

Feature Interactions Between Internet Services and Telecommunication Services

Xiaotao Wu, John Buford, Kishore Dhara, Venky Krishnaswamy
IP Communications Department
Avaya Labs Research
xwu, buford, dhara, venky@avaya.com

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

ABSTRACT

The automated run-time detection of feature interactions in Web 2.0 communications applications is an important problem that has not been addressed to date. Such web-enabled communication services are constructed as reusable web widgets that can be composed by users in a web interface. Widgetizing communication features as web services can better serve users with highly customizable features, friendly user interfaces, and easier integration with other web services. However, it also introduces new feature interaction problems. As we show, these composite communication services combine web services and VoIP features in highly dynamic interfaces with application and service state that is typically distributed across multiple domains. In this paper, we present these new feature interactions and propose our solution. We present ten different feature interactions and organize them into a taxonomy. A general method for detecting FIs is given in FOL notation. We also present a coordinator plug-in mechanism for independent widgets to share feature information and state that can be implemented in today's browsers. We finally describe a run-time algorithm for FI detection that is suitable for this architecture and uses the FI notation presented in the paper.

Categories and Subject Descriptors

H.4.3 [Communications Applications]: Computer communication and information browsers

General Terms

Management, Design

Keywords

Feature Interactions, Widgetized Communication, Voice over IP

1. INTRODUCTION

The introduction of Voice over IP (VoIP) allows tighter integration of telecommunication services and other Internet services. For example, web-based communication services like web conferencing, click-to-dial, and voicemail as email attachments are widely used. This tighter integration enables many innovative services. However, it also introduces several challenges, such as handling feature interactions (FI) across both Internet and telecommunication service domains.

In separate work which is the basis for this paper, Wu and Krishnaswamy [14] have developed a general method for combining widgetized communications and web services. These

browser-based interfaces allow a rich set of information and communication user interface components called *widgets* to be composed into an integrated dashboard style application. Later in Section 3 of this paper we briefly summarize some of the capabilities of such widgets. A key point for this paper is that while the widgets are standalone components, when composed into a web browser, they can be used to cooperatively provide composite converged services. *The identification, modeling and detection of intra-widget feature interactions in such converged services is the key focus of this paper.*

The majority of work in feature interaction detection and resolution to date has typically focused on either telecommunication services or more recently web services. Since the convergence of these types of services has been feasible only recently, the issues of feature interactions in converged services is relatively unexplored. As we discuss later, the browser-based delivery of these converged applications is attractive because it is widely accessible on a variety of end user devices, gives the end user a great degree of control over customization and configuration, and permits the user to access a rich third party component set.

These integrated mash-up style interfaces and the distributed nature of the backend services make it difficult to perform feature interaction detection. First the services are mediated by a collection of client-side components which have certain intra-widget communication restrictions due to security constraints at the browser. Second, the state information about each service is potentially distributed to a large number of servers in different administrative domains. Third, the user has a high degree of dynamic control over the selection of widgets from potentially different vendors.

In this paper we present the following results:

1. We formalize our approach to FI detection and resolution of converged widgetized services by introducing a notation to describe the necessary rules (section 4). This notation focuses on communications related services. Its extension to general web services, which we believe is feasible, is not described in this paper.
2. We identify a taxonomy of feature interactions for converged services and give a detailed description with many examples (section 5).
3. We present a new widget coordinator mechanism for client-side collection of intra-widget communication which is the basis for FI detection (section 6)

- We explain how previous work by the authors [17] in FI detection and resolution in communication services can be used together with our notation and coordinator plug-in to provide FI detection and resolution at the converged services level (section 7).

Section 2 summarizes related work in run-time FI detection. Section 3 briefly introduces our widgetized communication service framework and presents some example services. Section 4 presents the formal notation for describing FIs. Section 5 discusses new feature interaction problems in widgetized communication services. Section 6 presents the detection algorithm, Section 7 describes how an existing FI detection mechanism for communication services can be integrated with our design, and Section 8 concludes this paper.

2. RELATED WORK

Telecommunication services and web services employ different models for feature composition. For telecommunication services, one popular feature composition model is the pipe-and-filter model, in which features are connected in a chain of filters in the order in which they get to process events. While in web services, services are more distributed and usually, multiple services run in parallel. There are many different service composition techniques [6][7][8], but in general, we can consider the composition of web services as building a *flow system*, which can be more complicated than the pipe-and-filter model. The stages in the flow system can be connected in many different ways, not just in sequence, data exchange can also be in many ways, and the stage orchestration can be centralized (e.g., a blackboard that coordinates all the events and requests) or distributed (each stage subscribes to other stages' events).

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 [20]. Quite 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.

There exists a very substantial body of work on feature interaction [18][19]. 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. Online approaches can 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.

Modeling the behavior of services has been the basis for a number of 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.[21] use Linear temporal logic to show inconsistencies between incompatible services. Gibson [22] use First order logic and temporal logic of actions to show invariant violation and

nondeterminism as an indication of feature interactions. Frappier et al [23] also use first order logic to show inconsistencies between services. Wu et al [24] consider pre/post conditions of actions to detect interactions between high level services.

Interactions between Web services have been explored by Weiss et al [1][2][3][4]. Here rather complex models of web services are built and then manually analyzed to detect interactions. In this paper we use pre/post conditions to capture the behavior of web services. Furthermore, we show how this approach can be integrated with a run-time approach developed by the authors.

3. WIDGETIZING COMMUNICATION SERVICES

This section briefly introduces how to make communication services as web widgets. A web widget is a reusable and composable web component that can be added on a web page by an end user without requiring additional programming. Since the main focus of the paper is on feature interactions, we will not provide much detail about the widgetization process. Further information about our design is available in [14]. In the next section we divide the widgets into two categories:

- Communication Widgets (CW)
- Communication Enabled Widgets (CEW)

However the principles of widgetization and widget coordination are the same for each category, and the distinction between CW and CEW is primarily used in our FI detection algorithm presented in section 6.

Widgetized communication services have the benefit of being globally accessible, and usually with better user interface. For example, Figure 1 shows a *call forwarding* widget that can be embedded in a call handling page to forward incoming calls. Certainly it has a better user interface than displaying it on a phone. In addition, it can be more easily integrated with other Internet services, such as presence and information retrieval. Further, the widget can be reused in other web pages, e.g., in a web-based business processing flow.

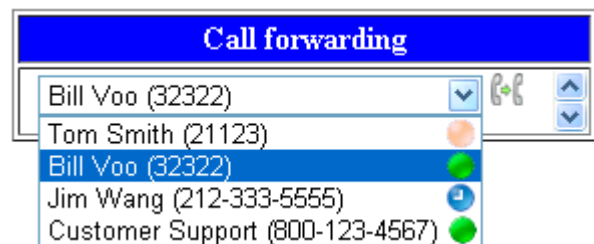


Figure 1 A call forwarding widget

To web-enable a communication feature is more than just providing a web interface to the feature. Rather, we need to make the feature reusable, composable, and customizable, like web widgets which are described later. In addition, by defining security and privacy constraints, the communication widgets should not only be embedded in telecommunication service providers' web pages, but also in other service providers' web pages.

Below we use the *Call Forwarding* widget shown in Figure 1 to illustrate the widgetization process. The Call Forwarding widget

involves call event notification, information retrieval, and presence information. It is triggered by an incoming call event, then employs information retrieval to get related people based on the caller or the subject of the call. Afterwards, it then fetches the presence information of the related people.

Building a web interface for call forwarding is straightforward. But the web interface has to interact with call control features to get incoming call events and forward calls. The interaction can be achieved at the server side by integrating the web server with the telecommunication server. It can also be achieved at the client side by using Asynchronous JavaScript and XML (Ajax) and Document Object Model (DOM) [10]. The benefit of client side interaction is that the widget does not have to rely on a specific telecommunication server so it can be reused by different service providers and can be easily composed with other web services. But client side interaction also requires the session management widget, which can receive dialog events and can control calls, be presented in the same page. Session management widget controls calls by accessing call control nodes. Call control nodes can be communication application servers, phones, or agents that can subscribe to users' call states. Section 6 discusses intra widget communication in more detail.

By interacting with the session management widget, the call forwarding widget can achieve basic call forwarding functions. It can then optionally interact with the information retrieval widget to get related people and the presence widget to get presence status.

4. FEATURE SPECIFICATION NOTATION

4.1 ECA Model

In telecommunication domain, we can follow the Event-Condition-Action (ECA) model and define a communication feature as “an action is performed when a trigger occurs and some conditions hold”, and can represent it as a tuple [15]:

$$\text{feature} ::= (\text{trigger}, \text{pre-cond}, \text{action}, \text{post-cond})$$

where:

$$\text{pre-cond} ::= (\text{states}, \text{action parameters})$$

$$\text{action} ::= f(\text{trigger}, \text{action parameters})$$

$$\text{post-cond} ::= (\text{new triggers}, \text{new states}, \text{affected values})$$

In this definition, pre-condition refers to feature-related values prior to performing feature action, and post-condition refers to feature-related values after feature action being performed. In the telecommunication domain, generally triggers are limited to call events, pre-conditions and post-conditions are limited to call states, and actions are limited to call handling. For widgetized communication services, we can still use the tuple to represent services, but need to extend the value scope of *triggers*, and *conditions*. The values will not be limited to telecommunication elements, it can also be values from other Internet services, such as presence status, a Really Simple Syndication (RSS) feed, an email, or an instant message. However, we still limit the *action* as call handling functions to distinguish communication widgets from communication enabled widgets. Then for each communication feature, accordingly, we have a *communication widget* (CW) defined by:

$$\text{CW} ::= (\text{trigger}', \text{pre-cond}', \text{action}, \text{post-cond}')$$

where:

$$\{\text{trigger}'\} = \{\text{trigger}\} + \{\text{Internet service events}\}$$

$$\{\text{pre-cond}'\} = \{\text{pre-cond}\} + \{\text{Internet service states}\}$$

$$\{\text{post-cond}'\} = \{\text{post-cond}\} + \{\text{Internet service states}\}$$

In addition to CWs, there are also *communication enabled widgets* (CEW), which cannot directly change call states. In other words, the *actions* of CEWs are not call handling functions. For example, performing information retrieval (IR) using caller's name as a key will not change call states and this IR is a CEW. We also use a tuple to define CEWs:

$$\text{CEW} ::= (\text{trigger}^*, \text{pre-cond}^*, \text{action}^*, \text{post-cond}^*)$$

In this tuple, at least one element must be related, or derived from a CW tuple element:

$$\exists \text{cew_e} \in \text{CEW}: \text{cew_e} = f(\text{cw_e}) \text{ where } \text{cw_e} \in \text{CW}$$

Therefore, we can also have a notation for CEW

$$\text{CEW} ::=$$

$$(\text{triggering CWs}, \text{CW related args}, \text{affected CWs})$$

$$\exists \text{cew_ne} \in \text{CEW notation}: \text{cew_ne} \neq \emptyset$$

To apply the above notation to define services, we first need to provide a dictionary of available events and states. The tables in the appendix section show the dictionary. These tables are not a complete list but captures states and events that are used in the examples in the rest of the paper. In addition, this dictionary is restricted to communication services. We expect that the extension of the dictionary to other categories of widgetized services is feasible without requiring changes to our detection algorithm or coordination plug-in described later. However we do not consider these extensions in this paper.

We use the states and events defined in RFC4235 [18] for call states and events, shown in Table 2 and Table 3. There are also device states and events, as well as user states and events shown in Table 4 to Table 7 respectively. Table 8 to Table 14 contain service specific states and events (voice mail, email, and information retrieval). This state and event information is then used in the remainder of the paper to capture the behavior of services and to detect interactions between them.

4.2 Feature Interaction Analysis Notation

To analyze feature interactions, we check trigger and pre and post-conditions of features. We follow the definition in [15] and have the following feature interaction types for two features F_1 and F_2 :

- Concurrency conflict:

$$(\exists c_1 \in \text{pre}(F_1)) \wedge (\exists c_2 \in (\text{pre}(F_2)) \wedge \text{trigger}(F_1) = \text{trigger}(F_2) \rightarrow (c_1 \rightarrow \neg c_2))$$

- Disabling conflict:

$$(\exists c_1 \in \text{post}(F_1)) \wedge (\exists c_2 \in \text{pre}(F_2)) \wedge \text{trigger}(F_1) \neq \text{trigger}(F_2) \rightarrow (c_1 \rightarrow \neg c_2)$$

- Result conflict:

$$(\exists c1 \in \text{post}(F1)) \wedge (\exists c2 \in \text{post}(F2)) \wedge \text{trigger}(F1) = \text{trigger}(F2) \rightarrow (c1 \rightarrow \neg c2)$$

- Enabling interaction (F_1 enables F_2):

$$(\exists c1 \in \text{post}(F1)) \wedge (\exists c2 \in ((\text{pre}(F2) \cup \text{trigger}(F2))) \wedge \text{trigger}(F1) \neq \text{trigger}(F2) \rightarrow (c1 \rightarrow c2)$$

In this definition, $\text{pre}(F)$ represents the pre-condition of a feature F , and $\text{post}(F)$ represents the post-condition of the feature. $\text{trigger}(F)$ is the correspondent trigger for $\text{pre}(F)$ or $\text{post}(F)$. Note that among these interactions, the last one is different from the other interactions in that F_1 can trigger F_2 or fulfill certain pre-conditions of F_2 . While in the other interactions, F_1 conflicts with F_2 . In this paper, we only focus on those conflicting feature interactions because usually the conflicting feature interactions may cause undesirable results and should be avoided. Also note

that conflicts can be propagated. For example, if F_1 disables F_2 , and F_2 enables F_3 , we have F_1 disables F_3 . We call this the conflict propagation rule.

4.3 Service Examples

Table 1 shows some service examples, including widgetized communication services and communication related web services. We use these services to illustrate feature interactions between communication services and Internet services. In this table, for each service, we use *local* to refer to the user or device that subscribes to the service. For a two party call, we use *remote* to represent the user that talks to the service subscriber. If there are multiple calls, we use C_i to present each different call. To make a concise table, for each feature, we only show the states and events being used in our feature interaction examples.

Table 1 Service examples

Feature name	Trigger	Pre-condition	Action	Post-condition
click-to-transfer	$E_u(\text{click}(\text{transfer}))$	$S_c(C_1) = \text{confirmed}$, $C_1 = \text{call}(\text{local}, \text{remote})$ $S_d(\text{referred-to}) = \text{avail}$, $S_u(\text{referred-to}) = \text{avail}$, $S_d(\text{local}) = \text{busy}$, $S_u(\text{local}) = \text{busy}$, $S_t(\text{remote}) = \text{ready}$	$\text{referred-to} = \text{select}(\text{IR results})$, $\text{connect}(\text{remote}, \text{referred-to})$, $\text{disconnect}(\text{local})$ (IR information retrieval)	$S_c(C_1) = \text{terminated}$, $S_c(C_2) = \text{confirmed}$, $C_2 = \text{call}(\text{referred-to}, \text{remote})$, $S_d(\text{referred-to}) = \text{busy}$, $S_u(\text{referred-to}) = \text{busy}$, $S_d(\text{local}) = \text{avail}$, $S_u(\text{local}) = \text{avail}$,
click-to-forward	$E_c(\text{incoming})$ $E_u(\text{click}(\text{forward}))$	$S_c(C_1) = \text{early}$, $C_1 = \text{call}(\text{local}, \text{remote})$ $S_d(\text{forwarded}) = \text{avail}$, $S_u(\text{forwarded}) = \text{avail}$, $S_d(\text{local}) = \text{avail}$, $S_u(\text{local}) = \text{avail}$, $S_d(\text{remote}) = \text{busy}$, $S_u(\text{remote}) = \text{busy}$, IR results available	$\text{forwarded} = \text{select}(\text{IR results})$, $\text{connect}(\text{remote}, \text{forwarded})$, $\text{disconnect}(\text{local})$	$S_c(C_1) = \text{terminated}$, $S_c(C_2) = \text{trying/proceeding/early/confirmed}$, $C_2 = \text{call}(\text{forwarded}, \text{remote})$, $S_d(\text{local}) = \text{avail}$, $S_u(\text{local}) = \text{avail}$, $S_d(\text{remote}) = \text{busy}$, $S_u(\text{remote}) = \text{busy}$, $E_c(\text{incoming}(\text{forwarded}))$
message waiting indication (MWI)	$E_d(\text{message-waiting})$	$S_v = \text{unread-message}$, $S_d(\text{local}) = \text{no-message}$	device: $\text{turn-on}(\text{MWI light})$	$S_v = \text{unread-message}$, $S_d(\text{local}) = \text{unread-message}$
	$E_d(\text{no-message-waiting})$	$S_v = \text{no-message}$, $S_d(\text{local}) = \text{unread-message}$	device: $\text{turn-off}(\text{MWI light})$	$S_v = \text{no-message}$, $S_d(\text{local}) = \text{no-message}$
sync-evoice (read e-voice and sync with voicemail server)	$E_c(\text{open}(\text{email}(v_1)))$ v_1 is a voicemail	$S_c(\text{email}(v_1)) = \text{unread}$	email server: $\text{read}(\text{email}(v_1))$ $\text{notify}(\text{voicemail server}, \text{open}(v_1))$	$S_c(\text{email}(v_1)) = \text{read}$ $E_c(\text{open}(v_1))$
	$E_c(\text{new}(\text{email}(v_1)))$ v_1 is a voicemail		email server: $\text{new}(\text{email}(v_1))$	$S_c(\text{email}(v_1)) = \text{unread}$
e-voice (voicemail as email attachment)	$E_v(\text{open}(v_1))$ v_1 is the last unread voicemail	$S_v = \text{unread-message}$,	voicemail server: $\text{changestate}(v_1)$, $\text{notify}(\text{phone}, \text{no-message})$	$S_v = \text{no-message}$, $E_d(\text{no-message-waiting})$
	$E_v(\text{new}(v_1))$ v_1 is the first unread voicemail	$S_v = \text{no-message}$,	voicemail server: $\text{save}(v_1)$, $\text{notify}(\text{phone}, \text{message-waiting})$ $\text{sendmail}(\text{email server}, (\text{email}(v_1)))$	$S_v = \text{unread-message}$, $E_d(\text{message-waiting})$ $E_c(\text{open}(\text{email}(v_1)))$
web-based personal assistant (for busy handling)	$E_c(\text{incoming})$	$S_c(C_1) = \text{early}$, $C_1 = \text{call}(\text{local}, \text{remote})$ $S_d(\text{local}) = \text{busy}$, $S_u(\text{local}) = \text{busy}$, $S_d(\text{bridge}) = \text{avail}$,	bridged extension: $\text{answer}()$	$S_c(C_1) = \text{held}$, $S_c(C_2) = \text{confirmed}$, $C_2 = \text{call}(\text{bridge}, \text{remote})$, $S_d(\text{local}) = \text{busy}$, $S_u(\text{local}) = \text{busy}$, $S_d(\text{remote}) = \text{busy}$,

				$S_u(remote) = busy,$ $S_d(bridge) = busy$
email forwarding	$E_c(new(email(v_1)))$ v_1 is a voicemail	$email(v_1) = unread$	email server: $forward(email(v_1))$	$S_c(email(v_1)) = removed$
information retrieval	$E_c(incoming) $ $E_c(outgoing)$	$S_c(C_1) = \neg terminated,$ $C_1 = call(local,$ $remote)$ $S_i(caller-id) = unavail$	$retrieve-info(caller-id)$	$S_i(caller-id) = ready$ $E_i(ir-done(caller-id))$ $S_c(C_1) = \neg terminated$
calendar-based call handling	$E_c(incoming)$	$S_u(local) = busy,$ $S_c(C_1) =$ $trying/proceeding/earl,$ $C_1 = call(local,$ $remote)$	$reject(C_1)$	$S_u(local) = busy,$ $S_c(C_1) = terminated$
presence	$E_u(presence)$		$notify(S_u)$	$E_u(presence)$
Call Forward Unconditional	$E_c(incoming)$	$S_c(C_1) = early,$ $C_1 = call(local,$ $remote)$	$referred-to = select(forwarded-to),$ $connect(remote, forwarded-to),$ $disconnect(local)$	$S_c(C_1) = terminated,$ $S_c(C_2) = trying/$ $proceeding/ early/$ $confirmed,$ $C_2 = call(forwarded-to,$ $remote),$ $S_d(local) = avail,$ $S_u(local) = avail,$ $S_d(remote) = busy,$ $S_u(remote) = busy$
Call Forward Busy	$E_c(incoming)$	$S_c(C_1) = early,$ $C_1 = call(local,$ $remote)$ $S_d(local) = busy,$ $S_u(local) = busy,$ $S_d(remote) = busy,$ $S_u(remote) = busy$	$referred-to = select(forwarded-to),$ $connect(remote, forwarded-to),$ $disconnect(local)$	$S_c(C_1) = terminated,$ $S_c(C_2) = trying /$ $proceeding/ early/$ $confirmed,$ $C_2 = call(forwarded-to,$ $remote),$ $S_d(local) = avail,$ $S_u(local) = avail,$ $S_d(remote) = busy,$ $S_u(remote) = busy$
Originating Call Screening	$E_c(outgoing)$	$S_c(C_1) = early,$ $C_1 = call(local,$ $remote)$	$disconnect(local)$	$S_c(C_1) = terminated,$ $S_d(local) = avail,$ $S_u(local) = avail,$ $S_d(remote) = avail,$ $S_u(remote) = avail$
Terminating Call Screening	$E_c(incoming)$	$S_c(C_1) = early,$ $C_1 = call(local,$ $remote)$	$disconnect(local)$	$S_c(C_1) = terminated,$ $S_d(local) = avail,$ $S_u(local) = avail,$ $S_d(remote) = avail,$ $S_u(remote) = avail$

5. FI BETWEEN COMMUNICATION SERVICES AND INTERNET SERVICES

Feature interactions in web services are not limited to functional interactions, which related to specific functional goals. There can also be non-functional goals[9], such as security, privacy, trust, and performance, putting constraints on functionalities of target systems. This section discusses both functional and non-functional feature interaction between Internet services and communication services in our widgetized communication framework.

5.1 Functional Feature Interactions

Functional feature interactions are similar to those in telecommunication feature interactions.

5.1.1 Goal conflicts

Goals describe the objectives that a feature should achieve. Goal conflicts means that achieving the goal of one feature may negatively affect achieving the goal of another feature

Example 1. Information retrieval and call screening:

Information retrieval can provide call related information, such as related people or related documents based on the ongoing call information, such as remote party's name, or the notation of the call. The retrieved information can be provided to other features, such as click-to-transfer or click-to-forward so users can transfer or forward calls with one click.

Users expect to click the retrieved people contact to transfer or forward calls, but if a retrieved person is on the call screening list,

the call cannot be completed. The information retrieval service (in Internet service domain) then conflicts with call screening service (in telecommunication domain).

In this example F_1 is Information Retrieval and F_2 is Call Screening. From Table 1, we can derive $c_1 = \text{post}(S_c(C_1) = \text{terminated})$ and $a_{c_2} = \text{post}(S_c(C_2) = \text{terminated})$, we have $c_2 \rightarrow \neg c_1$. From Section 3.3, we can conclude that this causes a ‘result conflicts’ feature interaction.

Example 2. Message Waiting Indication (MWI) and e-voice and email forwarding

Message Waiting Indication (MWI) notifies users for unread voicemails. E-voice delivers voicemail as email attachments.

If the voicemail server and the email server are not associated, and if the user listens to all his voicemails through email attachments, the user will still see the message waiting indication. If the voicemail server and the email server are associated so the voicemail server can be notified when the user listens to his voicemails through the email server, then if the user has email forwarding service enabled, the voicemail server will still lose track of voicemail accessing events because the forwarded email account is not associated with the voicemail server, and thus the MWI won’t be updated correctly.

For this feature interaction, we have

when an email with voicemail attachment arrives at an email server, it triggers *email-forwarding* feature:

$$\text{trigger}(\text{email-forwarding}) = E_e(\text{new}(\text{email}(v_1)))$$

For *sync-evoice* feature, it expects that user can read the email:

$$\text{trigger}(\text{sync-evoice}) = E_e(\text{open}(\text{email}(v_1)))$$

But with *email-forwarding*, the user will never read the email on the email server with *sync-evoice* feature, we have:

$$\text{trigger}(\text{email-forwarding}) \neq \text{trigger}(\text{sync-evoice}) \wedge$$

$$S_e(\text{email}(v_1) = \text{removed}) \in \text{post}(\text{email-forwarding}) \wedge$$

$$S_e(\text{email}(v_1) = \text{unread}) \in \text{pre}(\text{e-voice}) \wedge$$

$$(S_e(\text{email}(v_1) = \text{removed}) \rightarrow \neg S_e(\text{email}(v_1) = \text{unread}))$$

So, this causes a disabling conflict. *Email-forwarding* disables *sync-evoice* so that the status of the email with voicemail attachment cannot be synchronized with voicemail status. Further, we have

The post-condition of *sync-evoice* contains the trigger for *e-voice*:

$$E_e(\text{open}(v_1)) \in \text{post}(\text{sync-evoice}) \wedge$$

$$\text{trigger}(\text{e-voice}) = E_e(\text{open}(v_1)) \wedge$$

$$\text{trigger}(\text{sync-evoice}) \neq E_e(\text{open}(v_1))$$

So, *sync-evoice* enables *e-voice* on the $E_e(\text{open}(\text{email}(v_1)))$ event. Similarly, *e-voice* enables *MWI* on the $E_d(\text{no-message-waiting})$ event. Based on the conflict propagation rule, we have *email forwarding* in fact disables *MWI* on the $E_d(\text{no-message-waiting})$ event.

5.1.2 Shared trigger

Multiple features may be triggered by the same event. Executing one feature may block or change the behavior of another feature.

Example 3. Calendar-based call handling and Call Forwarding Always

Calendar-based call handling can check users’ calendar server (e.g., an Outlook server) and automatically answer or reject calls based on users’ schedule.

Both calendar-based call handling and Call Forwarding Always (CFA) features are triggered by an incoming call event. If CFA is enabled, calendar-based call handling will never be executed.

The two features are $F_1 = \text{Call Forwarding Always}$ and $F_2 = \text{Calendar-based Call Handling}$.

From Table 1, we have a $c_1 = \text{post}(S_c(C_1) = \text{terminated})$ and a $c_2 = \text{post}(S_c(C_2) = \text{ring} \mid \text{proceeding} \mid \text{early})$, we have $c_1 \rightarrow \neg c_2$. From Section 3.3, we can conclude that this causes ‘result conflict’ feature interaction.

Even in cases where $F_1 = \text{Calendar-based Call Handling}$ and $F_2 = \text{Call Forwarding Always}$, that is calendar-based call handling is non-deterministically sequenced first we still have feature interaction as follows.

Again, from Table 1, consider $c_1 = \text{pre}(S_u(\text{local}) = \text{busy})$ and $c_2 = \text{pre}(S_u(\text{local}) = \text{avail})$, we have $c_1 \rightarrow \neg c_2$ matching the ‘concurrency conflicts’ feature interaction. That is, if the Calendar-based Call Handling feature is triggered (user is busy), the Call Forwarding Always does not get triggered.

Example 4. Call Forwarding Always and web-based personal assistant functions

A web-based personal assistant uses a bridged extension to monitor the calls of a user and can then help handling calls. Figure 2 shows the widget. In a telecommunication system supporting Session Initiation Protocol (SIP), the personal assistant (PA) can subscribe to the primary extension’s dialog event. The assistant can answer calls by using SIP INVITE message with SIP “Replace” header. Figure 3 shows the call flow of using text-to-speech to talk to the remote user (useful when the user cannot talk, e.g., in a conference).

For an incoming call, the PA pops up on the dialog event notification, but with the Call Forwarding Always, the call will never be handled by the primary extension associated with the PA. Therefore, the PA functions will not work.

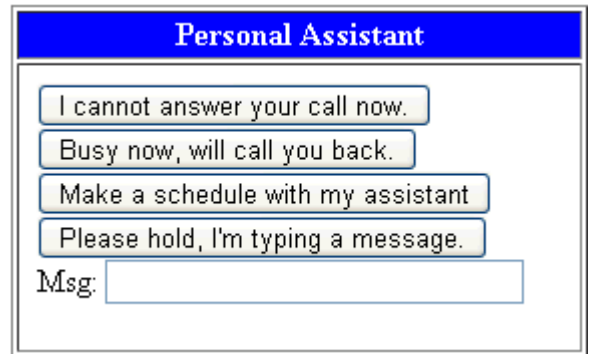


Figure 2 Personal assistant widget handling incoming calls

Let $F_1 = \text{CFA}$ and $F_2 = \text{PA}$ then we have both features triggered by the same incoming call event. Also, from Table 1, we have a $c_1 = (C_2 = \text{call(forwarded, remote)})$ and a $c_2 = (C_2 = \text{call(bridge, remote)})$, we have $c_1 \rightarrow \neg c_2$. From Section 3.3, we can conclude that this causes ‘disabling conflicts’ feature interaction.

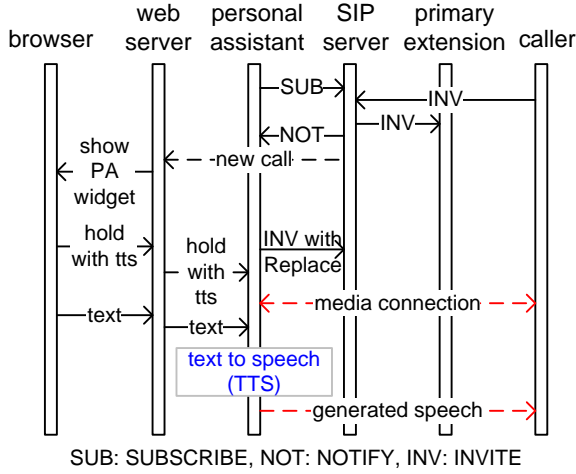


Figure 3 Call flow of using personal assistant widget

5.1.3 Race condition

Two features may be executed at the same time. Their execution order is non-determinable and a different order of execution will lead to different results.

Example 5. Event polling for presence-based call handling and presence status change

In many cases, web browsers use polling to get event or state updates. For a feature that relies on event polling, different order of the polling time and the event happening time may lead to different result.

For a presence-based call forwarding feature as shown in Figure 1, the feature keeps polling the presence status of related people. If a person’s status changes from “Busy” to “Available”, between the status changing time and the polling time, the user cannot make a right decision on presence-based call forwarding.

In this example, the problem is caused by the presence status measurement at the communication widget being out of sync with the status of the remote user. In general, representing race conditions requires some changes to the notation presented in section 3. Possible ways of dealing with these types of situations include:

- Introducing time in to the status notation. While this is the most general approach, it introduces the most complexity in to the notation. Since many FI cases are not race conditions, a goal is to avoid this approach.
- Distinguishing between measurement points, such as local value of a state and remote value of a state. For example,

$$S_u(\text{remote}) = \text{available} \wedge S_u(\text{remote}@browser) = \text{busy} \rightarrow \text{conflict.}$$

Here the user state as seen at the browser is different than the user state at the remote-device. As long as the number of

categories of such endpoints is limited, this approach could be practical.

- Elucidating distinct processing steps in an feature operation such as "poll", "retrieve", and so on.

We also note that the race condition scenario also effects the detection mechanism, since the ability for the widget coordinator (described in section 6) to detect a FI due to race condition depends on its ability to retrieve the data about the two endpoints having out-of-sync state. In practice the cost of this in terms of network overhead might outweigh the benefits.

5.1.4 Violation of feature assumptions

Violation of assumptions are caused by a feature that has an incorrect assumption about how another service works.

Example 6. Call forwarding and information retrieval on a remote party

Information retrieval on a remote party automatically retrieves the remote party’s information, including people, documents, appointments, and web links that may be related to the remote party, when a user tries to accept or make a call.

For an outgoing call, the information retrieval service retrieves the callee’s information. However, the call may be forwarded to another user by the downstream proxy server. The caller then gets incorrect retrieved information.

In this example F_1 is Information Retrieval and F_2 is Call Forwarding. Hence we can derive $c_1 = \text{post}(S_c(C_1) = \neg\text{terminated})$ and a $c_2 = \text{post}(S_c(C_2) = \text{terminated})$, we have $c_1 \rightarrow \neg c_2$. From Section 3.3, we can conclude that this causes ‘results conflicts’ feature interaction.

5.1.5 Resource contention

The user of one service may make some resources unavailable for other services to use.

Example 7. Competing for valuable screen space

Different locations on a homepage have different values to users. If a widget is located out of the screen and the user has to use the scrollbar to access the widget, this widget will be of less value to the user.

If we allow communication widgets to set their own size, one widget may push other widgets out of the screen. For example, a click-to-transfer may have a long list of related people and make the size of the widget for this feature very big and push other features out of the viewable area. This is especially important for portable devices that have very limited screen size.

For widgetized communication features, Table 1 does not show a common pre-condition for all the features. Usually, communication widgets require screen real estate to display the feature. As shown in 4, we use S_{disp} to refer to the state. The available values are visible and invisible. In this example, click-to-transfer’s display state is changed from invisible to visible when the call is answered. Suppose that the information retrieval widget is pushed out of the screen (invisible), this becomes a “disabling conflict”. We have:

$$S_{disp}(\text{click-to-transfer}) = \text{visible} \in \text{post}(\text{click-to-transfer}) \wedge$$

$$S_{disp}(IR) = invisible \in post(screen-mgmt) \wedge$$

$$S_{disp}(IR) = visible \in pre(IR) \wedge$$

$$S_{disp}(IR) = invisible \rightarrow \neg(S_{disp}(IR) = visible)$$

5.2 Non-functional Feature Interactions

Non-functional FIs [5] involve security, privacy, usability, performance.

5.2.1 Security consideration

Supporting third-party created services can greatly facilitate rapid service creation. However, it may also cause severe security problems.

Example 8. Secure ASR services and call forwarding

A CEW that performs speech recognition services can be used by many CWs. To securely analyze calls, the ASR CEW could be triggered only if certain security considerations such as caller or called identity are met. The security constraint for ASR CEW may be broken if the call is forwarded to an unauthorized party.

For example, only members of group A are authorized to use the ASR_A, and the policy requires that all parties in the call must be authorized for ASR_A to be used for that call. On some call, the caller and callee satisfy the security requirement. But when the call is forwarded, the new party doesn't satisfy the security requirement. This is expressed as an instance of result conflict which states: 1) a pre-condition of ASR_A is all users are members of group A, 2) a post-condition of ASR_A is all users are members of group A, 3) a post-condition of CF is some user is not a member of group A.

$$\forall u \exists c_1 \in pre(ASR_A) = S_u(u \in A) \wedge$$

$$\forall u \exists c_1 \in post(ASR_A) = S_u(u \in A) \wedge$$

$$(\exists c_2 \in post(CF) = S_u(u \notin A))$$

$$\rightarrow (c_1 \rightarrow \neg c_2)$$

As another example, the results of the ASR_A might only available to members of group A.

5.2.2 Performance Consideration

Rendering a widget with a sophisticated or visual complex user interface may affect the loading of other widgets, thus affect the result of those features with time constraints.

Example 9. Delayed loading of the call handling widget and voicemail

An analytic widget with a lot of dynamic content may delay the loading of a call handling widget. The long wait of the call handling widget may delay the user to click to handle the call and may result in triggering call coverage features, such as the voicemail feature.

5.2.3 Trust management

Trust management is very important when we combine telecommunication services with Internet services. Telecommunication services used to be in a closed, centralized system so the trust management is not as important as that for Internet services.

Example 10. Dialog state notification, presence notification, and eavesdropping on ongoing calls

Dialog state notification can notify subscribers about the states of calls (e.g, an incoming call event). Presence notification can notify subscribers about users' presence status, including "on the phone" status, and an eavesdropping service can monitor users' ongoing calls.

Suppose there is an eavesdropping service that subscribes to the dialog state notification service so it can be triggered once the user is on a call. But the dialog state notification service does not trust the eavesdropping service so the eavesdropping service cannot get call states. However, the eavesdropping service is trusted by a presence service, and the presence service is trusted by the dialog state notification service. The eavesdropping can then get part of the user's dialog state information through the presence service. This is a common relay chain problem for trust management. The special part of the problem is that two telecommunication services that do not trust each other can be bridged by an Internet service.

5.3 Taxonomy

Figure 4 summaries the relationship between the preceding examples into a taxonomy of FIs for widgetized communication services.

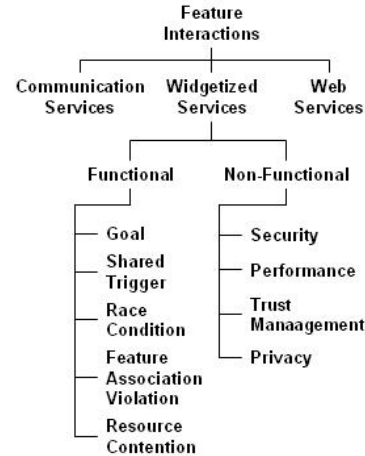


Figure 4 Taxonomy of FIs for widgetized communication services

6. DETECTING FI BETWEEN INTERNET AND TELECOMMUNICATION SERVICES

6.1 Intra Widget Communication

As discussed in section 2, there are various run-time approaches to detecting feature interactions both in web services and in the telecommunication protocols [6]. To apply existing techniques to detecting FIs between Internet services and telecommunication services, widgets of each type must be able to communicate with each other. The technical issues here include:

- Avoiding requiring each widget to interface to two or more server programming architectures
- Avoiding custom server-to-server communication and/or custom widget-to-widget communication

- Conforming to existing security constraints on browser API and intra component communication
- Permitting widgets that use web and communication servers which are in different network administrative domains

The intra-widget communication can be achieved both at the server side and at the client side. The following sub-sections compare client-side and server-side alternatives. The benefit of handling intra widget communication at the server side is that it is easier to deploy and can better enforce security and privacy policies. But it is difficult to utilize desktop resources and requires more network traffic. For computationally intensive features, such as speech-to-text, a distributed solution of utilizing desktop resources is more attractive.

6.1.1 Server side using Java Portlets or WSRP

The Java Portlet specification [11] provides a componentized user interface for web services. It specifies several means for coordinating different services, such as events, application sessions, and public rendering parameters. By these means, portlets can exchange information. The drawback of the specification is that it does not support cross-domain communication, that is, the ability of a widget not using a portlet service to communicate with another widget using a portlet service.

Complimentary to Java Portlet specification, the Web Services for Remote Portlets (WSRP) specification [12] provides a means for using web services, such as SOAP, to handle cross-domain portlet communication.

6.1.2 Client side using Ajax and DOM

Widgets can also interact with each other at the client side (web browsers) by using Asynchronous JavaScript and XML (Ajax) [13] and HTML Document Object Model (DOM) [10]. There are two benefits of handling intra-widget communication at the client side. First, widgets residing in the same page do not need to send and receive events back-and-forth via a server. This reduces the use of network bandwidth and improves performance. Second, it is easier to develop a coordinator (a blackboard) to handle all the widgets, including third-party created services, for a single user at the browser.

Using Ajax and DOM, we define a new way of achieving intra-widget communication at the client side by using a browser plug-in and DOM event model. We refer to this as the *Coordinator Plug-in (CP)*. As discussed in the next sub-section, the CP receives feature related events from the CWs and CEWs, and performs the FI detection algorithm. It then passes FI detection and resolution events to the corresponding widgets.

We require that each CW and CEW contains two pre-defined XML elements, one for sending events to the CP, and the other for receiving events from the CP. The CP inspects every loaded page, finds the pre-defined elements, and subscribes to the *update* events of the elements. This way, widgets written by different server programming technologies and in different *iframes* can communicate with each other through the plug-in. A detailed description of this method is presented in [14].

6.2 FI Detection Using the CP

Since the CP can access all the widgets, we can easily use it to handle feature interactions between widgets. For each event type,

the coordinator maintains a list of widgets to relay the event to. For each widget, the coordinator has the tuple of the widget. The coordinator then checks the tuple of all the widgets that may get the event and does the FI detection for each pair among the widgets:

- If both widgets are CWs, we use existing detection rules, such as [15][16][17] to detect feature interactions. In the next section we describe how the rules from [17] can be mapped to the notation presented in section 4.
- If one widget is a communication service and the other is an Internet service, we need to check whether the post-condition of the communication service affects the pre-condition of the Internet service. The reason we perform this check is that in general, Internet services will not change call states. They usually rely on some attributes to perform their actions
- If both widgets are Internet services, we need to check the goal and post-condition of both widgets and see whether they conflict.

For each communication session, the number of states handled by a CP is bound to the states used in the active widgets presented in the web page for that session. Practically, we do not expect a user to present hundreds of features in a single web page, and the number of states in pre-conditions and post-conditions of a feature is limited, as shown in Table 1. Therefore, in practice, CP should be able to handle FI detection in real time on a moderate personal computer. More detailed measurement on the performance of FI detection is in our future work.

6.3 Two-Step FI Detection

We separated the definition of CW from CEW so we can compose features in two steps. First, for CWs, we can use the existing techniques, such as pipe-and-filter and Distributed Feature Composition (DFC), to sequence them. In the second step, we can then consider CEWs as branches to the CW sequence. Figure 5 shows the composition method. The components in thick dotted lines are in the second step.

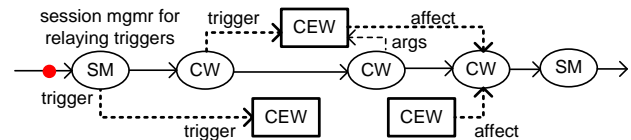


Figure 5 Two step feature composition

There are two reasons for this two step feature composition: First, the purpose of CEWs in our framework is to provide supports to CWs. It is appropriate to keep CW related information, including how features composed, unchanged. Therefore, building CW sequence first and keeping it are desirable. Second, usually the communication logic of CWs is in closed controlled telecommunication systems. Compared to open distributed web environment, in which CEWs are usually performed, it is easier to handle feature interactions in the closed telecommunication systems. Further, there are many existing works on feature composition in telecommunication systems. We can employ the existing work to handle CW feature interactions.

Figure 6 shows the feature interaction between information retrieval (IR) and call forwarding (CFW), which we discussed in

Section 4. Both SM and CFW affects IR and causes undesired feature interaction.

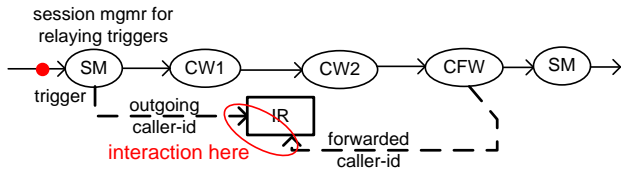


Figure 6 IR and CFW feature interaction

This two step feature composition can also illustrate a new feature interaction that is not discussed in Section 4. Figure 7 shows this kind of feature interaction. *CW1* invokes *CEW*, while *CEW* invokes *CW3*. Therefore, the *CEW* creates a branch that may cause *CW3* be executed before *CW2*, which is not the expected *CW* sequence. We call this kind of feature interactions “changing *CW* order” feature interaction.

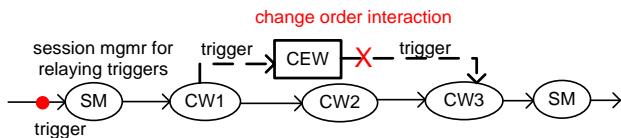


Figure 7 Changing CW order

7. ADDING PRE AND POST CONDITIONS TO THE K-M APPROACH

The K-M approach [16][17] uses high-level compact service descriptions which are inserted in the SIP message after a service has been active on the call. Before subsequent services are activated on a call, an algorithm using simple rules checks each service description carried in the SIP message against the description of the service to be activated. If no interaction is detected, the service is activated as usual. However, if an interaction is detected, only one of the involved services is allowed to be active on the call. This might be achieved by simply disallowing the second service, or by repeating the call such that the first service is prevented from being activated.

The rule notation describes the behavior of a service with the triggering party and a connection type. The latter consists of the connection to be set up before the service is activated, and the connection set up after the service has been triggered. For example, the feature Call Forwarding Unconditional (CFU), which redirects all incoming calls to a predefined third user, can be described as follows. Assume party A is the originator, B the terminator, and C the party where the call is redirected to. In the first part, the notation TP.: B indicates that B is the triggering party, as CFU is triggered at the terminating end of a call. In the connection type, $(A,B) \rightarrow (A,C)$, (A,B) is the original connection and (A,C) is the connection after activating the service (called resulting connection). For a pair, such as (A,B) , A is the source and B the destination. The call starts with A attempting to connect to B. However, due to CFU, A is connected to C instead.

Treatments are an important aspect of this approach. Treatments are announcements or tones triggered by the network to handle certain conditions during a call, for example when a call is screened or blocked. For example, the connection type for Terminating Call Screening can be shown as $(A,B) \rightarrow$

(A, TREAT) , that is A called B but the call is blocked and A gets Treatment instead.

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

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

For brevity the rules are not repeated here, but can be studied in [16][25].

Each service which gets activated includes its description into the SIP message. If there is already one or more entries in the message, these are checked against the description of the current service. Thus the algorithm is executed wherever necessary and a central feature manager is *not* required. This makes the approach highly scalable.

In order for the K-M approach to work together with the web service approach presented in this paper, information on all services communications services (e.g. SIP call control services) and web services needs to be brought together and analyzed. The communication services amongst themselves can be analyzed using the K-M approach. However, checking the communication services against web services requires checking both types of descriptions with each other. Here we advocate to forward the communications service descriptions out of band to the coordinator plug-in. Here the K-M service descriptions get translated to the pre/post condition format used in this paper. This can be automatically done. For instance, if the receiving party of the original connection is the triggering party, the trigger is set to an incoming call at that party. Similarly, if the triggering party is the originator of the original connection, the trigger is set to an outgoing call at that party. Connection types such as $(A,B) \rightarrow (A,C)$, that is the original connection gets changed, gets translated to a precondition indicating a call attempt between A and B, and parties A and B being busy. The post condition shows terminating the original call and setting up a second one. Furthermore, part B is set to available and parties A and C are busy. A connection type $(A,B) \rightarrow (A, \text{TREAT})$ also gets translated into a pre condition showing a call attempt between A and B, and both parties being busy. The post condition in this case shows the terminated first call, but no second call setup. Parties A and B are available. The examples of CFU and TCS services shown in the K-M notation above are also captured in the pre/post condition syntax in Table 1. Clearly, the pre/post condition notation is more expressive than the K-M syntax, so the translated descriptions from the K-M syntax are not as detailed as the pre/post condition syntax would allow. For instance, the difference of Call Forwarding on Busy and Call Forwarding Unconditional cannot be captured as the K-M syntax has no concept of this. However, as has been shown in an extensive case study [17] these abstractions do not impact on feature interaction detection between call control services.

8. CONCLUSION AND FUTURE WORK

This paper discusses new feature interaction problems introduced by widgetizing communication services and integrating them into a web environment. This integration may cause Internet services to interact with communication services in several different ways. We illustrate both functional interactions and non-functional interactions between Internet services and communication services. To detect and resolve these new feature interactions, Internet services and communication services must be able to communicate with each other. We introduced both server side and client side intra-widget communication methods. The client side cross domain intra-widget communication, which we implemented by using a browser plug-in, has the benefit of having the plug-in being able to communicate with all the widgets. We then proposed to use the plug-in as a coordinator to handle feature interactions. The coordinator checks the tuple of each widget and uses different means to handle interactions between two communication widgets, a communication widget and an Internet widget, and between Internet widgets.

Research on handling feature interactions in converged services is still in its early stage. There are still many problems that need to be discovered and resolved. We will continue experience our client-side plug-in approach for coordinating features. We expect to discover more research and development problems during the experience such as how to retrieve feature sequence from communication networks and how to handle dynamic feature composition. In addition, we also plan to investigate how to combine server-side feature composition approaches and client-side feature composition approaches to improve the performance, scalability, flexibility, and extensibility of web-based communication services.

9. REFERENCES

[1] M. Weiss, B. Esfandiari, Y. Luo. Towards a classification of web service feature interactions. *Comput. Netw.* 51, 2 (Feb. 2007), 359-381. <http://dx.doi.org/10.1016/j.comnet.2006.08.003>

[2] M. Weiss, A. Oreshkin, B. Esfandiari. Method for detecting functional feature interactions of web services. *Journal of Computer Systems Science and Engineering.* v21 i4. 273-284.

[3] M. Weiss, B. Esfandiari. On feature interactions among web services. *Interl. Journal on Web Services Research.* v2 i4. 21-45.

[4] M. Weiss, A. Oreshkin, and B. Esfandiari. Invocation Order Matters: Functional Feature Interactions of Web Services. *Workshop on Engineering Service Compositions (WESC)*, 69-76, IBM, 2005.

[5] J. O'Sullivan, D. Edmond, A. ter Hofstede. What's in a Service? Towards Accurate Description of Non-Functional Service Properties, *Distributed and Parallel Databases*, 12, 117-133, Kluwer, 2002.

[6] A. Alamri, M. Eid and A. El Saddik. Classification of the state-of-the-art dynamic web services composition techniques, *International Journal on Web and Grid Services*, Vol. 2, No. 2, 2006, 148-166.

[7] N. Milanovic, M. Malek. Current solutions for web service composition, *IEEE Transaction of Internet Computing*, December, Vol. 8, No. 6, pp.51-59, 2004.

[8] M. Mrissa, D. Benslimane, Z. Maamar, C. Ghedira. Towards a semantic- and context-based approach for composing web services, *International Journal of Web and Grid Services*, Interscience Publishers, Vol. 1, Nos. 3-4, pp.268-286, 2005.

[9] Y. Badr, A. Abraham, F. Biennier and C. Grosan. Enhancing Web Service Selection by User Preferences of Non-functional Features. In *Proceedings of the 2008 4th international Conference on Next*

Generation Web Services Practices (October 20 - 22, 2008). NWESP. IEEE Computer Society, Washington, DC, 60-65.

[10] Document Object Model (DOM), World Wide Web Consortium, <http://www.w3.org/DOM/>

[11] JSR 168 Portlet Specification, <http://jcp.org/aboutJava/communityprocess/final/jsr168/index.html>

[12] Web Services for Remote Portlets Specification v2.0, <http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec-os-01.html>

[13] J. J. Garrett, Ajax: A New Approach to Web Applications, <http://www.adaptivepath.com/ideas/essays/archives/000385.php>

[14] X. Wu and V. Krishnaswamy, Widgetizing communication services, submitted to Globecom 2009

[15] A. F. Layouni, L. Logrippo, K. J. Turner, Conflict Detection in Call Control Using First-Order Logic Model Checking, *ICFI 2007*, 3-5 September 2007, Grenoble, France

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

[17] M. Kolberg, J. Buford, K. Dhara, V. Krishnaswamy, X. Wu, V. Krishnaswamy. Handling Distributed Feature Interactions in Enterprise SIP Application Services. *IEEE ICC 2009*, June 2009.

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

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

[20] E.J. Cameron, N.D. Griffith, Y.-J. Lin, M.E. Nilson, W.K. Schnure and H. Velthuisen, *A feature interaction benchmark for IN and beyond*, Proceedings of Second International Workshop on Feature Interactions in Telecommunication Software Systems, Edited by W. Bouma and H. Velthuisen, IOS Press, Amsterdam, 1994, pp. 1-23.

[21] A. Felty, K. Namjoshi, Feature specification and automatic conflict detection, in: M. Calder, E. Magill (Eds.), *Feature Interactions in Telecommunications and Software Systems*, IOS Press, Amsterdam, 2000, pp. 179-192.

[22] P. Gibson, Towards a feature interaction algebra, in: K. Kimbler, L.G. Bouma (Eds.), *Feature Interactions in Telecommunications and Software Systems V*, IOS Press, Amsterdam, 1998, pp. 217-231.

[23] M. Frappier, A. Mili, J. Desharnais, Detecting feature interactions in relational specifications, in: P. Dini, R. Boutaba, L. Logrippo (Eds.), *Feature Interactions in Telecommunication Networks IV*, IOS Press, Amsterdam, 1997, pp. 123-137.

[24] X. Wu, H. Schulzrinne, Handling Feature Interactions in the language for End System Services, *Computer Networks* 51(2), 515-535, 2007.

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

10. APPENDIX

Table 2 Call States

Call States (S_c)	Description
Trying	Indicates that a call is 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 3 Call events

Call Events (E _c)	Description
cancelled	UAS gets a CANCEL message
rejected	UAC gets 486, or 603 message
replaced	UAC receives a new INVITE with Replace
local-bye	UAC sends a BYE
remote-bye	UAS sends a BYE
error	UAC receives 481 or 408 responses
timeout	UAC does not receive any responses
redirected	UAC receives 3xx responses
answered	UAC receives 2xx responses
incoming	UAS receives INVITE
outgoing	UAC sends INVITE

Table 4 Device states

Device states (S _d)	Description
available	Audio device available
busy	Audio device busy
muted	Audio device is muted
unread-message	Message waiting light on
no-message	Message waiting light off

Table 5 Device Events

Device events (E _d)	Description
on-hook	put handset on the cradle
off-hook	put handset off the cradle
key-press	press a key on the phone (different key generate different events)
message-waiting	An event indicating unread-message
no-message-waiting	An event indicating all messages are read

Table 6 User States

User states (S _u)	Description
offline	User is offline
available	User is available for voice communication
busy	User is busy
do-not-disturb	User cannot accept any conversation requests

Table 7 User generated events

User Events (E _u)	Description
user input	click button, input text, etc.
presence	events about user status

Table 8 Voice mail states

Voicemail States (S _v)	Description
no-message	All voicemail has been read
unread-message	At least one new voicemail

Table 9 Voice mail events

Voicemail Events (E _v)	Description
open	Open and listen to a voicemail
new	A new voicemail

Table 10 Email states

Email States (S _e)	Description
read	An email is read
unread	An email is new
removed	An email is removed

Table 11 Email events

Email Events (E _e)	Description
open	Open and read an email (with attachment)
new	A new email

Table 12 Information retrieval states

IR states (S _i)	Description
unavailable	Information about a topic is unavailable
ready	Information about a topic is ready

Table 13 Information retrieval events

IR events (E _i)	Description
ir-done	information retrieval operation completed

Table 14 Widget displaying states

Display states	Description
visible	a widget is visible to users without scrolling
invisible	a widget is invisible to users without scrolling