoin: TP: A; {A,C} A,B → A,B,C
TCS: TP: B; C,B → C,TREAT
```

Here, a connection between C and B is disallowed by the TCS service. However, as part of the conference, C and B will be connected, arguably violating the TCS service. However, this interaction will be detected if each pairwise connection from the multiparty call is checked against the other feature.

Rule 4 is applied unchanged. This rule captures forwarded calls which are subsequently reversed. Consider the following scenario:

```
ConfJoin: TP: A; {A,C} A,B → A,B,C
AR :   TP: B; A,B → B,A
```

Multiparty calls do not fall into this category. Hence Rule 4 and does not apply to this scenario.

3.1.2 Bridged Appearances

Bridged Appearance (BA) is a common service in enterprise environments, for instance between an executive and their secretary. In the approach BA behavior can be described as

```
BA: TP: B; A,B → A,B-C
```

Here A phones B, but B is on a bridged appearance with C, so A gets connected to B with C also connected. This scenario has the BA on the terminating end of the call. The scenario below shows BA on the originating side of the call.

```
BA: TP: B; B,A → B-C, A
```

With this new notation, Rule 1 is not changed. Similarly, Rule 2 is applied as before. The BA connection (e.g. B-C) below, is not considered a connection in itself. Consequently, Rule 2 does not apply to the example below.

```
CFU: TP: C; B,C → B,A
BA:  TP: B; B,A → B-C, A
```

Rule 3 needs to consider all the parties involved in a call. For instance, in the scenario below, C is not allowed to call and be connected to A. However, if B phones A with B having C on a BA, the TCS feature is violated. Hence, Rule 3 needs to consider parties on BA.

```
TCS: TP: A; C,A → C,TREAT
BA:   TP: B; B,A → B-C, A
```

Applying Rule 4 to a pair of features involving BA does not yield any interaction cases. Considering the scenario below, the called returned by the AR feature does not conflict with the BA feature.

```
BA:  TP: A; A,C → A-B,C
AR : TP: C; A,C → C,A
```

In other words, the BA connection is not considered as a forwarding of the call.

3.1.3 Detection and Resolution of Feature Interaction

In previous work [13] we advocated an approach in which detection and resolution occurred only after both features were executed and triggered. In some circumstances, the resolution then required ‘undoing’ an executed service. Clearly this is problematic with services changing state or communicating with other resources. Hence the current design employs pre-activation detection.

When a message arrives at a SIP component, the message is checked for a P-ConType header. If a P-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 executing the five rules discussed in the previous section. If no service interaction is detected, and no further P-ConType header is found in the message, the message is processed by the to the servlet and subsequently, the P-ConType header for the current service is inserted into the message and the message is sent on to its destination.

If an interaction is detected, the outcome of one of the two services involved needs to be discarded. If the actions of the second service are to be discarded, the service is simply not carried out. If the actions of the first service are to be discarded, the session setup might need to be repeated, but only if the second service actually gets triggered (depend on time of day, other service data). Hence the second service will be executed. If it does not get triggered, there is no interaction and the call proceeds as normal with just the P-ConType header of the first service. However, if the second service gets triggered, an interaction would occur and hence the call attempt needs to be repeated – disabling the first service (see Figure 4).

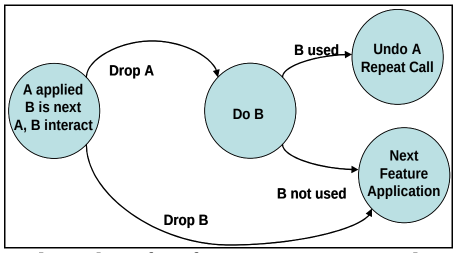

Figure 4: Algorithm for feature interaction detection and resolution for features A and B.

A message indicating this is sent back to the originator (a SIP Response 380 Alternative Service). The P-ConType header for the service to be disabled is extended by the field Status=disabled. The UA client will receive the message and issue a new Invite request, again with the P-ConType header and flag copied in. When this request is received by the service which the P-ConType header matches, it will not trigger the service.

3.2 Results

Table 1 provides details of the 32 features in the experimental set. As with the previous case study, a number of features have the same descriptions even though they are quite different. This is due to the abstract nature of the approach, and has been discussed previously [13]. Features from previous case studies have been marked with a grey background. Here, the features have been grouped according to their description. Each group has been giving a name: Originating Call Setup (OCSet), Originating Call Filtering (OCFil), Terminating Call Establishment (TCEst), Terminating Call Diversion (TCDiv), Terminating Call Filtering (TCFil), Terminating Call Reversing (TCRev), Service Call Features (SC), and Call Conference (Conf) and Bridged Appearance (BA).

Clearly, the interactions for features within a group are identical. Hence these feature groups can then be used to show interactions between the different groups (Table 2). Even though the table suggests interactions between many feature groups, this does not necessarily translate to interactions in all calls involving features from two incompatible categories.

Table 1: Modeling and Grouping of Services.

#	Feature Group		Feature	Description in Notation
1	OCSet	HL	Hotline	TP:A; A,B → A,B
2		LCR	Last Call Return	TP:A; A,B → A,B
3		PA	Paging	TP:A; A,B → A,B
4		LND	Last Number Dialed	TP:A; A,B → A,B
5		MI	Manual Intercom	TP:A; A,B → A,B
6		SR	Save and Redial	TP:A; A,B → A,B
7	OCFil	OCS	Call Managemet (Outgoing)	TP:A; A,B → A,TREAT
8	TCEst	CW	Call Waiting	TP:B; A,B → A,B
9		ACB	Automatic Callback	TP:B; A,B → A,B
10		CON	Camp-On	TP:B; A,B → A,B
11	TCDiv	CFU	Call Forwarding Uncond.	TP:B; A,B → A,C
12		CFB	Call Forwarding Busy	TP:B; A,B → A,C
13		CFNA	Call Forward No Answer	TP:B; A,B → A,C
14		CFFMe	Call Forward Follow Me	TP:B; A,B → A,C
15		GR	Group Ringing	TP:B; A,B → A,C
16		CFOP	Call Forward Off-Premises	TP:B; A,B → A,C
17		CFR	Call Forward Ringing	TP:B; A,B → A,C
18		CT	Call Transfer	TP:B; A,B → A,C
19		SAC	Send All Calls	TP:B; A,B → A,C
20		COV	Coverage	TP:B; A,B → A,C
21		HG	Hunt Group	TP:B; A,B → A,C
22	TCFil	TCS	Terminating Call Screening	TP:B; A,B → A,TREAT
23		VMS	Voice Mail	TP:B; A,B → A,TREAT
24		DND	Do-Not-Disturb	TP:B; A,B → A,TREAT
25		SCR	Selective Call Rejection	TP:B; A,B → A,TREAT
26		CB	Call Block	TP:B; A,B → A,TREAT
27		ACR	Anonymous Call Rejection	TP:B; A,B → A,TREAT
28	TCRev	AR	Automatic Ringback	TP:B; A,B → B,A
29	SC	WAK	Hotel-wake up	TP:Treat; A,Treat → Treat,A
30		REM	Reminder	TP:Treat; A,Treat → Treat,A
31	Conf	Conf	Conference Join	TP:A; {A,C} A,B → A,B,C
32	BA	BA	Bridged Appearance	TP:B; A,B → A,B-C

For an interaction to occur, the exact configuration is crucial. Many features use feature data and are only triggered based on them, e.g. screening lists. Some features are only triggered in certain conditions (party busy, no answer). Thus only calls which meet all these conditions actually lead to FI.

Table 2: Interactions between Feature Groups.

	OCSet	OCFil	TCEst	TCDiv	TCFil	TCRev	SC	Conf	BA
OCSet	1	1,3			3			1	1
OCFil		1		3		3		1,3	1,3
TCEst			1	1	1,3	1			1
TCDiv				1,2	1,3	1,4	4		
TCFil					1	1,3	4	3	1,3
TCRev						1,2	4		
SC							1		
Conf								1	1
BA									1

In a previous study looking at features deployed on SIP UAs and Proxy Servers, we found that some interactions could not be detected when the features are deployed on the same component. This affected features which are triggered by an INVITE request, but one (or both) service drops the INVITE and generates a response message (e.g. Terminating Call Screening). In this case an interaction could only be detected if the service dropping the INVITE is triggered second. If the service dropping the INVITE is triggered first, the second service did not get triggered at all and hence the feature interaction algorithm was not executed. This issue is resolved by the combination of the interaction approach with the Application Router. As the interaction algorithm can be applied before any services have been executed (based on the service selection by the Container), the algorithm will be applied to all service pairs. This applies to Missed Trigger Interactions [12] generally, as long as the services are deployed on the same Application Server.

4. Conclusions

This paper prsents a novel feature interaction approach operating on distributed SIP application servers with application routing based on JSR 289. Besides handling interactions, the approach can also be used to influence the decision of the Application Router on the order in which services are to be executed.

The Application Server environment provides the potential to apply the algorithm to all services deployed on a single Application Server before any of them are executed. This helps with the detection of Missed Trigger Interactions.

This paper considers the application of the approach in certain restrictive topologies including B2BUAs. As the approach relies on certain information to be transmitted in the SIP messages, components which map or filter information contained in SIP messages are an issue. The paper shows how the approach can operate when such components are in the signaling path.

The notation of the approach includes complex call control services such as conference and bridged appearance services. The approach has been validated against a large case study involving 32 common services.

REFERENCES

[1] X. Wu and H. Schulzrinne. Handling Feature Interactions in the Language for End System Services, *Computer Networks*, Volume 51(2), pp.515-535, February 2007.

[2] M. Calder, M. Kolberg, E.H. Magill and S. Reiff-Marganiec. Feature Interaction: A Critical Review and Considered Forecast, *Computer Networks*, Elsevier Science, Vol. 41, No. 1, pp. 115-141, 2003.

[3] JSR 116: SIP Servlet API V1.0, http://jcp.org/en/jsr/detail?id=116

[4] JSR 289: SIP Servlet API V1.1, http://jcp.org/en/jsr/detail?id=289

[5] M. Jackson, P. Zave. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Transactions on Software Engineering* XXIV(10):831-847, October 1998.

[6] M. Calder, M. Kolberg, E. H. Magill and S. Reiff-Marganiec. Feature Interaction: A critical Review and Considered Forecast, Computer Networks J., Vol. 41(1), 2003, pp. 115-141.

[7] D. O. Keck and P. J. Kuehn. The Feature and Service Interaction Problem in Telecommunications Systems: A Survey, IEEE Transactions on Software Engineering, Vol. 24(10), pp. 779—796.

[8] E. Cheung and K.H. Purdy. Application Composition in the SIP Servlet Environment, IEEE International Conference on Communications (ICC) 2007, Glasgow, UK.

[9] E. Cheung and K.H. Purdy. An Application Router for SIP Servlet Application Composition, IEEE International Conference on Communications (ICC) 2008, Beijing, China.

[10] K.F. Kocan, W.D. Roome, and V. Anupam. A Novel Software Approach for Service Brokering in Advanced Service Architecture, Bell Labs Technical Journal, Vol. 11(1), pp. 5-20, 2006.

[11] M. Kolberg and E.H. Magill. A pragmatic approach to Service Interaction Filtering between Call Control Services, *Computer Networks J.,* Elsevier Science, Vol. 38, pp. 591-602, 2002.

[12] M. Calder, M. Kolberg, E.H. Magill, D. Marples and S. Reiff-Marganiec. Hybrid Solutions to the Feature Interaction Problem, In D. Amyot and L. Logrippo, *Feature Interaction in Telecommunications and Software Systems VII*, IOS Press, Amsterdam, pp. 295-312, 2003.

[13] M. Kolberg and E.H. Magill. Managing Feature Interactions between Distributed SIP Call Control Services, *Computer Networks J.*, Elsevier Science, Volume 51, Issue 2, 7 February 2007, pp. 536-557.

[14] *J. Zhou, J. Buford, K. Dhara, X. Wu, M. Kolberg. Discovery and Composition of Communication Services in Peer-to-Peer Overlays. IEEE Workshop on Service Discovery and Composition in Ubiquitous and Pervasive Environments (SUPE'07). Nov. 2007.*