Feature Interactions in Internet Telephony End Systems

Xiaotao Wu and Henning Schulzrinne
Department of Computer Science
Columbia University
{xiaotaow,hgs}@cs.columbia.edu

January 24, 2004

Abstract

Internet telephony end systems can offer many services. Different services may interfere with each other, a problem which is known as feature interaction. The feature interaction problem has existed in telecommunication systems for many years. The introduction of Internet telephony helps to solve some interaction problems due to the richness of its signaling information. However, many new feature interaction problems are also introduced in Internet telephony systems, especially in end systems, which are usually dumb in PSTN systems, but highly functional in Internet telephony systems. Internet telephony end systems, such as SIP soft-agents, can run on personal computers. The soft-agents can then perform call control and many other functions, such as presence information handling, instant messaging, and network appliance control. These new functionalities make the end system feature interaction problems more complicated. In this paper, we investigate ways features interact in Internet telephony end systems and propose a potential solution for detecting and avoiding feature interactions. Our solutions are based on the Session Initiation Protocol (SIP) and the Language for End System Services (LESS), which is a markup language specifically for end system service creation.

1 Introduction

A feature is an optional functionality of a system. We assume every system has a base specification. The process of adding features is to modify the base specification. Every modification is based on a specific context. For example, when a user wants to apply a feature to automatically accept an incoming call, he assumes the call setup is still pending (e.g., has not been rejected or forwarded) when the feature applies. If another feature breaks the assumption, a feature interaction happens.

Before we investigate the feature interactions in end systems, we will first emphasize two points mentioned by Pamela Zave [29]:

- Many feature interactions are undesirable, but some are desirable or necessary.
- Feature interactions are an inevitable by-product of feature modularity.

The first point indicates that we should not prohibit all feature interactions. The second point emphasizes the feature modularity which allows users to create features without knowing existing features. Feature modularity is very important for the efficiency of feature creation. Service creation environments integrate independently created features together by using feature-composition operators. For example, a simple feature-composition operator is to choose only one active feature to execute. A more complicated operator is to define the feature execution order. How to design feature-composition operators is critical for solving feature interaction problems.

In this paper, we will first introduce the related work of the paper. We then classify end system features and analyze feature interactions for each class in Section 3. For all the feature interactions, we will illustrate both desirable and undesirable feature interactions. We then discuss the way to perform feature interaction detection in Section 4 and investigate how to solve feature interactions in Section 5. Section 6 conclude the paper.

2 Related work

This paper is based on Session Initiation Protocol [23] and Language for End System Services (LESS) [28].

SIP is an IETF standard used for Internet telephony call session setup. With SIP extensions, such as SIP extensions for presence [25] and SIP extensions for instant messaging [1], SIP can also be used to perform functions beyond multimedia session setup.

There have been many efforts [21] [7] [19] [10] [30] on the service creation and execution in Internet telephony systems. The existing work mainly focus on the services on network servers. We define network services as the services executed on network servers, and end system services as the services executed on user-operated, Internet-connected devices or agents, such as Ethernet phones, software phones, and instant messengers. There are many differences between end system services and network services:

Different call models: Network services focus on establishing connections between multiple addresses, while end system services focus on instructing local applications to send media to and receive media from remote addresses.

Different developers: Network services are usually implemented by experienced programmers so the functional richness of the service language is more important than its simplicity. On the other hand, end system services are often developed by non-programmers, making simplicity a requirement.

Different user interaction: Providing services in user-operated end systems has the advantage that on-the-spot interaction with users is much easier. Network services can only interact via protocol messages and possibly media content, rather than GUIs.

Different media handling: In Internet telephony, end systems are the only entities where signaling and media flows are guaranteed to converge. This is probably the single largest architectural difference to the legacy PSTN. Thus, any service that requires interaction with user media is likely to be easier to implement in end systems.

The differences motivate us to define a language specifically for end system service creation. We named the new language Language for End System Services (LESS) [28]. LESS is extensible, can be easily understood by non-programmers and contains commands and events for direct user interactions and direct media application control. It inherits the tree-like structure of Call Processing Language (CPL) [16] and avoids the use of loops, and recursion to allow program inspection and the back-and-forth translation between a graphical and textual representation. We base our feature interaction discussion on the use of LESS as the service creation language.

LESS has four kinds of elements, namely toplevelactions, switches, modifiers and actions. Toplevelactions represent the events that can trigger the scripts. Switches check the context of the events, e.g., `address-switch` may check the caller's address of an incoming call event, and decide what actions to perform. Modifiers are used to set the parameters of the actions. Multiple LESS scripts may interact with each other when one toplevelaction triggers multiple different actions. For every toplevelaction, LESS has the tree-like structure, there are no feature conflicts within a single LESS tree.

Prior to the work in this paper, Jonathan Lennox had written a technical report on implementing intelligent network (IN) services with the Session Initiation Protocol [14]. Jonathan's report describes service implementations on the signaling protocol level. In this report, we focus on service creation on the service description language level, a higher level than the work in Jonathan's report. Jonathan's report can serve as the base to design the LESS based service creation environment. Jonathan Lennox has another paper on feature interactions in Internet telephony systems [15]. In his paper, he discussed the differences of the feature interactions between PSTN network and Internet telephony systems. His paper also provides a case-by-case study on some feature interaction problems in Internet telephony systems. His paper can help us to better understand the feature interactions in

Internet telephony. However, the paper mainly focus on services on signaling servers in the network. There is no in-depth discussion on feature interactions in end systems, which we will discuss in this paper.

3 Classify end system feature interactions

The article by E.J. Cameron et al. "A Feature Interaction Benchmark for IN and Beyond" [6] classified feature interaction problems into three dimensions, namely customer-system dimension, single-multiple user dimension and single-multiple component dimension, and five categories, namely SUSC (Single-User-Single-Component), SUMC (Single-User-Multiple-Component), MUSC (Multiple-User-Single-Component), MUMC (Multiple-User-Multiple-Component), and CUSY (CUstomer-SYstem) interactions.

The customer-system dimension distinguishes the features involving customer call processing from the features involving system operations, administration and maintainance (OA&M), such as billing. End system services usually only involves customer call processing features. We will not discuss system maintainance features in this paper.

The single-multiple user dimension distinguishes the features involving single user from the features involving multiple users. In most cases, the call control services in end systems only deal with single user features because at a given time, an end device can only be used by one user. However, an end system may control other devices. For example, an end system may use MGCP[3]/Megaco[**?**] or SIP third party call control [22] to control media applications in another device. An end system can also control network-connected appliances. If multiple users try to control the same devices at the same time, feature conflicts may happen.

The single-multiple component dimension distinguishes the features involving single component from the features involving multiple components. The components could be end devices, network servers, or network-connected appliances. End system call control services are usually single component services. However, end system services may interact with network services and cause multiple components to be involved in feature interactions.

Based on the above discussion on feature dimensions, end system call control services usually experience SUSC and SUMC (when interacting with network services) feature interactions. Network appliance control services usually experience SUMC and MUMC feature interactions. There are usually no MUSC and CUSY feature interactions in end systems.

At the time Cameron et al. classified feature interactions, the presence related services and network appliance control were not considered as part of the telecommunication services. Cameron's article focuses only on call control services. In Internet telephony end systems, presence related services and network appliance control are very useful services. Because presence related services, network appliance control, and call control services have different characteristics in terms of feature interactions, we will not exactly follow the classification defined in Cameron's article. Rather, we divide the services into three categories: call control services, presence and event based services, and other end system services, including network appliance control. In each category, we use the methods in the article by Cameron et al. to further classify the feature interactions.

3.1 End system call control feature interactions

Follow the above discussions, we investigate SUSC and SUMC feature interactions separately for end system call control services. The investigation is based on LESS.

3.1.1 SUSC feature interactions

As discussed before, the feature interactions in end system call control services are usually SUSC feature interactions, as long as we do not consider the interactions with network services. Multiple scripts belonging to one user and handling one device may cause SUSC feature interactions.

As indicated in Section 2, feature interactions may happen when multiple actions are triggered at the same time. To investigate feature interactions, we should first check all the possible actions LESS scripts can perform. The call control actions can be signaling or non-signaling actions, and can be in different call stages. Table 1 shows the actions.

	Signaling actions	Non-signaling actions
Incoming call setup	accept reject redirect	alert (all stages) log (all stages) mail (all stages)
Outgoing call setup	call	
Mid-call stage	transfer hold unhold mute unmute	
Call termination	disconnect	

Table 1: Call control actions

For signaling actions, the actions belonging to the same call stage usually conflict with each other. For example, an end system can only choose one of 'accept', 'reject', and 'redirect' to handle an incoming call. The actions at different call stages can also interact with each other. For example, accepting a call then transferring the call is a desirable interaction, however, rejecting a call then transferring the call is an undesirable interaction. The non-signaling actions will not conflict with the signaling actions.

When we check feature interactions between two actions, we need to define the execution order of the actions and check possible interactions in different orders. For example, if we want to check the interactions between action A and action B, we first check the situation where A is performed before B. The checking consists of two parts: one is to check whether action A's result changes or conflicts with the context precondition of action B, the other is to check whether action B's result changes the expected result of action A. We then check the interactions with a different execution order with action B performed first and do the same checking. We define the context precondition and expected result of each action in Table 2.

	precondition	expected result
accept	The call setup is pending. The audio device is available.	The call setup is finalized. The communication session is setup. The audio device is occupied.
reject	The call setup is pending.	The call setup is finalized.
redirect	The call setup is pending.	The call setup is finalized on the current end system.
call	The audio device is available.	If the callee side accepts the call, a communication session is setup and audio device is occupied.
transfer	There is an existing session and it means the audio device may be occupied.	If the action succeeds, the session is end in the current end system.
hold	There is an existing session.	There is no media transmission for the session but the session will still be alive.
unhold	There is an existing session.	Enable the media transmission for the session. The session will be alive.
mute	There is an existing session.	Disable the user's audio input. The session will still be alive.
unmute	There is an existing session.	Enable the user's audio input. The session will still be alive.
disconnect	There is an existing session.	The session is terminated.

Table 2: The context assumption and expected result of call control actions

We further investigate the cause of feature interactions and find five kinds of interactions.

The first is action conflicts, such as the conflicts between the `accept` and the `reject` actions.

The second is attribute conflicts, for example, two scripts both perform the `redirect` action, but to different locations. We treat LESS modifiers as action attributes. If the modifiers of two actions are different, they conflict with each other.

The third is avoidable conflicts, which can be avoided by putting restrictions on LESS language design. For example, we can restrict LESS to not allow mid-call actions (such as `transfer`) and call termination actions (such as `disconnect`) as the subsequent actions of the `reject` and `redirect` actions. With the restriction, there is no chance for `reject` and `redirect` to interact with `transfer` and `disconnect`. The avoidable conflicts are in fact special cases of the first kind of conflicts.

The fourth is resouce competing conflicts. Multiple actions may compete the resource usage. For example, if there is only one audio device in a device, two calls both using the audio device will cause conflicts. Accepting an incoming call and making an outgoing call to another address at the same time cause this kind of conflicts.

We call the last kind of interactions 'enabling' interactions. It happens when one action makes another action possible. This kind of interactions are desirable. For example, `accept`ing an incoming call enables the action to `transfer` the call. Table 3 shows the conflict table for handling an `incoming` toplevelaction. The assumption of the table is that there is only one audio device in the end device, which is the most common case for end devices.

Note that the table is not a symmetric table, row `m`, column `n` and row `n`, column `m` do not have the same value. In the table, we define the row actions are performed before the column actions, except when the row action and the column action are the same. For example, row 1, column 5 means '`accept` then `transfer`'.

Mid-call and termination actions may apply to an existing call, instead of the call in an incoming call event. For example, call `A` already established, another call comes in, a script may transfer the existing call, then accept the incoming call. We append a `*` to the `transfer` action to represent this situation. For example, the `transfer*`, `hold*`, `unhold*`, `mute*`, `unmute*`, and `disconnect*` are used to handle an existing call, not to handle the call in the `incoming` call setup request. The `call*` is used to make a new call.

The `N/A` in the table means the indicated situation should not happen. For example, the `transfer` action cannot be used alone for handling an incoming call because it's for mid-call handling, not for call setup handling.

3.1.2 SUMC feature interactions

A user's service scripts can be hosted on his end devices, as well as the signaling servers in the network. The scripts in different places may interact with each other. For example, if the scripts on the proxy server reject all the calls, the scripts on the end devices will never get executed. If the proxy server proxies calls to all of the user's end devices, and one of the user's end devices (e.g., voicemail server) automatically accept calls immediately, the other end devices of the user will not be able to accept incoming calls.

We divide the SUMC feature interactions into two categories, one is end system–proxy server feature interactions, the other is end system–end system feature interactions.

	accept	reject	redirect	call*	transfer	transfer*
accept	A(media)	C	C	R	E	R
reject	C	A(reason)	C	-	(C)	-
redirect	C	C	A(location)	-	(C)	-
call*	R	-	-	A(location)	N/A	R
transfer*	E	-	-	E	N/A	A(location)
hold*	E	-	-	E	N/A	C
unhold*	R	-	-	R	N/A	C
mute*	R	-	-	R	N/A	C
unmute*	R	-	-	R	N/A	C
disconnect*	E	-	-	E	N/A	C
	hold	**hold***	**unhold**	**unhold***	**mute**	**mute***
accept	E	R	E	R	E	R
reject	(C)	-	(C)	-	(C)	-
redirect	(C)	-	(C)	-	(C)	-
call*	N/A	R	N/A	R	N/A	R
transfer*	N/A	C	N/A	C	N/A	C
hold*	N/A	-	N/A	C	N/A	-
unhold*	N/A	C	N/A	-	N/A	C
mute*	N/A	-	N/A	C	N/A	-
unmute*	N/A	C	N/A	-	N/A	C
disconnect*	N/A	C	N/A	C	N/A	C
	unmute	**unmute***	**disconnect**	**disconnect***		
accept	E	R	E	R		
reject	C	-	(C)	-		
redirect	C	-	(C)	-		
call*	N/A	R	N/A	R		
transfer*	N/A	C	N/A	C		
hold*	N/A	C	N/A	C		
unhold*	N/A	-	N/A	C		
mute*	N/A	C	N/A	C		
unmute*	N/A	-	N/A	C		
disconnect*	N/A	C	N/A	-		

-: no interaction, A: attribute conflict, C: action conflict,
(C): avoidable conflict, E: enabling, R: resource competition

Table 3: Call control action conflict table for handling `incoming` toplevelaction

End system–proxy server feature interactions End system–proxy server feature interactions are caused by the CPL scripts on proxy servers interacting with the LESS scripts on end systems. There are only three signaling actions for the CPL scripts running on proxy servers, namely `proxy`, `redirect` and `reject`. Every action may interact with the actions on end systems. For incoming calls, the proxy server scripts are executed before the end system scripts. For outgoing calls, the end system scripts are executed before the proxy server scripts. For incoming calls, the proxy server scripts may interact end system scripts in two ways: blocking the execution of end system scripts or overlapped with the end system scripts. For outgoing calls, the proxy server scripts may modify the bahaviour of end system service scripts. Table 4 shows the possible interactions.

end / server	reject	redirect	proxy
accept	blocking	blocking	blocking
reject	overlapping	blocking	blocking
redirect	blocking	blocking/overlapping	blocking
call	modifying	modifying	-
transfer	modifying	modifying	-
disconnect	-	-	-

Table 4: Interactions between end system services and proxy server services

End system–end system feature interactions End system–end system feature interactions involve multiple end devices belonging to one user. This kind of feature interactions are the most complicated feature interactions for end system services. It sometimes also involves the services scripts running on proxy servers. For example, if a proxy server does sequential forking [23] and has the voicemail server as the last one to fork to, the auto-accept script running on voicemail server will not affect the other end devices' behavior. However, if the proxy server does paralle forking, the inappropriate timeout value of the auto-accept script running on voicemail server may make the other end devices not able to accept incoming calls. When we try to detect the end system–end system feature interactions, we must also take service scripts on proxy servers into consideration. The action conflict between two end systems can also be expressed as a table (Table 5).

The table shows that for outgoing calls, mid-call and call termination actions will not interact with each other because every end system is independent to each other. During the call setup stage, due to the forking proxy in Internet telephony systems, multiple end devices may all get the incoming call setup at the same time. If they all try to accept the call, a feature conflict happens. If they try to redirect the call setup to different locations, a feature conflict also happens.

	accept	reject	redirect	call	transfer	disconnect
accept	C	-	-	-	-	-
reject	-	-	-	-	-	-
redirect	-	-	C	-	-	-
call	-	-	-	-	-	-
transfer	-	-	-	-	-	-
disconnect	-	-	-	-	-	-

C: action conflict

Table 5: Call control action conflict between two end systems

3.2 Feature interactions for end system presence and event-based services

For event-based services, we base our discussion on the SIP event notification architecture [20].

An end system can watch other users' presence status, and can notify the others of his own presence status. The feature interactions can be SUSC or SUMC interactions, depending on whether the Presence User Agent (PUA) and the Presence Agent (PA) [25] are co-located together or not. If PUA and PA are co-located, we need to deal with SUSC feature interactions, otherwise, SUMC interactions.

The actions belonging to the event based services can be divided into two parts. One is for incoming subscription handling, such as `accept` a subscription, or `deny` a subscription. The other is to send outgoing messages, such as `subscribe` and `notify`. Feature conflicts can also be categorized into action conflicts and action attribute conflicts.

3.2.1 SUSC feature interactions

For an incoming subscription, the `accept` and the `deny` actions conflict with each other. `accept` actions with different attributes, such as different expiration time, conflict with each other. `deny` actions with different reasons conflict with each other. `subscribe` and `notify` actions do not conflict with the other actions, but they may cause action attribute conflicts. Two `subscribe` actions conflict if they have the same destination, the same event package, but different in the other attributes, such as expiration time. Two `notify` actions conflict if they have the same destination, the same event package, but different events.

3.2.2 SUMC feature interactions

If a PUA and a PA are separated, the PA usually resides on a network server, for example, it can co-locate with a SIP proxy server. The PUA will reside in users' end devices. A

PUA may use PUBLISH [18] request to update its status in the PA, and use XCAP [24] to retrieve and modify watcher and presentity information. [1]

For an incoming subscription, PA can decide whether to `accept` or `deny` it. PA can also set the subscription status as 'pending', and send a notification to the PUA about the watcher-list changes. Once the PUA get the watcher-list notification, the PUA will use XCAP to update the watcher-list document on the PA to authorize the subscription. A user may have multiple PUAs. The PA and all the PUAs will involve in incoming subscription handling and feature interactions may happen between them. For an incoming subscription, if a PA and all the PUAs make different decisions, for example, the PA accepts a subscription but one PUA denies the subscription, an action conflict happens.

3.3 Other end system services

An end system can perform many other communication functions, such as instant messaging and network appliance control. There will be many more new communication functions developed in end systems. In this paper, we only focus on the instant messaging and network appliance control for feature interaction analysis.

3.3.1 Feature interactions for instant messaging

For instant messaging, there is only one LESS action defined, namely `im`, which is used to send an outgoing message. If we don't concern the content of the message, there is no conflict between multiple `ims`. However, the content of an `im` may have special meanings in some circumstances. For example, if we use SIP MESSAGE to perform shared web browsing [27], the message content will be used to convey URL information. Two `ims` with different content may conflict with each other. The user should decide the order of the `ims`, or choose one `im` and discard the other.

Instant messaging may also experience SUMC feature interactions. One incoming message may be sent to multiple contacts of a user. If more than one contact can automatically send a message back, SUMC feature interactions may happen. There is not much difference between SUSC and SUMC instant messaging feature interactions. Both interactions depend on whether the content of the messages conflicts with each other.

3.3.2 Feature interactions for network appliance control

Services related to network appliance control can be very complicated. Different sensors may trigger different control actions. The actions performed by multiple network appliances may conflict with each other. For example, turning on air conditioner to lower the temperature and turning on heater to make the room warmer conflict with each other. This kind of feature interaction has been detailed in [13].

[1] Watcher information contains the entities watching the user's status, presentity information contains the entities watched by the user.

In our paper, we focus on multimedia communication related services, and consider the network appliance control as an additional part for a more convenient and comfortable communication environment. In this paper, we will not cover all the sensors and events that may trigger control commands. We limit the events to the toplevelactions we have already discussed above, which are all communication related. The only sensor generated events we are going to discuss is the events triggered by location sensors. The reason we only choose location sensors is because location information is critical to communications, especially to emergency services.

We will not consider the situation that the result of one control action triggers another control action. In addition, we will not consider the situation that multiple appliances interact with each other. For example, we will not consider the interactions between a heater and an air conditioner. With these limitations, we can detect feature interactions by inspecting service scripts.

If multiple scripts try to control a network appliance to perform different actions, feature interactions may happen. Different network appliances may have different interactions. For example, to control a lamp, `power on` and `power off` conflict with each other. To control a stereo, `power on` and `tune` the stereo to a specific channel do not conflict with each other. To analyze feature interactions, we need to first identify the network appliances we want to control. We then need to build the context precondition and expected result table for the device control actions. Based on the table, we can build the feature interaction table for the device. In this paper, we choose two network appliances to as sample devices to analyze. One is a lamp, controlled through a X10 controller, the other is a stereo, controlled through a sLinke [11] controller.

The commands for lamp can be `power on`, `power off`, `dim`, and `bright`. The commands for stereo can be `power on`, `power off`, `volume up`, `volume down`, `tune`, `play`, and `stop`. Table 6 shows the context assumption and expected result of lamp control actions. Table 7 shows the context assumption and expected result of stereo control actions.

	precondition	expected result
power on	-	The lamp is on.
power off	-	The lamp is off.
dim	The lamp is on.	The lamp is dimmer. The lamp is still on.
bright	The lamp is on.	The lamp is brighter. The lamp is still on.

Table 6: The context assumption and expected result of lamp control actions

For both lamp and stereo, we define `power on` and `power off` as basic control actions, and the other actions as additional control actions. All the additional control actions require appliances to be powered on. So, `power on` does not conflict with all the addi-

	precondition	expected result
power on	-	The stereo is on.
power off	-	The stereo is off.
tune	The stereo is on.	Play radio on a specific channel. The stereo is still on.
play	The stereo is on.	Play a specific CD. The stereo is still on.
stop	The stereo is on.	Stop playing CD. The stereo is still on.
volume up	The stereo is on.	Play louder. The stereo is still on.
volume down	The stereo is on.	Play in a lower volume. The stereo is still on.

Table 7: The context assumption and expected result of stereo control actions

tional control actions but `power off` will conflict with them. `power on` and `power off` conflict with each other. `dim` and `bright` conflict with each other. `volume up` and `volume down` conflict with each other. `play` and `stop` conflict with each other. `tune` and `play` conflict with each other because `tune` tries to play radio but `play` tries to play CD. Table 8 and Table 9 shows the conflict tables. The conflict tables are based on the assumption that multiple scripts trying to control the same device at the same time.

	power on	power off	dim	bright
power on	-	C	E	E
power off	C	-	C	C
dim	-	C	A	C
bright	-	C	C	A

A: attribute conflict, C: conflict, E: enabling

Table 8: Interactions between lamp control actions

Network appliance control intrinsically involves multiple components, one is the controller, the others are the appliances. Network appliance control may have SUMC and MUMC feature interactions.

Both SUMC and MUMC feature interaction detection are based on feature interaction tables. However, different type feature interaction detections happen in different stages. SUMC feature interaction detection happens in service creation stage because one user can access all of his own service scripts. But MUMC feature interaction need to be detected during service execution stage because one user cannot access another user's service scripts.

We should use different ways to solve SUMC and MUMC interactions. For SUMC

	power on	power off	volume up	volume down	tune	play	stop
power on	-	C	E	E	E	E	E
power off	C	-	C	C	C	C	C
volume up	E	C	A	C	-	-	-
volume down	E	C	C	A	-	-	-
tune	E	C	-	-	A	C	-
play	E	C	-	-	C	A	C
stop	-	-	-	-	-	C	-

A: attribute conflict, C: conflict, E: enabling

Table 9: Interactions between stereo control actions

interactions, the owner of the scripts can decide the way to solve the conflicts.

For MUMC interactions, since the service scripts belong to different users, not a single user can solve the conflicts. If all the users access the device through the same appliance controller, (e.g., a network appliance gateway), the policies residing on the network appliance gateway may help to solve the conflicts. For example, the administrator of the gateway may define the priority of the users. The actions performed by the user with higher priority may override the actions performed by the user wihh lower priority. If the users access the device through different controllers, the communication between the device controllers is required to solve feature conflicts.

3.4 Feature interactions across classes

The toplevelactions in one category may trigger actions in another category. For example, an incoming call may cause an end system to send an instant message and lower the volume of a stereo. The actions in different classes will not conflict with each other directly. However, some actions may trigger additional actions and cause feature conflicts. For example, the `call` action may invoke the scripts handling the `outgoing` toplevelaction. The `outgoing` toplevelaction handling may perform network appliance control actions.

To illustrate the feature interactions across classes, we define three service scripts. The first one is 'when I am in IRT lab, `power on` the lab stereo'. The second one is 'when I am in IRT lab, `call` my home'. The third one is 'when making an `outgoing` call, `power off` the stereo'. There is no conflict when composing the first two or the last two scripts. However, composing all three scripts together will cause a conflict. In this example, the `power off` is a subsequent action caused by the `call` action. Though `call` and `power on` do not conflict with each other, the subsequent action of `call` may conflict with `power on` and cause feature conflict. This kind of feature conflicts require us to do an additional step to check the feature conflicts involving subsequent actions. There is no need to have another feature interaction table because the conflict checking still ends at inspecting the conflicts of the actions in the same category.

4 Feature interaction detection

There are many existing languages and notations for describing telecommunication services, such as Use Case Map (UCM) [2], Chisel Representation Employing Systematic Specification (CRESS) [26], LOTOS (Language Of Temporal Ordering Specification) [5], Promela [4], and SDL (Specification and Description Language) [26]. Accordingly, there are some tools designed based on the formal languages to detect feature interactions. For example, Spin [9] is used to analyze logical consistency of concurrent systems described by Promela. The feature interaction detection in UCM and CRESS is done by compiling them into LOTOS or SDL, and use the model checking tools for LOTOS or SDL for feature interaction detection. For LESS and CPL features, we can also convert them into a formal language and use the formal model checking tools to detect interactions. However, there are several disadvantages in doing that. First, CPL and LESS are designed to sacrifice some functionalities to make the language simpler. For example, there is no loop in CPL and LESS scripts. It is used to define a small set, but commonly used features. The formal languages like LOTOS and Promela are used to describe all possible features, which makes the language much more complex than CPL and LESS and the feature interaction detection process can be complicated. Second, one way of solving feature interactions between multiple LESS scripts is to merge multiple scripts into one script. LESS scripts can be merged since they have tree-like structure. The merge process cannot be done in formal languages.

There is an existing research work on detecting CPL script feature interaction without any translation [17]. However, that work only applies to services on network servers, and the feature interaction analysis is incomplete. What we need to do is to build a complete LESS feature interaction detection system. One possible solution is to develop an algorithm that can traverse the tree structure of multiple scripts and find the possible interactions. Usually, the interactions happen at the intersection of multiple conditions. We then check whether the actions in different scripts interact with each other or not. For example, we have a service script "`reject` a call if the time is between 12:30PM to 1:00PM because I am at lunch", and we have another service script "Automatically `accept` a call if it's from sip:hgs@cs.columbia.edu". The interaction of the conditions between the two scripts is "A call from sip:hgs@cs.columbia.edu between 12:30PM and 1:00PM". The action `reject` and `accept` conflict with each other and there is an unwanted feature interaction between these two scripts.

5 Resolving feature interactions

Compared to the work on interaction detection, the work is much less on solving feature interactions. This is because how to solve a interaction highly depends on the requirement of features and the expectation of users. It is hard to find a general solution for feature interactions. However, there are still some architectural approaches trying to deal with the feature interactions in general. For example, the pipe-and-filter architecture and Distributed Feature Composition (DFC) [12].

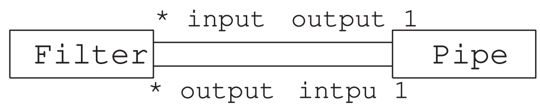

Figure 1: Pipe and filter architecture

Figure 1 shows the basic pipe-and-filter architecture. A filter can have many inputs and outputs. A pipe connect one of the outputs of a filter to one of the inputs of another filter. The features are applied in the filters. The order of the filters defines the precedence of the features and may help to solve the feature interaction problems.

DFC is a well-designed pipe-and-filter architecture. The key novelty of DFC is the choice of an architecture in a dynamic pipe-and-filter style. The order of the filters can be dynamically changed based on the service usage so DFC can handle non-linear usages.

There are some other architectural approaches, such as agent-based architecture [8]. We will not describe these approaches in detail. We will just note one thing that is common to these approaches. All these approaches assume the features are carefully designed and modularized. In another words, they assume the service creators are professional telecommunication service designers. However, for end system services, usually the service creators are non-professional, which means they will not create modularized features. A new created feature may overlap with the existing features or can be divided into multiple modules if it was created by professionals. The architecture approaches are not suitable for the 'ill-formatted' features in an end system. However, in a multi-user environment, since one user cannot access the others' scripts, the architecture approaches are still appropriate for feature conflict avoidance.

For LESS based services, due to the tree-like structure of LESS, we consider it simpler and more efficient to design an algorithm to merge multiple scripts into one script. At any given time, there is only one active LESS script at a device. We will still keep the original scripts for users to modify them independently. By this way, service execution gets more efficient because it only needs to go through one decision tree to perform services, and we can still ensure the service creation efficiency because users can still edit every script separately.

If we regard the merging operation as one way of doing feation composition, the approaches to perform the operation can be considered as composition-operators. There are many approaches of merging two scripts into one. For example, based on the feature interaction tables we defined before, we can sequentially execute multiple actions, or execute multiple actions in parallel. We can also keep one action and abandon the others. We need further investigation on the composition-operators.

6 Conclusion

This paper investigates feature interactions in end systems. It first classifies the interactions, then proposes approaches for detecting feature interaction and solving the interactions. We plan to develop algorithms for the proposed approaches.

References

[1] Session initiation protocol (SIP) extension for instant messaging. RFC 3428, Internet Engineering Task Force, December 2002.

[2] Daniel Amyot. Use case maps as a feature description notation. In *FIREworks Feature Constructs Workshop*, May 2000.

[3] M. Arango, A. Dugan, I. Elliott, C. Huitema, and S. Pickett. Media gateway control protocol (MGCP) version 1.0. RFC 2705, Internet Engineering Task Force, October 1999.

[4] A. Basu, G. Morrisett, and T. von Eicken. Promela++: A language for constructing correct and efficient protocols. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, page 455, San Francisco, California, March/April 1998.

[5] Ed Brinksma, G. Scollo, and C. Steenbergen. LOTOS specifications, their implementations and their tests. In *Protocol Specification Testing and Verification VI*, pages 349–360. IFIP, 1987.

[6] E. J. Cameron, N. Griffeth, Y. Lin, Margaret E. Nilson, William K. Schure, and Hugo Velthuijsen. A feature interaction benchmark for IN and beyond. In *Feature Interactions in Telecommunications Systems*, pages 1–23, Amsterdam, Netherlands, 1994.

[7] John de Keijzer, Douglas Tait, and Rob Goedman. JAIN: a new approach to services in communication networks. *IEEE Communications Magazine*, 38(1), January 2000.

[8] N. Griffeth and Hugo Velthuijsen. The negotiating agents approach to runtime feature interaction resolution. *Feature Interactions in Telecommunications Systems, IOS Press*, pages 217–235, 1994.

[9] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[10] R. Stubbs I. I. The intelligent network - changing the face of telecommunications. *Proceedings of the IEEE*, 79(1):7–20, 1991.

[11] Nirvis Inc. Slink-e. http://www.nirvis.com/slink-e.htm.

[12] Michael Jackson and Pamela Zave. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Transactions on Software Engineering*, August 1998.

[13] Mario Kolberg, Evan H. Magill, and Michael Wilson. Compatibility issues between services supporting networked appliances, November 2003.

[14] J. Lennox, Henning Schulzrinne, and Thomas F. La Porta. Implementing intelligent network services with the session initiation protocol. Technical Report CUCS-002-99, Columbia University, New York, New York, January 1999.

[15] Jonathan Lennox and Henning Schulzrinne. Feature interaction in Internet telephony. In *Feature Interaction in Telecommunications and Software Systems VI*, Glasgow, United Kingdom, May 2000.

[16] Jonathan Lennox, Xiaotao Wu, and Henning Schulzrinne. CPL: a language for user control of Internet telephony services. Internet draft, Internet Engineering Task Force, August 2003. Work in progress.

[17] Masahide Nakamura, Pattara Leelaprute, Ken ichi Matsumoto, and Tohru Kikuno. Detecting script-to-script interactions in call processing language. In *Seventh International Workshop on Feature Interactions in Telecommunications and Software Systems*, June 2003.

[18] A. Niemi. Session initiation protocol (SIP) extension for event state publication. Internet Draft draft-ietf-sip-publish-02, Internet Engineering Task Force, January 2004. Work in progress.

[19] Parlay. Parlay framework api. http://www.parlay.org/specs/index.asp.

[20] A. B. Roach. Session initiation protocol (sip)-specific event notification. RFC 3265, Internet Engineering Task Force, June 2002.

[21] J. Rosenberg, J. Lennox, and Henning Schulzrinne. Programming Internet telephony services. *IEEE Network*, 13(3):42–49, May/June 1999.

[22] J. Rosenberg, J. Peterson, Henning Schulzrinne, and G. Camarillo. Best current practices for third party call control in the session initiation protocol. Internet Draft draft-ietf-sipping-3pcc-06, Internet Engineering Task Force, January 2004. Work in progress.

[23] J. Rosenberg, Henning Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: session initiation protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[24] Jonathan Rosenberg. The extensible markup language (XML) configuration access protocol (XCAP). Internet Draft draft-ietf-simple-xcap-01, Internet Engineering Task Force, October 2003. Work in progress.

[25] Jonathan Rosenberg. A presence event package for the session initiation protocol (SIP). Internet draft, Internet Engineering Task Force, January 2003. Work in progress.

[26] Kenneth Turner. Representing new voice services and their features. In *Feature Interactions in Telecommunication Networks*, Ottawa, Canada, June 2003. IOS Press.

[27] X. Wu and Henning Schulzrinne. Use SIP MESSAGE method for shared web browsing. Internet draft, Internet Engineering Task Force, November 2001. Work in progress.

[28] Xiaotao Wu and Henning Schulzrinne. Programmable end system services using SIP. In *Conference Record of the International Conference on Communications (ICC)*, May 2003.

[29] Pamela Zave. An experiment in feature engineering. In *Programming Methodology*, February 2003.

[30] Pietro Zolzettich and Arye R. Ephrath. Customized service creation: A new order for telecommunications services. *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, pages 1014–1019, 1992.