Feature Interaction in a Federated Communications-Enabled Collaboration Platform

M. Kolberg, J. F. Buford, K. Dhara, X. Wu, V. Krishnaswamy

1University of Stirling, Stirling, Scotland, UK
2Avaya Labs Research, Basking Ridge, NJ 07920, USA
3Fidelity, NY, USA
mko@cs.stir.ac.uk, {buford,dhara,venky}@avaya.com, xiaotaow@gmail.com

ABSTRACT

Existing online collaboration tools and platforms provide basic communications integration and the ability to include some real-time information sources. Moreover, users prefer to be able to choose which collaboration tool they use for a given interaction, and over the course of long-term collaboration, will typically use a variety of tools, including email, instant messages, wikis, blogs, web conferences, and shared documents.

For enterprise use there is a need to integrate the various tools as well as link with existing intelligent communication systems to support long-term collaborations in a variety of ways. Due to the number and different nature of collaboration services used in enterprises today, building such a federated collaboration platform is challenging. Collaboration tools differ in terms of storage model, APIs, content organization, addressing, formats, user authentication, and user interface. By the very nature of such systems they include a large number of independently developed features and services and thus provide a strong potential for feature interactions. This paper presents novel work on feature interaction analysis in collaboration environments and presents an approach to detect and resolve such interactions where the collaboration space is used as a communication endpoint.

In this paper ConnectedSpaces is used as a basis to carry out a detailed analysis of feature interaction problems in collaboration environments. ConnectedSpaces is a new model for federated collaboration environments. Like a number of existing systems, ConnectedSpaces uses a collaboration space as the basic construct. ConnectedSpaces enables the user to work directly in the client application of their choice; this is illustrated with plug-ins for MS Outlook, Internet Explorer and Skype.

This paper presents distinctive characteristics of ConnectedSpaces, including views, spaces as communication endpoints, space persistence and structuring, and embedded objects. Using these features, new types of feature interactions for collaboration platforms are categorized and analyzed. The work presented in this paper focuses on handling feature interactions which are caused by using ConnectedSpaces as a communication endpoint with enterprise communication platforms. This work is novel as it is the first investigation into feature interactions with collaboration platforms. Our approach uses a runtime feature interaction technique which can cope with features being provided by different organizations.

Keywords: Collaboration Tools, Enterprise Communication, Feature Interaction

1. Introduction

Many enterprise collaborations start informally, with a chat, a phone call or an email. Through further communication the idea is then developed. This may involve further emails and calls, and documents which are shared between the collaborators. As the idea matures, additional collaborators might become involved, again through email, collaborative document authoring, or web conferences.

Clearly, this involves a number of tools and software packages from various vendors commonly used in enterprises today. Currently, there is no single-vendor collaboration platform that adequately incorporates all types of communication modes (VoIP, IM, blogging, email, document sharing, wikis, web conferences, etc.). As sharing document between applications is hardly possible, this leads to a fragmentation of information belonging to the same collaboration. Clearly, an integrated view of all the dimensions of a collaboration is important, but so far this process has to be manually managed by the user. This paper presents a federated real-time collaboration platform for the enterprise domain which supports features for the integration of various communications and shared documents belonging to a collaboration.
Unfortunately, when software implementing new functions is introduced into a system, the joint behavior is not always compatible. In software development such a software component of additional functionality is called a feature. Features may be added incrementally, potentially by different developers. Some authors distinguish the terms features and services. However, for the purposes of this paper, the distinction between feature and service is not significant and the two terms are used interchangeably. In telephony, common examples of features are call forwarding, or ring back when free. Example services in collaboration platforms include Space Composition and Spaces as Communication Endpoints. Services in collaboration environments are more complex and their functionality is more diverse than in traditional telephony. Collaboration platforms, like telephony systems, are highly distributed, but unlike in telephony a number of previously separately used services are interworking. In both settings, features or services are commonly developed and tested in isolation.

If different services interwork, and their joint behavior is unacceptable, these services are said to interact. Within feature interactions the terms interworking and incompatibility have well understood meanings. Services must interwork to share a (communications) resource, for instance a collaboration space. The services may interwork explicitly through an exchange of information with each other, or implicitly through changing the space. In the second case, the services often have no knowledge that the other exists.

Importantly, feature interactions are not due to coding errors or wrongly used interfaces or protocols. Rather feature interactions are due to conflicting goals of the services involved or broken assumptions [15]. When services meet for the first time in the network, one service may break assumptions made by the other. Major research challenges include how to predict scenarios in which there is potential for an interaction to occur, how to detect that an interaction does indeed occur, and how to resolve an interaction. In this paper the authors’ previous work in feature interaction [22][14][13][15] is extended by investigating feature interactions in collaboration spaces. A runtime feature interaction approach is employed and it is shown through examples that the approach can be applied to such environments.

As this is the first record of feature interactions analysis in collaboration spaces and the first proposal of an approach to handle feature interactions in such environments, ConnectedSpaces is described as a representative example. ConnectedSpaces is used as a vehicle to illustrate general concepts as well as common collaboration features.

In particular this paper presents the following results:

- A description of features for ConnectedSpaces, including space addressing and nesting, the use of a collaboration space as communication endpoint, space history, temporal control, group management and security.

- The ConnectedSpaces framework is presented for building a scalable collaboration platform and present implemented components of ConnectedSpaces functionality as illustration of key ideas. ConnectedSpaces is integrated with clients including MS Outlook, Internet Explorer, and Skype.

- The proposed feature set is used to investigate feature interactions. Collaboration features are analyzed in terms of feature interactions. New categories of feature interactions are described and organized into a taxonomy of feature interactions for collaboration spaces. The taxonomy includes a detailed description with examples.

- A novel runtime feature interaction handling approach is discussed to capture these feature interactions. Using examples, the paper demonstrates the applicability of the approach.

This paper as organized in the following sections: Section 2 summarizes related work in collaboration systems and feature interaction. As the foundation for the later discussion on feature interactions, Section 3 describes the ConnectedSpaces collaboration platform including its framework and architecture. Section 4 discusses categories of features in collaboration platforms, describes example use cases and discusses new feature interaction problems. Section 5 illustrates the application of a novel feature interaction approach for collaboration platform features. Section 6 concludes this paper.
2. Related Work

A substantial body of work [11]-[20] exists on dealing with feature interactions. Most approaches can be categorized 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, whereas the latter are applicable during testing and deployment.

Offline approaches are often based on the application of formal methods, and as such require considerable information of each individual software increment. In a competitive market environment, this information may not be available. Thus offline approaches suffer from the limited information available on which services will be deployed in the network and hence will interwork. Also, as the number of services increases, the work in analyzing 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, online or runtime approaches consider services as they interwork in the network. However, after detecting an interaction online approaches need to find a resolution to the detected interaction in order to be useful. The results of offline approaches may simply feed back to the design process of the services. Due to the involved complexities, very few online approaches have been proposed in the literature [15]. Most of these are tightly coupled with their target execution environment, be it traditional telephony [30] or SIP networks [14][20]. Consequently, these approaches are not readily applicable to services found in federated collaboration platforms.

Little work has been done on feature interaction in collaboration environments [29]. Design type approaches to feature interactions in web services have been studied by Weiss et al. [7][8][9]. There, rather complex models of web services are built and then manually analyzed to detect interactions. More generally, modeling the behavior of services has been the basis for a number of previous approaches. Some approaches use abstract properties of services, often in a logic. Interaction is expressed in terms of that logic, usually as inconsistency or unsatisfiability. For example, Felty et al.[31] use Linear temporal logic to show inconsistencies between incompatible services. Gibson [32] use First order logic and temporal logic of actions to show invariant violation and nondeterminism as an indication of feature interactions. Frappier et al [33] also use first order logic to show inconsistencies between services. Wu et al [34] consider pre/post conditions of actions to detect interactions between high level services. Interactions between software widgets have been investigated in [22].

In this paper the concepts of pre- and post-conditions are used to capture the behavior of services in collaboration platforms and show how this approach can be integrated with a run-time approach developed by the authors.

Collaboration platforms vary from wikis, blogs, and shared documents to web-based collaboration systems to 3D collaboration spaces offered by virtual worlds. There are many types of tools today, provided by different vendors. For a survey of collaboration platforms, see for example [1]. The metaphor of a shared space or room has been discussed in [23][24]. The integration of communication services into collaboration tools has been described in [25][26][27][28][38]. The use of virtual worlds and environments for collaboration has been introduced in [31][36][37]. Many such environments have combined document sharing, collaborative annotation, white boards, voting mechanisms, calendars, chat windows, and session recording.

Web based collaborations such as wikis, blogs, conferencing systems such as WebEx or Meeting Exchange are used for collaboration in enterprises. While wikis and blogs are used as collaborative authoring tools for a large number of users, other web-based conferencing systems are used to create a space that combines users’ communication links with desk-top application sharing. Typically, these include audio and video conferencing and features such as side-bar, remote-party mute, etc. These systems are based on the notion that there is a common space that is accessed through a browser and users can collaborate in that space.
Microsoft Sharepoint [41] supports collaboration on a set of files or documents. In addition, a Sharepoint site can include a variety of team interaction areas including blog and a wiki. There is no integration of third party collaboration tools, although APIs on both client and server side are available for extending a Sharepoint. IBM Lotus Notes [42] provides email, IM, calendar, personal journal, and contacts. IBM Connections [43] provides collaboration spaces called Communities, blogs, social bookmarking activities, wikis, and file sharing. In Connections, a shared space called an Activity is used to integrate related documents, emails, and IMs. In addition, through integration with Google Desktop Server, an Activity can reference documents on the local file system. Connections integrates with existing applications through plug-ins including Lotus Notes, MS Office, MS Outlook, and MS Sharepoint. Google Apps (google.com/apps) is a set of web-based applications which include some collaboration capabilities. These tools include Gmail, Google Groups, Calendar, Talk, Docs, and Sites.

There is another set of collaboration platforms that are based on virtual worlds, such as Second Life (www.secondlife.com), Kaneva (www.kaneva.com), and There.com. These virtual worlds offer features such as immediacy (real-time interaction), interaction (ability to view, create, modify, and act) on a shared space that is closer to replicating reality. While these platforms offer rich user-experiences, often the creation of collaboration spaces and the navigation in those spaces is not easy. The taxonomy of networked virtual environments [2] discusses the need for designing a network architecture within virtual environments. Broll et al. [3] defined an approach for inter-world communication. Bouras et al. [4] defined an approach for inter-world communication. Bouras et al. [5] proposed a distributed virtual reality networking platform for multi-user interaction. Sallnas [6] provides a comparative study of different modes of communications. All of these efforts improve communication and interaction among users of virtual worlds, but are limited to instant messaging or in-world voice. In this paper novel concepts are proposed for integrating enterprise communications in collaboration platforms that is mixing in-world (virtual) communication with real world enterprise communication systems. VoIP based services provided by Vivox (www.vivox.com) offer communication within virtual worlds, but are not enterprise grade with respect to their scope and their features.

A growing number of collaboration tools are leveraging cloud-based resources. These include cloud shared storage services such as Dropbox (www.dropbox.com), GoogleDrive (drive.google.com), Microsoft SkyDrive (skydrive.live.com), Egnyte (www.egnyte.com), and Box (www.box.com). The ability to operate both from the desktop and from a mobile device is also important, such as project coordination applications Soonr (www.soonr.com) and Trello (www.trello.com), and shared task management application Asana (www.asana.com).

3. ConnectedSpaces

3.1 Motivation and Goals

Today's collaboration tools are powerful and widely used. Nevertheless the following observations can be made:

- The need for better integration of intelligent communication capability with collaboration environments.
- The value of simplifying the creation and initialization of new collaborations.
- The importance of being able to structure collaborations and treat them as persistent and externally reference-able, since enterprise collaborations are often long-term, deal with complex information, and are important to document.

To achieve these goals, our ConnectedSpaces framework uses increased automation, meta (view) mechanisms, integration with external information and communication resources, and semantic processing where feasible. The following table summarizes key concepts in our collaboration model.
Table 1 Concepts in ConnectedSpaces

<table>
<thead>
<tr>
<th>Concept</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>space (collaboration space)</td>
<td>A collaboration space provides a shared persistent container in which users to perform collaboration activities. It requires resources, such as computation, communication, and storage devices, to support those activities. For example, Google Wave, Microsoft Sharepoint, and many virtual worlds, such as the Second Life, are all collaboration spaces.</td>
</tr>
<tr>
<td>view</td>
<td>A view of a shared space is a user, a group, or a project specific meta perspective of the collaboration space that itself can be shared, annotated, analyzed, and stored for further retrieval.</td>
</tr>
<tr>
<td>entity</td>
<td>An agent that can view and modify the space and its attributes. Entities are also referred to as members of a space. Each entity has a unique identifier.</td>
</tr>
<tr>
<td>contact</td>
<td>Any entity which a given user may share a space with</td>
</tr>
<tr>
<td>user</td>
<td>A human entity</td>
</tr>
<tr>
<td>robot</td>
<td>A system owned entity that can automatically perform some actions in the space.</td>
</tr>
<tr>
<td>avatar</td>
<td>The representation of an entity in a space</td>
</tr>
<tr>
<td>object</td>
<td>A component embedded in a space that users and robots can operate on. It can be system created or created by users. Objects include content, gadgets, real-time information sources, other spaces, and gateways to components of other collaboration platforms.</td>
</tr>
<tr>
<td>gadget</td>
<td>An object that contains application logic that may affect other entities or communicate with applications outside of the collaboration space.</td>
</tr>
<tr>
<td>application</td>
<td>A collaboration application is used to provide certain functions to manipulate entities in a collaboration space.</td>
</tr>
<tr>
<td>event</td>
<td>An event driven collaboration space uses events to notify one entity about the system and other entities’ states and activities.</td>
</tr>
<tr>
<td>session</td>
<td>A collection of collaboration activities among users, robots, and objects. It spans a certain period of time, contains some specific semantic information, and requires resources, such as communication channels, storage, and network bandwidth, to support the collaboration activities.</td>
</tr>
<tr>
<td>template</td>
<td>A pre-initialized set of objects that can be inserted into a space that provide a pattern for some collaboration activity.</td>
</tr>
<tr>
<td>policy</td>
<td>A rule specified by the entities managing a space and enforced by the collaboration framework which specifies constraints on sharing and accessing the space and its objects.</td>
</tr>
</tbody>
</table>

3.2 Enterprise Collaboration Model

As shown in Figure 1, a collaboration space in ConnectedSpaces is represented in three dimensions: resources, time, and semantics. Each object in the collaboration space uses some resources, spans a certain period of time (life cycle of the entity), and has certain semantic properties (either pre-defined or dynamically updated).
Each space has one or more entities which are members of the collaboration. Each entity has a unique identity. Entities can be organized in groups, and groups can be members of a collaboration space. Identities of entities are managed by the collaboration system. In this paper system owned entities are called collaboration robots or simply robots. In the collaboration space, there can also be sharable objects that member entities space can operate on, such as documents and images [39].

Spaces can be nested, and as in Figure 1, a session can include or refer to another session. There can be session specific robots and objects. The life cycle of session owned robots and objects is within the session. This is similar to Google Wave, where the ‘wavebot’ is session owned robot. A wavebot becomes active only if a user invites it to a session. Robots and objects can also be independent of sessions. A robot may associate with a specific user. For example, a user may have an assistant robot to help her manage her sessions, such as preparing documents, automatically creating a session and inviting her to join, and recording the session.

An important concept in the collaboration space is session. A session represents a collection of collaboration activities among users, robots, and objects within a space. It spans a certain period of time, contains some specific semantic information, and requires resources, such as communication channels, storage, and network bandwidth, to support the collaboration activities.

Outside of the space, there can be applications that can manipulate objects in the space or provide collaboration channels. For example, call routing functions can be considered as collaboration applications. Embedded communications widgets [10] are examples of such applications. In addition, the manipulation of user preferences and policies about appropriate collaboration behavior in space can also be considered as collaboration applications. Such policies, preferences, and the history of the collaboration activity information can be saved in database for later mining by analytical functions.

### 3.3 Client Application Integration

Each application provides an API or REST interface to support major operations (push, pull, search, dereference, tag). An application server runs mediation services which in turn invoke custom APIs or REST interfaces on either the client application on the desktop or the server running in the enterprise. Each service on the app server provides a common API to simplify the client side integration. The API offers an interface to create, clone and edit spaces, as well accessing the contents of a space. Using a mediation service rather than going directly from each client reduces the complexity of client integration. The mediation server can also act as content repository for services that might be offline or inaccessible in the future. When a federated client needs to access content in another collaboration app, it issues the requested operation (push, pull, search, tag) to the corresponding application or server. Authentication credentials can be pre-stored at the application server or dynamically requested from the user.

Some externally hosted applications such as RSS Feeds or web conferences can be also accessed via this architecture. An RSS feed is an external data source and is not receiving data from the enterprise. A web conference receives specific types of content (presentations, shared apps or shared desktops) and is designed specifically for use outside the enterprise network.

Three client add-ins were implemented for testing the federated functionality, for Skype, MS Outlook and Internet Explorer. The MS Outlook and Internet Explorer add-ins are shown in Figure 2. A task pane shows the selected email or email thread which is to be pushed to a selected collaboration tool. The lower half of the task pane allows the user to select the destination, and in the case of browse-able repositories like wikis and Sharepoints, to navigate to the specific destination folder of interest. Additionally, a ribbon extension provides icons for launching other collaboration tools. The main features of this design are to support easy push of email content to other collaboration tools, to manage meeting schedules through the calendar, and to be able to quickly launch the other collaboration tool if needed.
The top pane of the Internet Explorer browser is the current collaboration environment, using a web-based UI. Using the URL, the plug-in detects which type of content is being pushed. The lower pane is used to select and navigate to the destination collaboration tool. The destination does not need to be a web-based interface. Alternately a toolbar can be used for selecting the destination and the operation of interest.

The current implementation includes connectivity to Microsoft Exchange, Skype, various components of Microsoft Sharepoint, RSS, Avaya web conferencing, wikis using MediaWiki, and Sociocast-based blogs.

In Skype, the client application extensions run in a separate window, which then uses the Skype API to access the functionality of the Skype client. As shown in Figure 3, chat history can be browsed and then pushed to a specified email or wiki.
4. Key Features in ConnectedSpaces

Currently, features in most collaboration sessions include setting up shared sessions, adding collaboration tools, communication within or outside the space, and managing access controls to the collaboration spaces. To facilitate the discussion in this section on novel features of ConnectedSpaces, the term \textit{space} is used to indicate a collaboration environment with one or more members. This is similar to other systems, which use spaces as containers for collaboration such as TeamRooms [23], shared workspaces [24], media spaces [25], waves, or a shared virtual world space [26] that allows participants to interact with each other.

In the rest of this section, several collaboration features are presented that are important for enterprises and give rise to feature interactions. The categories used in this section are for functional grouping, and the features in one category are not independent and may interact with features in other categories. Importantly, these features may involve independently developed software components which have their own goals and make assumptions about the environment in which they operate. These are two major causes for feature interactions which will be discussed in detail in Section 5.

4.1 Collaboration Views

While setting up sharing in collaboration spaces is essential for enterprises, valuable meeting time is lost in bringing appropriate content to the shared spaces. The persistence of a collaborative space allows instant access to the shared content and a set of commonly used tools. However, they do not address a fundamental issue of view in shared collaboration spaces. A view of a shared space is a user, a group, or a project specific meta perspective of the collaboration space that itself can be shared, annotated, analyzed, and stored for further retrieval. In ConnectedSpaces, there is a notion to instantly bring user specific dynamic context to a collaboration space.

**User-Specific Views:** Based on users’ personal data and preferences, ConnectedSpaces allows an overlay of views to collaboration sessions. An example of such a feature is a gadget or an object in a shared space that presents user specific dynamic data such as their interactions across enterprise data that is not shared with all the participants in the session. This overlay presents appropriate information to a user in their active session.

Figure 4 presents views in their simplest form. The figure depicts a simple collaboration space of an end-user. It depicts the overlay of a user’s collaboration space with two views that contain data mined from user’s data. The first of these views is a relevant contacts view that captures the user’s collaboration context, mines data from user’s previous sessions, email, calendar, and other data sources to present a list of contacts that the user may need during the collaboration session. The second view is a relevant documents view that presents documents that may be useful for the user in the current session. Figure 4 also shows a third personal view that is related to the context of a session. It shows a list of shared colleagues with the remote party of a session.

![Figure 4 Views in a Collaboration Space](image)
While these examples of views are simple, they present two important aspects. One is that views enhance a user’s interaction in a collaboration session. A second aspect is the dynamic nature of views.

**Sharing Views:** With appropriate access control mechanisms and authentication, users can share personal views with other users or with users who are not participating in the collaboration sessions.

This feature could be used as a side-bar between users in a collaboration session. Also, in enterprise collaboration, where access to information and resources is often hierarchical, a manager may wish to share views with a delegate to make appropriate decisions during a collaboration session.

**Managing Views:** Views can be attached to sessions. For views that are dynamic, then robots ensure that they are synchronized appropriately.

### 4.2 Sharing Space and Navigation

Typically, collaboration sessions contain several tools such as a desktop application sharing, document sharing, audio/video conferencing, and the ability to add new tools to shared collaboration spaces. Despite being part of a shared space, these tools are independent. That is, the navigation controls and context of these tools are not visible to the other tools or gadgets in the collaboration space. Users have to work with each of these tools appropriately and try to connect with the context of their collaboration. Some static context such as participants and existing documents can be shared in some collaboration space gadgets, but this notion is not extended to inter-gadget communication or navigation. ConnectedSpaces offers extensions to provide new features that include dynamic exchange of context and navigation in across gadgets in a collaboration space.

**Inter-session communication:** ConnectedSpaces allows objects that communicate with each other during a collaboration session. As an example, consider a collaboration session with a tool (gadget) that handles shared relevant documents. If a new user joins the collaboration session through a communication session, the shared relevant documents gadget automatically updates its content to include documents that relate to the new participant.

**Nested Sessions:** As discussed in the previous section, ConnectedSpaces collaboration sessions can have nested sessions. Within a collaboration session these sessions allow certain users to focus on a particular issue or permit a sub-session that contains confidential data.

**Navigation:** ConnectedSpaces collaboration sessions allow navigation within a gadget or an object to automatically reflect in other objects.

### 4.3 Managing Collaboration Sessions

Apart from the basic management of starting, ending, and storing collaboration sessions, ConnectedSpaces provides additional features that assist user interactions with collaborations sessions.

**Automatic Initiation of Sessions:** Based on the information available on stored sessions, ConnectedSpaces robots allow automatic startup of relevant collaboration sessions. The automatic startup could be in the form of allowing users to start a collaboration session that is related to a topic or that is unrelated but requires a session that is similar to a session on an existing topic. In both cases, the robot in some sense predicts the participants, the gadgets or objects required, and the data required for the collaboration session.

**Session Template:** While storing existing sessions, ConnectedSpaces allows users to save the session template for future use. For example, a user can ask for a collaboration session that could be represented as ‘department collaboration session’. The stored template understands the participants, their capabilities, their context, and starts up a collaboration session with appropriate collaboration space, gadgets, views, and content.
4.4 Collaboration Spaces as Communication Endpoints

In ConnectedSpaces, the space itself represents a communications endpoint. The advantages of such representation are as follows.

- Each communication within a space is part of that space’s content and history.
- Communications capability to all space members is by default integrated in each space without additional effort by the user.
- Different spaces can be used to organize one’s past and future communications.
- Communications to non-members can be provided by embedding specific communications gadgets with those participants.

This means that the space is addressable for communications and that all members of the space are notified of call initiation. Potentially, non-members may also call the space. Addressability can be achieved by associating an identifier in the telephony network with each space. For this purpose, it is assumed that the framework includes or integrates a SIP stack or other call stack, and automatically registers each space with the appropriate registrar.

Each space has a default communications device representation, such as a softphone interface in a 2D space or a 3D representation in a virtual world. This representation is bound to one or more personal communication devices. A member uses their local device representation as the interface. When initiating a call, it can be set up as conference call to all the members of the space, a subset, or external endpoints. Robots which are members of the space can be on calls or initiate calls through the space.

**Example 1:** Alice defines two spaces, one for work and the other for recreation, and Bob is a member of each space. Alice selects the communications device for the space to initiate a call to Bob. Bob gets a call initiation indication on his device representation(s) for the given space.

**Example 2:** Alice, Bob, and Charlie are members of a space. When one of them initiates a call, both members receive a call initiation indication on their device representation(s). This is a type of follow-me conferencing. If Jim (a non-member) initiates a call to the addressed assigned to the space, then the associated endpoints of Alice, Bob, and Charlie receive a call initiation indication.

**Example 3:** Alice uses the communications device in the recreation space to call Bob. The call events are included in the recreation space timeline. Later Alice calls Bob using the communication device in the work space. The call events are included in the work space timeline.

4.5 Context Aware Collaboration Sessions

Enterprise collaboration sessions have two factors that distinguish them from other forms of collaboration sessions. One is the context that surrounds the collaboration session and the other is the need for a sequence of related collaboration sessions over a period of time. Note that though the participants are important, often it is the case that the context and temporal aspects are important. For example, collaboration sessions that involve a project continue even if members of the team leave. Discussion of context is beyond scope of this paper. However, context is used here as a general term to capture key aspects of collaboration sessions such as the intent of the session, temporal nature of data, content associated with the session, information about participants, etc.

One feature of such context aware collaboration session is to allow applications, such as relevant contacts, to use the context to mine relevant data to generate a user specific view for the session. One can see that the intent of one participant, a customer, can be the context of a collaboration session. The session would involve an appropriate customer agent with one or more experts trying to resolve the customer issue.

4.6 Groups in Collaboration Sessions

ConnectedSpaces provides a notion of a group, where a set of users can be identified as a group. Their capabilities and access controls can be managed as a group. This group could have a separate group view that contains data mined from the group’s information and shared among members of the group. The
ability to have groups allows collaboration sessions to include a large set of people without requiring all of them to be part of the session and without managing their individual identities.

5. Feature Interactions

Open platforms with distributed shared resources represent systems with specific affinity for feature interactions that can’t easily be engineered away due to the number of contributing developers and continual change in the services. Our ultimate goal is to formally define a run-time feature interaction and detection approach, building on previous work by the authors [22][18][20] and others. The system detects at run-time the potential for features to interact and either blocks the low priority feature causing the interaction or alerts the affected user to take action.

In addition, as the majority of feature interaction research to date has focused on telephony and more recently web services, the identification of new categories of feature action in collaboration environments like ConnectedSpaces is an important result for establishing directions for further study.

In the following sections feature interactions are categorized according to the functional areas described in the previous section. Table 2 presents feature interactions organized by feature category and then feature interaction (FI) category. Subsequent subsections describe these in more detail.

<table>
<thead>
<tr>
<th>Feature Category</th>
<th>FI Category</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space Composition</td>
<td>Multiple simultaneous writes to a non-transactional shared resource via gadgets</td>
<td>Alice and Bob have a calendar gadget to a group calendar in different spaces, and update the same entry in the group calendar at the same time.</td>
</tr>
<tr>
<td></td>
<td>One or more read operations simultaneous with a write of a non-transactional shared resource in one or more spaces via gadgets</td>
<td>Alice and Bob use a meeting room reservation gadget to book a specific meeting room via the company’s meeting room reservation site at a specific time. Alice accesses the reservation form first and finds the room is available, and but it needs a projector. Alice takes time to decide. Bob signs on and reserves it instantly. Alice decides to reserve the room, and finds it is no longer available.</td>
</tr>
<tr>
<td></td>
<td>Changing application data through a gadget while simultaneously using the application to update the data outside the space</td>
<td>Alice directly updates a group calendar at a specific entry while Bob updates the entry using a gadget in a shared space.</td>
</tr>
<tr>
<td></td>
<td>Two or more “real-time” features, mediated by gadgets in the same space</td>
<td>Alice and Bob are simultaneously using a shared space which contains one gadget which is viewing an eBay auction and another gadget to place bids for the same auction. Bob is placing a bid while Alice is viewing the auction. Alice sees the bid transmitted but doesn’t see the auction view updated. Alice embeds a real-time search gadget into her space which looks for postings in the blogsphere about her company’s products. Around the time of new product announcements by her company, the search gadget produces a burst of redundant results from different sites distributing the same press releases. Alice embeds a follower-gadget into her team’s space which follows blogs by other company product groups. Later Alice is transferred to one of the other product groups and publishes a blog for it. Her blog entries end up in her original team’s space.</td>
</tr>
<tr>
<td></td>
<td>Space persistence and user memory</td>
<td>Alice creates a space S1 with a robot entity. The robot creates a new sub-space and sets up a real-time feed of related topics whenever Alice updates her interest profile. Later, for another space, Alice adds a topic to her interest profile, resulting in many sub-spaces being created in S1 and auto-subscriptions inserted.</td>
</tr>
<tr>
<td></td>
<td>Dynamic membership</td>
<td>Alice is a member of space for a period of time and then leaves it due to change in job function. When Alice later replays the space history, the space replay function allows her to see space sessions after she left the space membership.</td>
</tr>
<tr>
<td>Space is a Communications Endpoint</td>
<td>Incoming + outgoing calls. Conventional feature sets introduce conventional FIs</td>
<td>e.g., call forwarding with call blocking;</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Robots making simultaneous incoming calls</td>
<td>A robot calls 2 spaces of Alice at the same time. Multiple robots call into same space of Bob at same time. Multiple robots call into different spaces of Alice at same time</td>
<td></td>
</tr>
<tr>
<td>Incompatibility between call origination and call termination features</td>
<td>Alice and Bob share a space but have different call origination and call termination feature preferences. E.g. Alice and Bob have different block lists, and Alice enables call waiting while Bob doesn’t.</td>
<td></td>
</tr>
<tr>
<td>Calls between members of spaces vs calls to/from non-members</td>
<td>Alice and Bob share a space S1. Alice gives Charlie the space address to discuss something related to the space. Charlie calls Alice and the conversation about the topic is captured in to S1. During the conversation, Charlie brings up personal information that Alice doesn’t want Bob to know.</td>
<td></td>
</tr>
<tr>
<td>Device limits</td>
<td>Bob is out the office and has forwarded communications for certain spaces to his cell phone. Bob can participate in communications within those spaces but due to limits of his cell phone, cannot see the information displayed that others members of the space see.</td>
<td></td>
</tr>
<tr>
<td>Private communications</td>
<td>Alice, Bob, Charlie and Dawn share a space. Charlie and Dawn insert a private sub-space to discuss a topic related to the space. A robot that generates meeting highlights and that was added by Charlie in the space is able to see the private sub-space and posts highlights of the sub-space to the space.</td>
<td></td>
</tr>
<tr>
<td>Embedded Communications Conventional telephony FI</td>
<td>See [22][14][13][15].</td>
<td></td>
</tr>
<tr>
<td>Intra-telephony gadget coordination</td>
<td>Alice and Bob share two spaces, S1 and S2. Alice embeds a SIP-based telephony gadget in S1 and Bob embeds a Skype-based telephony gadget in S2. When Alice is on either gadget, and then picks up an incoming call on the 2nd gadget, the first call is not automatically placed on hold.</td>
<td></td>
</tr>
<tr>
<td>Sharing conflicts</td>
<td>Alice and Bob share two spaces, S1 and S2. Each space has an embedded telephony gadget. An incoming call to gadget in S1 is answered by Alice. While the first call is active, an incoming call to the gadget in S2 arrives. Alice has configured go-to-cover on busy. Bob answers the 2nd call while the connection to Alice goes to cover.</td>
<td></td>
</tr>
<tr>
<td>Component-to-Component Communication Robot feedback loops</td>
<td>Alice’s space produces an RSS feed using a robot, such that when the space is updated by an entity, a summary entry may be placed in the feed. Alice’s space receives other RSS feeds from other spaces. Bob’s space receive this Alice's feed into his space, which also produce RSS feeds via a robot. Charlie’s space receives Bob's RSS feed and publishes to a feed received by Alice in her space, creating a feedback loop, such that one or more entries published by Alice's feed are passed back to her space via Bob and Charlie’s connection, leading to continual republishing.</td>
<td></td>
</tr>
<tr>
<td>Robot ping-pong</td>
<td>Robots R2D2 and R2D3 are members of Alice’s space and may insert new content from external sources when an entry in the space triggers the robot. Later the operators of R2D3 expand its triggering and topic publishing list, with the result that R2D2 and R2D3 overlap and trigger each other to add entries to Alice’s space continuously.</td>
<td></td>
</tr>
<tr>
<td>Group Management Identify equivalence not enforced across group manager boundaries</td>
<td>Alice has Tom on her block list. Tom is a member of group G1. Alice creates a space with group G1 as a member. Tom is able to access the shared space.</td>
<td></td>
</tr>
<tr>
<td>Semantic Synchronization Semantic channel out of sync with syntactic channel</td>
<td>Alice and Bob have a shared space and have added a robot that provides real-time expert advice on topics discussed in the space. In addition, a separate robot provides transcription service. During a conversation in the space, Bob retracts an earlier point. This is recorded by the transcription service but the robot misses the retraction in the semantic level and continues to refer to the original.</td>
<td></td>
</tr>
</tbody>
</table>
5.1 Space Composition
Spaces can be hierarchically composed. This permits entities to structure the space to enhance navigation of its content. The variety of objects that can be included in a space is intended to be open-ended, and to extend beyond the collaboration platform to include websites, real-time information sources, applications, and other collaboration and communication environments.

A type of feature interactions that could arise as a result of this composition model is due to simultaneous manipulation of shared information on remote applications, through gadgets embedded in spaces. Users may or may not know of the sharing, depending on how access to the remote information source is mediated. Since the shared information is stored outside of the space, there is potentially no limit to the number of simultaneous spaces and users that may reference it. In addition, gadgets from different application providers may reference the same information or application data, and may not be designed to coordinate. Further, the user may have access to the same applications and information through tools outside of the collaboration framework. For example, a user could be editing their desktop calendar directly and also through a gadget in a space.

These examples are essentially distributed synchronization problems in which well-known uncoordinated write-write or write-read operations cause state inconsistencies for applications without transactional mechanisms. If the framework provides a protocol for locking or consistent distributed time-stamping to resolve these synchronization issues, there is nevertheless the possibility that many external information systems may not provide this support. While these problems are not unique to or necessarily due to the collaboration framework per se, the fact that application use is mediated by the space may give users a false sense of safety.

5.2 Embedded Communications
Embedded communications refers to real-time communications applications such as softphones embedded as a gadget in the space. This enables communications to include the space as context, and the communications act to become part of the space record.

Example: There are two call gadgets. One has an active call. When a call comes in on second gadget, the original call should be held automatically. There are variations of this: 1) the two call gadgets can be in separate spaces, 2) the gadgets are for different services, say P2P and SIP, especially when they are in different spaces.

5.3 Component-to-Component Communication
Objects can communicate directly, via connections within a space, or indirectly, through external connections. This can lead to feedback loops and ping-pong interactions between robots. In addition, robots and gadgets could be provided by third parties. During the lifetime of a space, a configuration of a robot or gadget could be changed by the provider. Thus an interaction could be uncovered between robots due to a subsequent configuration change.

5.4 Asymmetric Perspectives
In collaboration systems without user roles, a fundamental aspect of a shared space is symmetry among the participants. This assumption of symmetry underlies user behavior. Some features can introduce to asymmetric perspectives in a shared space, due to local settings (filters, block lists), or the asymmetric nature of the application (calls). (Note: Views may also affect this, but suggestions would help.)

Example: Stock ticker gadget gets live updates. Everyone belonging to the space sees (approximately) the same view as new updates arrive.

Example: Local filtering. Alice has settings which block certain types of news stories on news feed embedded in the space, Bob does not. Bob can see certain news stories in the space, Alice cannot.

Example: Locale-based filtering. Alice is in country A, Bob is in country B. Alice embeds a restricted web site gadget in the space. Bob cannot see it.
5.5 Group Management

Group management includes group creation, establishing rules for group membership, and join/leave functions. More advanced features include filters based on group settings, and group nesting (i.e. groups of groups).

Since there are many collaboration forums using different frameworks, it is convenient to be able to reference groups defined in external systems within the ConnectedSpaces group. For example, a mail list group set up in a standards body like the IETF, or members of the contact list on a particular social network account could be used as a member of a space.

Example: A group is a member of a wave. The membership of the group is determined outside the framework (e.g. IETF Group). So the size of the group can vary dynamically without the wave being able to control it.

Some attributes of the space depend on group size, e.g., the size of the voting gadget. Or in the virtual world case, a member is invited to a room in the space, but can’t enter it due to space constraint.

The group membership update rate could be much greater than the capability of the ConnectedSpaces server to handle it. This could lead to anomalies in enabling access. In addition, the external group might be used to circumvent block lists for a space.

5.6 Semantic Synchronization

The introduction of real-time semantic operators and agents in the space introduces interesting feature interaction categories. One has to do with synchronization between semantic and syntactical channels. The following example assumes there is a voice call in the space and that transcription and summarization tools provide additional channels. Agent applications can monitor these channels to provide additional information to the participants.

channel 1: voice
channel 2: transcription of channel 1
channel 3: semantic summary of channel 2. Robots listen to this channel for keywords which trigger particular actions

When a participant retracts an earlier statement, the semantic summary may miss the retraction. Or the robots listening may not recognize the retraction keyword. Thus, the robots continue to refer to the original statement.

5.7 Spaces as Communication Endpoints

Since spaces represent a communications endpoint, each space is addressable for communications signaling. Actual communication devices can be enterprise phones or any other communications endpoint.

In the following, the communications-related feature interactions are categorized. Generally, where conventional call control features are employed, conventional interactions can be observed (first category below). The remaining categories are due to functionality and conditions in collaboration platforms not commonly found in telephony environments.

- As in conventional telephony, feature sets for call-origination, call-termination, and mid-call can be defined, and the associated feature interactions can occur.
- The space is effectively a shared communications endpoint. Then there are feature interactions due to this shared view, such as 1) which member's features are used, 2) calls between members of spaces vs call to/from non-members.
- A user is member of multiple spaces with concurrent communications in these spaces. These may interact if there are different underlying communications infrastructure associated with different spaces.
- Different members of a space may have local configuration or locale-specific filtering differences. They may have defined views for the space which introduce asymmetry.
- A sub-set of a space’s members may create a private communications area as a subspace. The boundaries of this space may effect features such as space history or views.
Table 3 Feature Interactions for ConnectedSpaces as Communication Endpoints

<table>
<thead>
<tr>
<th>Feature Interaction</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space has incoming and outgoing &quot;call&quot; capability. Conventional call features</td>
<td>e.g., call forwarding with call blocking;</td>
</tr>
<tr>
<td>introduce conventional interactions</td>
<td></td>
</tr>
<tr>
<td>Robot making simultaneous incoming calls to the same user</td>
<td>A robot calls 2 spaces of Alice at the same time. The robot may be designed to avoid multiple simultaneous calls to the same endpoint, but because of the indirect endpoint addressing used, the robot may not distinguish different spaces as having the same users.</td>
</tr>
<tr>
<td>Robots making simultaneous incoming calls to the same user</td>
<td>Multiple robots call into same space of Bob at same time, when the roots are interrelated and are designed to coordinate their calls. Multiple robots call into different spaces of Alice at same time</td>
</tr>
<tr>
<td>Incompatibility between call origination and call termination features</td>
<td>Alice and Bob share a space but have different call origination and call termination preferences. For example Alice and Bob have different block lists, and Alice enables call waiting while Bob doesn't.</td>
</tr>
<tr>
<td>Calls between members of spaces vs. calls to/from non-members</td>
<td>Alice and Bob share a space S1. Alice gives Charlie the space address to discuss something related to the space. Charlie calls Alice and the conversation about the topic is captured in to S1. During the conversation, Charlie brings up private information that Alice doesn't want Bob to know.</td>
</tr>
<tr>
<td>Device limits</td>
<td>Bob is outside the office and has forwarded communications for certain spaces to his cell phone. Bob can participate in communications within those spaces but due to limits of his cell phone, cannot see the information displayed that others members of the space see.</td>
</tr>
<tr>
<td>Private communications</td>
<td>Alice, Bob, Charlie and Dawn share a space. Charlie and Dawn insert a private sub-space to discuss a topic related to the space. A robot that generates meeting highlights added by Charlie is able to see the private sub-space and posts highlights of the sub-space to the space.</td>
</tr>
</tbody>
</table>

6. Feature Interaction Approach

The components which operate in a collaboration space, be it robots, gadgets, applications or features will come from a variety of sources and development teams. As has been shown in previous work [15], in such environments run-time feature interaction handling brings a number of advantages. The number of involved components can be large and hence the number of feature pairs which need analysis is likely to be unmanageable. Furthermore, pair-wise analysis or testing of features before deployment may be simply impossible as the different teams might be unaware of each other’s offerings. On top of this, new features are continuously developed and hence may invalidate an earlier analysis. Consequently, this paper advocates a run-time technique.

Such techniques require a machine representation of features which is used to automatically detect feature interactions. In previous work [22], the authors adopted the Event-Condition-Action (ECA) model [11]. This model is sufficient and general enough to describe features in ConnectedSpaces.

With these approaches, users can define rules such as ‘if a call arrives from Bob during working hours, forward to the mobile’, using the common (trigger, condition, actions) paradigm. In this example, the trigger is the call attempt from Bob, the condition is that the time of day is during working hours, and the action is the forwarding to the mobile number. This paradigm is essentially identical to the ECA, or (event, condition, actions) paradigm that is well known from reactive databases, agent systems, access control systems and the semantic web.

Fundamentally, a rule is enabled when its trigger occurs and its condition is met. However, there is an important difference between the trigger and the condition: a trigger can be an external or internal event which may convey parameters used in the other two parts, condition and actions. Conditions, on the other hand, can query ‘context’ information, such as the time of day or entries in the user’s on-line diary. If the rule is triggered, and the condition is met, one or more actions are executed. This paradigm allows for many variations and has been previously applied to call control. One such variation, the use of post-conditions to describe the effects of the action, has previously been used for feature interaction analysis [12], [18], [19], [29], [38]. This then leads to the paradigm (trigger, pre-condition, action, post-condition) which is used in this paper.
feature ::= (trigger, pre-cond, action, post-cond)
where:  pre-cond ::= (states, action parameters)
           action ::= (trigger, action parameters)
           post-cond ::= (new triggers, new states, affected values)

Here, collaboration features are defined as the manipulation and viewing of a shared space and its objects by multiple entities.

6.1 Types of Feature Interactions

To detect feature interactions between pairs of feature descriptions, triggers and pre and post-conditions of the features are analyzed. Using the definition in [11] feature interactions are defined to be of one of the following four types of interactions between any two features $F_1$ and $F_2$:

- **Concurrency conflict:**
  \[
  \exists c_1 \in \text{pre}(F_1) \land \exists c_2 \in \text{pre}(F_2) \land \text{trigger}(F_1) = \text{trigger}(F_2) \implies (c_1 \rightarrow \neg c_2)
  \]

- **Disabling conflict:**
  \[
  \exists c_1 \in \text{post}(F_1) \land \exists c_2 \in \text{pre}(F_2) \land \neg \text{trigger}(F_1) \implies \text{trigger}(F_2) \implies (c_1 \rightarrow \neg c_2)
  \]

- **Result conflict:**
  \[
  \exists c_1 \in \text{post}(F_1) \land \exists c_2 \in \text{post}(F_2) \land \text{trigger}(F_1) = \text{trigger}(F_2) \implies (c_1 \rightarrow \neg c_2)
  \]

- **Enabling interaction ($F_1$ enables $F_2$):**
  \[
  \exists c_1 \in \text{post}(F_1) \land (\exists c_2 \in \text{pre}(F_2)) \land \text{trigger}(F_1) \neq \text{trigger}(F_2) \implies (c_1 \rightarrow c_2)
  \]

In this notation, $\text{pre}(F)$ is the pre-condition of a feature $F$, and $\text{post}(F)$ represents the post-condition of the feature. $\text{trigger}(F)$ is the trigger for $F$. It is important to note that conflicts can form a chain, e.g. if $F_1$ disables $F_2$, and $F_2$ enables $F_3$, then $F_1$ also disables $F_3$.

If two actions start from or lead to mutually inconsistent system states, they are incompatible and should not be simultaneously executed. Here interactions due to pre-conditions are named Concurrency conflicts and interactions due to conflicting post-conditions are named Result conflict interactions. In the case where one feature creates post-conditions which meet another feature’s pre-conditions and triggers are named enabling interactions. Where the post-conditions of one feature contradict the pre-conditions of another and thus prevent the second from executing this is a disabling conflict.

Conflicts among pre/post-conditions of more than two actions are also possible. However this kind of analysis is rarely performed because it becomes complex and very few concrete examples (where three actions can be in conflict without any two of them being in conflict) are known [40].

6.2 Call State and Events Dictionary

The approach advocated in this paper requires a dictionary of events and states. For telecommunications services a base dictionary is defined in [22]. For the benefit of the reader, core elements are included in this section and extended to support example services used to demonstrate feature interactions in collaboration platforms as endpoints for communication services.

| Table 4 Call States |
|---------------------|-----------------|
| **Call States ($S_c$)** | **Description** |
| Trying              | Indicates a call being attempted from a device (a SIP UAC/UAS sends/receives an INVITE). |
| Proceeding          | Indicates that a call is being received (a SIP UAC/UAS receives/sends a 1xx notag response). |
| Early               | Indicates that a call dialog has been defined, usually the called party is playing the ring tone and the calling party is playing the ringback tone at this state (a SIP UAC/UAS receives/sends a 1xx notag response). |
| Confirmed           | A call session has been established (a SIP UAC/UAS receives/sends a 2xx response). |
| Terminated          | A call or call attempt has been terminated (cancel from UAC, rejected from UAS, bye from either side, or being replaced by another call). |
Table 5 Call events

<table>
<thead>
<tr>
<th>Call Events (Ec)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancelled</td>
<td>UAS gets a CANCEL message</td>
</tr>
<tr>
<td>Rejected</td>
<td>UAC gets 486, or 603 message</td>
</tr>
<tr>
<td>Replaced</td>
<td>UAC receives a new INVITE with Replace</td>
</tr>
<tr>
<td>local-bye</td>
<td>UAC sends a BYE</td>
</tr>
<tr>
<td>remote-bye</td>
<td>UAS sends a BYE</td>
</tr>
<tr>
<td>Error</td>
<td>UAC receives 481 or 408 responses</td>
</tr>
<tr>
<td>Timeout</td>
<td>UAC does not receive any responses</td>
</tr>
<tr>
<td>Redirected</td>
<td>UAC receives 3xx responses</td>
</tr>
<tr>
<td>Answered</td>
<td>UAC receives 2xx responses</td>
</tr>
<tr>
<td>Incoming</td>
<td>UAS receives INVITE</td>
</tr>
<tr>
<td>Outgoing</td>
<td>UAC sends INVITE</td>
</tr>
</tbody>
</table>

Table 6 Device states

<table>
<thead>
<tr>
<th>Device states (Sa)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>available</td>
<td>Audio device available</td>
</tr>
<tr>
<td>busy</td>
<td>Audio device busy</td>
</tr>
<tr>
<td>muted</td>
<td>Audio device is muted</td>
</tr>
<tr>
<td>unread-message</td>
<td>Message waiting light on</td>
</tr>
<tr>
<td>no-message</td>
<td>Message waiting light off</td>
</tr>
</tbody>
</table>

Table 7 User States

<table>
<thead>
<tr>
<th>User states (Su)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline</td>
<td>User is offline</td>
</tr>
<tr>
<td>Available</td>
<td>User is available for voice communication</td>
</tr>
<tr>
<td>Busy</td>
<td>User is busy</td>
</tr>
<tr>
<td>do-not-disturb</td>
<td>User cannot accept any conversation requests</td>
</tr>
</tbody>
</table>

Table 8 Visibility states

<table>
<thead>
<tr>
<th>Visibility States (Sdisp)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>Information is visible on display</td>
</tr>
<tr>
<td>Invisible</td>
<td>Information is not visible on display</td>
</tr>
</tbody>
</table>

Table 9 Recording states

<table>
<thead>
<tr>
<th>Recording states (Srec)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data-avail</td>
<td>Recorded data available</td>
</tr>
<tr>
<td>no-data</td>
<td>No data available</td>
</tr>
</tbody>
</table>

These states and events can then be used to describe services. For illustration, two examples are included below. Further examples can be found in Section 6.4 and in [22].
<table>
<thead>
<tr>
<th>Feature name</th>
<th>Trigger</th>
<th>Pre-condition</th>
<th>Action</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call Forward</td>
<td>$E_i$(incoming)</td>
<td>$S_i(C_i) = \text{early}$, $C_i = \text{call(local, remote)}$</td>
<td>$\text{referred-to} = \text{select(forwarded-to)}$, $\text{connect(remote, forwarded-to)}$, $\text{disconnect(local)}$</td>
<td>$S_i(C_i) = \text{terminated}$, $S_i(C_j) = \text{trying</td>
</tr>
<tr>
<td>Unconditional</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information</td>
<td>$E_i$(answered)</td>
<td>$S_i(C_i) = \text{confirmed}$, $C_i = \text{call(local, remote)}$</td>
<td>$\text{record}(C_i)$</td>
<td>$S_{rec}(\text{data-avail}(C_i))$</td>
</tr>
<tr>
<td>Recording</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the following it is shown how the approach can be used to detect interactions from Section IV.

### 6.3 Applying the Feature Interaction Approach to ConnectedSpaces

As discussed earlier the types of features and the heterogeneity in the providers of the various applications interworking in a collaboration environment dictate the use of a runtime approach to tackle feature interactions. The application of the approach at runtime requires that the descriptions of the features and components can be brought together for applying the four rules from Section 5.1. In the literature, runtime approaches have been applied both as a centralized approach and distributed approach. Here the use of a hybrid of the two methods is advocated. Firstly, a Feature Interaction Manager is used within the ConnectedSpaces Server. Clients send feature information to the server at key events, and the feature manager component of the server will then apply the four rules to any pair of such feature descriptions. This centralized part of the approach will cover all components internal to the collaboration platform.

However, if one of the components is external to the platform (e.g. an external call placed from the space) then the features from the external component will need to be included in this check. Clearly, a centralized approach does not work well in such a setting. Thus the centralized feature manager is extended linking it with a distributed technique as described below.

![ConnectedSpaces server with communications server](image)
The main components of a collaboration platform are depicted in Figure 7. This shows three users (Alice, Bob and Charlie) which are members of a space (space-2334). This is served by the ConnectedSpaces server which is connected to a SIP server managing enterprise internal communications and also external links. The link between the ConnectedSpaces server and the SIP server is facilitated using event notifications. That is, individual robots, gadgets, applications, etc. will forward a description of their behavior to the ConnectedSpaces server which operates a feature interaction manager. This manager module within the server will collect all feature descriptions and then apply the conflict rules to each pair of these.

If an external component is involved in the collaboration platform (e.g. call to tom@abc.com in Figure 8) then the descriptions of any features which get activated outside the platform, will need to be included in the feature interaction analysis. In [22] and [14] the authors discuss the application of a distributed approach to an enterprise SIP setting. The basic idea is that SIP messages are used to transport the feature descriptions. A P-Contype header is defined which carries the feature descriptions between distributed components. Each SIP component will have an implementation of the feature interaction algorithm and will apply the conflict rules to any pair of the component’s own features and combinations of its own features and feature descriptions contained in the message. In this distributed case, the server will deal with the interactions locally to the server (as described above) and then forward any feature descriptions to external nodes. Likewise it will include feature descriptions received from external nodes in its own checks.

![Figure 8: Call Flow to initiate a collaboration session.](image)

If these checks result in no interaction being found, the subsequent behavior is unaltered. If an interaction is detected, it will need resolving. In some circumstances, the resolution requires ‘undoing’ an earlier executed feature. Clearly this is problematic with features changing state or communicating with other resources. Hence the case where an already triggered feature needs ‘undoing’ should be limited as much as possible. Thus the detection and resolution strategy proposed here, employs pre-activation detection.

In terms of resolving a detected interaction, one of the two features involved in the interaction needs to be disabled. However, determining which one is difficult and depends very much on preference and context. It might depend on the roles of the owners of the components (manager vs. secretary) or if one of the features is external (internal features overrule external ones). Indeed, there are 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 [44]. However, other approaches [45][46] may also be applicable. Based on these algorithms, the outcome of one of the two features involved needs to be discarded.

If the actions of the second service are to be discarded, the service is simply not carried out and the resolution ends. If the actions of the first service are to be discarded, the call setup might need to be repeated, if the second feature actually gets triggered (depend on feature data). Hence the second feature will be executed to determine this. If it does not get triggered, there is no interaction in this particular instance and the call proceeds as normal with just the description of the first feature included in the P-Contype header. However, if the second service does get triggered, an interaction would occur and hence the call attempt needs to be repeated – disabling the first service.
A message indicating this is sent back to the call originator (a SIP Response 380 Alternative Service). The P-ConType header for the feature to be disabled is extended by the field Status=disabled. The SIP 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 relevant feature, the feature will not execute.

In order to ensure that the presented approach can cover external components to ConnectedSpaces, these components will need to support and implement the approach. Features which are applied within a domain which does not support the approach cannot be included in the analysis. In this case, feature interaction analysis within the other domains is still possible. While this will resolve interactions between services used within these domains, it will not capture interactions involving services from the ‘closed’ domain.

### 6.4 Experimental Results

In the following the operation of the approach is demonstrated by using the examples from Table 3. For a more complete list of feature descriptions please refer to [22].

**Example 1:** Incompatibility between call origination and call termination features: Here two (or more) users share a space which acts as a communication endpoint. Users essentially use the same line and can be considered being connected using a Party Line like approach. Both users may use different sets of features, or have incompatible settings for the same features. For example, user 1 may have a Call Forwarding Unconditional feature whereas user 2 has no call handling features active. In this case, the behavior which both users expect is different. User 1 expects all calls being forwarded to an external location and hence no calls should ever be established. The description for this behavior is:

<table>
<thead>
<tr>
<th>trigger(E, (incoming))</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre(S_c(C_1) = early, C_1 = call(local, remote))</td>
</tr>
<tr>
<td>action(referred-to = select(forwarded-to), connect(remote, forwarded-to), disconnect(local))</td>
</tr>
<tr>
<td>post(S_c(C_1) = terminated, S_c(C_2) = trying</td>
</tr>
</tbody>
</table>

User 2 on the other hand expects calls to arrive. That is the post condition can be formulated as:

| post(S_c(C_1) = confirmed, S_d(local) = busy, S_d(remote) = busy, S_d(remote) = busy) |

Combining both descriptions will result in a *results conflict* interaction:

\[ c_1 = \text{post}(Sc(c1)) = \text{terminated (Call Forwarding)} \]
\[ c_2 = \text{post}(Sc(c2)) = \text{confirmed (no feature)} \]

Clearly, here \( c_1 \rightarrow \neg c_2. \)

**Example 2:** Device limits: In this example the interaction is due to the expectation that all Space communication will be consumed on a big screen. This assumption is broken by forwarding features which direct communication towards devices with limited capabilities, such as a mobile phone. This might lead to some information not being displayed. To analyze this interaction visibility states are required for the information of particular features.

| \( S_{disp}(f1) = \text{visible} \in \text{post}(f1) \land \) |
| \( S_{disp}(f2) = \text{invisible} \in \text{post(mobile-screen-mgmt)} \land \# \) |
| \( S_{disp}(f2) = \text{visible} \in \text{pre}(f2) \land \) |
| \( S_{disp}(f2) = \text{invisible} \rightarrow \neg(S_{disp}(f2) = \text{visible}) \) |

This behavior leads to a *Disabling conflict* interaction as displaying the information belonging to feature 1 prevents the information of feature 2 being shown.
**Example 3:** Private Communication: In this example, one feature implemented by a robot is trying to record information on some communication. However, a private communication perhaps in a subspace should not be recorded. This interaction is a typical example of an interaction which is due to broken assumptions. The recording feature is unaware of the concept of a private subspace which might have been introduced subsequently. This interaction can be captured by focusing on the post-conditions of the two features. One feature states that recorded data is available, which is prevented by the other. The recording feature can be described as:

```
trigger(E, answered))
pre(S, C1 = confirmed, C1 = call(local, remote))
action(record(C))
post(data avail(C))
```

The privacy feature of the private space can be described as:

```
trigger(E, answered))
pre(S, C2 = confirmed, C2 = call(local, remote))
action()
post(no_data(C))
```

The conflict is due to different post conditions, one feature providing the recorded data, whereas the other explicitly preventing this.

\[
\begin{align*}
C_1 &= \text{post}(Sc(C)) = \text{data avail (Recording)} \\
C_2 &= \text{post}(Sc(C)) = \text{no data (private comms)}
\end{align*}
\]

The result of this is a results conflict interaction.

### 6.5 Non-functional interactions

Besides the functional interactions discussed above, the use of collaboration spaces as communication endpoint also gives rise to a number of non-functional interactions mainly due to security, performance and trust implications.

A secure space might have the feature that all calls are transcribed and available for subsequent reference. A requirement of this feature might be that all participants of such calls must be members of this space. However, clearly a forwarding feature might redirect a call to an external party which is not a member of the group, thus disabling the recording functionality.

Some elements of a space may be complex and resource intensive. Such elements may slow down the processing for other elements which have some real-time requirements. This might affect time outs of communication features such as treatment of calls on ‘no answer’.

Finally, trust is a major issue when closed environments, such as enterprise communications systems, are integrated with online collaboration spaces.
7. Conclusions

This paper has reported novel work on analyzing interactions between features in federated collaboration environments. A new model is presented for a federated real-time collaboration environment called *ConnectedSpaces* and used this to illustrate key feature sets in collaboration platforms, including views, spaces as communication endpoints, space persistence and structuring, space history, embedded gadgets and robots, semantic processing, and integration with other collaboration frameworks.

Using the presented feature set, novel types of feature interactions have been categorized, illustrated, and analyzed. Interactions have been identified which are due to space composition, embedded communication, using a space as a communication endpoint, group communication, component to component communication, asymmetric perspectives, and semantic synchronization. As these types do not rely on proprietary functionality of ConnectedSpaces, these types are believed to apply to collaboration platforms generally rather than being specific to ConnectedSpaces.

Furthermore, an approach is presented which handles feature interactions which involve features using the collaboration platform as a communications endpoint. This approach uses the paradigm *(trigger, pre-condition, action, post-condition)* to describe the features. The flexibility of the approach is demonstrated by being able to capture the variety of services found in collaboration platforms. As is demonstrated by the case study in this paper, this approach copes well with the complexity of the services encountered.

Pairs of feature descriptions are analyzed using four types of feature conflict: concurrency conflict, enabling conflict, disabling conflict, and result conflict. These types of conflict have previously been shown to apply in policy based systems and widgetized communication. Here their applicability to collaboration platforms is demonstrated. To capture all features from different sources a runtime approach is employed which has been applied in a centralized manner (with a feature manager) and also in distributed fashion. The technique is demonstrated through examples and it is shown how the approach can be integrated with an online collaboration environment. However, the approach depends on capturing key conditions of services. Missing pre- or post-conditions in the description of features leads to potentially missing feature interactions. Due to the pairwise application of the feature interaction detection algorithm at runtime, the approach scales well with increasing number of services in the system. The paper illustrates strategies to resolve detected interactions. In the future the experimentation with the feature interaction detection approach will be extended to include additional types of features encountered in federated collaboration environments.

REFERENCES


[10] X. Wu and V. Krishnaswamy, Widgetizing communication services, ICC 2010, Capetown, South Africa
[27] X. Wu and V. Krishnaswamy, Widgetizing communication services, ICC 2010, Capetown, South Africa.


