Feature Interaction Analysis for Collaboration Spaces with Communication Endpoints

Mario Kolberg
Dept. of Computing Science and Mathematics, Univ. of Stirling
mko@cs.stir.ac.uk

I. INTRODUCTION

Today's enterprise collaboration platforms include web conferencing systems, online document editing, shared document repositories, and voice and video conferencing. Some integration of basic communication features in such platform is starting to appear, e.g. softphones and instant message components are being integrated.

Previously [1] we have defined ConnectedSpaces, a collaboration platform extending existing offerings with features to extend their functionality and to integrate them with enterprise communication systems.

Unfortunately, the introduction of new features will also lead to new feature interactions. Here we extend our previous work in feature interaction [4][5][6][7] and investigate for the first time feature interactions in collaboration spaces. We employ a runtime feature interaction approach and show through examples that the approach can be applied to such environments. In this paper we present the following results:

- We describe communications features within ConnectedSpaces, an enterprise collaboration platform.
- We analyze the proposed feature set and describe new categories of feature interactions.
- We propose a runtime feature interaction approach to capture these feature interactions and demonstrate the applicability of the approach with examples.

II. RELATED WORK

Collaboration platforms vary from wikis, blogs, and shared documents to web-based collaboration systems to 3D collaboration spaces such as virtual worlds. The focus in this paper is on added features to collaboration systems rather than the underlying collaboration tools used. For a survey of collaboration platforms, see for example [12].

In separate work, we have introduced ConnectedSpaces, an advanced online collaboration platform which supports enterprise grade features including space addressing and nesting, group management, space history and temporal control as well as using collaboration spaces as communication endpoints. In this paper we extend this work, investigating the types of feature interactions in such environments. In particular we focus on interactions associated with using the collaboration platform as a communication endpoint.

If different services interwork, and their joint behavior is unacceptable, these services are said to interact. 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]. Often services are written in isolation without knowledge of the other service. When the services meet in the network, one service may break assumptions made by the other.

While little work has been done on feature interactions in collaboration environments, there exists a very substantial body of work on feature interaction in telecommunication systems [7][14]. Feature interactions in web services have been studied by Weiss et al.[8][9][10]. Generally approaches can be applied before deployment or after deployment of the services. The former group is referred to as offline approaches and the later on-line or runtime approaches. Offline approaches suffer from the limited information available on which services will be deployed in the network and hence will interwork. 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.

III. 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 [4][16][17] and others. The system would detect at run-time the features interaction and either block the low priority feature causing the interaction or alert 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, we believe that the identification of new categories of feature action in collaboration environments like ConnectedSpaces is an important result for establishing directions for further study. Table 1 presents feature interactions organized by feature category and then feature interaction (FI) category. Subsequent subsections describe these in more detail.
Table 1 Categories of Feature Interactions

<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 in one or more spaces 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>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>
<td></td>
</tr>
<tr>
<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>
<td></td>
</tr>
<tr>
<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 update. Alice embeds a real-time search gadget into her space which looks for postings in the blogosphere 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>
<td></td>
</tr>
<tr>
<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 S2, Alice adds a general topic to her interest profile, resulting in many sub-spaces being created in S1 and auto-subscriptions being inserted.</td>
<td></td>
</tr>
<tr>
<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>
<td></td>
</tr>
<tr>
<td>Space is a Communications Endpoint</td>
<td>Space has incoming and outgoing ‘call’ capability. Adding conventional call feature sets introduces conventional Fls e.g., call forwarding with call blocking.</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. For example Alice and Bob have different block list, 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 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 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>
<td></td>
</tr>
<tr>
<td>Embedded Cloud</td>
<td>Conventional telephony features interactions</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 an incoming call, and then picks up an incoming call on the 2nd gadget, the first call is not automatically placed on hold.</td>
</tr>
<tr>
<td>Intra-telephony gadget coordination</td>
<td>Alice and Bob have a two-space, 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 second call while the connection to Alice goes to cover.</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 a space produces a non-SIP 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 receives a RSS 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 entities published by Alice’s feed are passed back to her space via Bob and Charlie’s connection, leading to continual rebroadcasting.</td>
<td></td>
</tr>
<tr>
<td>Semantic Synchonization</td>
<td>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 record is by the transcription service but the expert robot misses the retraction in the semantic level and continues to refer to the original point.</td>
</tr>
</tbody>
</table>

A. 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 space. 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.  

**Example:** A web.alive gadget is embedded in a space which shares a user’s desktop. Members of the space may simultaneously change the desktop including attributes of the ConnectedSpaces application.

### B. Space as Communication Endpoint

In this category, the space itself represents a communications endpoint. Thus the space is addressable for communications signaling, and that all members of the space are notified for call setup. This category is discussed in more detail in Section IV.

### C. Embedded Communications

Embedded communications refers to real-time communications applications such as softphones embedded as gadgets in the space. This enables communications to include the space as context and the communication be included in 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, especially when they are in different spaces.

### D. 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.

### E. 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 can not.

**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 can not see it.

### F. Group Management

Group management includes group creation, establishing rules for membership, and join/leave functions. Advanced features include filters based on group settings, and group nesting. 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. So the group size can vary 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. Furthermore, the 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. Further, the external group might be used to circumvent block lists for a space.

### G. 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. Agents 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 actions

When a participant retracts something said previously, the semantic summary may miss the retraction. Or the robots listening may not recognize the retraction keyword. As a result, the robots continue to refer to the original statement.

### IV. 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. We categorize the communications-related feature interactions as follows:

- 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 the different 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 2 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 ‘call’ capability. Conventional call features introduce conventional interactions</td>
<td>e.g., call forwarding with call blocking;</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 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>

V. 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 [7], 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 others offerings. On top of this, new features are continuously developed and hence may invalidate an earlier analysis. Consequently, in this paper we advocate a run-time technique.

Such techniques require a machine representation of features which is used to automatically detect feature interactions. In previous work [4], we adopted the Event-Condition-Action (ECA) model [13]. This model is sufficient and general enough to describe features in ConnectedSpaces.

Here, we define collaboration features as manipulation and viewing of a shared space and its objects by multiple entities.

This approach requires a dictionary of events and states. For telecommunications services we define a base dictionary in [4]. Here this dictionary is extended to support collaboration platforms as endpoints for communication services.

A. Types of Feature Interactions

To detect feature interactions between pairs of feature descriptions, we analyze triggers and pre and post-conditions of the features. Using the definition in [13] we define feature interactions of one of the following four types of interactions between any two features $F_1$ and $F_2$:

- Concurrency conflict:
  $$(\exists c_1 \in \text{pref}(F_1)) \land (\exists c_2 \in \text{pref}(F_2)) \land \text{trigger}(F_i) = \text{trigger}(F_j) \rightarrow (c_1 \rightarrow \neg c_2)$$

- Disabling conflict:
  $$(\exists c_1 \in \text{post}(F_i)) \land (\exists c_2 \in \text{post}(F_j)) \land \text{trigger}(F_i) \neq \text{trigger}(F_j) \rightarrow (c_1 \rightarrow \neg c_2)$$

- Result conflict:
  $$(\exists c_1 \in \text{post}(F_1)) \land (\exists c_2 \in \text{post}(F_2)) \land \text{trigger}(F_i) = \text{trigger}(F_2) \rightarrow (c_1 \rightarrow \neg c_2)$$

- Enabling interaction ($F_i$ enables $F_j$):
  $$(\exists c_1 \in \text{post}(F_i)) \land (\exists c_2 \in \text{post}(F_2)) \land \text{trigger}(F_i) \neq \text{trigger}(F_2) \rightarrow (c_1 \rightarrow c_2)$$

In this notation, $\text{pref}(F)$ 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 $F_3$. In the following we show how the approach can be used to detect interactions from Section IV.

B. Feature Interaction Examples

In the following we demonstrate the operation of the approach by using the examples from Table 2. For a more complete list of feature descriptions please refer to [4].

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 behaviour is:
trigger(Ec(incoming))
pre(S_c(C_1) = early, C_1 = call(local, remote))
action(referred-to = select(forwarded-to), connect(remote, forwarded-to), disconnect(local))
post(S_c(C_1) = terminated, S_c(C_2) = trying| proceeding| early| confirmed, C_2 = call(forwarded-to, remote), S_d(local) = avail, S_u(local) = busy, S_d(remote) = busy, S_u(remote) = busy)

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_u(local) = busy, S_d(remote) = busy)

Combining both descriptions will result in a results conflict interaction:
c1 = post(S_c(C1)) = terminated (Call Forwarding)
c2 = post(S_c(C2)) = confirmed (no feature)

C. 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.

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 we need visibility states for the information of particular features.

S disp(f1) = visible ∈ post(f1) ∧
S disp(f2) = invisible ∈ post(mobile-screen-mgmt) ∧
S disp(f2) = visible ∈ pref(f2) ∧
S disp(f2) = invisible → ¬(S disp(f2) = 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 one 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_c(answered))
pre(S_c(C_1) = confirmed, C_1 = call(local, remote))
action(record(C_1))
post(data_avail(C_1))

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

trigger(E_c(answered))
pre(S_c(C_2) = confirmed, C_2 = call(local, remote))
action()
post(no_data(C_2))

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

C1 = post(S_c(C1)) = data available (Recording)
C2 = post(S_c(C2)) = no data (private comm)

The result of this is a results conflict interaction.

VI. INTEGRATION OF THE APPROACH IN A SPACE

The runtime application of the approach requires that the descriptions of the features and components can be brought together for applying the rules as defined in Section III. We propose the use of a Feature Interaction Manager within the ConnectedSpaces Server. Clients will then send feature information to the server at key events.
The general proposed architecture is depicted in Figure 2. 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.

For runtime detection of feature interaction, we advocate a centralized approach covering all internal components to the collaboration platform. That is, individual robots, gadgets, applications, etc. will forward a description of their behavior (as discussed in Section III) to the ConnectedSpaces server which operates as 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.

![Figure 3: Call Flow to initiate a collaboration session.](image)

However, if one of the components is external to the platform (e.g. an external call placed from the space as in Figure 3) then the features from the external component will need to be included in this check. However, a centralized approach does not work well in such a setting. Hence we advocate the use of a distributed technique for this scenario. In [4] and [5] we 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. Each 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.

If these checks result in no interaction being found, the subsequent behavior is unaltered. If an interaction is detected, it will need resolving. In this case one of the two features involved in the interaction needs to be disabled. Which one 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). A in-depth discussion on such cases is provided in [5].

VII. CONCLUSION

Collaboration platforms are emerging in the enterprise context. ConnectedSpaces is a platform which supports advanced enterprise functionality including space history, embedded gadgets and robots, semantic processing, and integration with other collaboration frameworks. In this paper using the ConnectedSpaces model, for the first time, we have categorized, illustrated, and analyzed new types of feature interactions for collaboration platforms. We have further presented an approach which handles feature interactions which involve using the collaboration platform as a communications endpoint. To capture all features from different sources we have employed a runtime approach. Through examples we have demonstrated the technique and shown how the approach can be integrated with an online collaboration environment.

REFERENCES