Methods for Designing SIP Services in SDL with Fewer Feature Interactions

Ken Y. Chan and Gregor v. Bochmann
School of Information Technology and Engineering (SITE),
University of Ottawa
P.O.Box 450, Stn. A, Ottawa, Ontario, Canada. K1N 6N5
{kchan,bochmann}@site.uottawa.ca

Abstract: This paper describes methods for implementing telephony services in SIP with fewer traditional feature interactions. A formal SDL model of SIP and its services has been derived from published SIP specifications for verification and validation. It is known that the SIP RFC describes only the protocol specification. The specifications of SIP services and additional service features are informal and can only be found in various IETF drafts. Nevertheless, the service designers are still faced with new feature interaction problems. These new feature interactions are unique to SIP because SIP has flexible signaling features, such as request forking and dynamic assignment of contact addresses, which have both cooperative and adversarial side effects on each other. This paper also describes an extension to the classical feature interaction taxonomy, which is used to associate the causes, effects/symptoms with the preventive measures of the new and traditional feature interactions. Finally, SIP services can be designed and implemented without certain feature interactions by following certain design rules which are based on the knowledge deduced from the verification.

Introduction

In the telecommunication industry, many people believe that Internet telephony would be the next major market for many carriers. Internet telephony is defined as the provisioning of telephone-like services over the Internet. With many Internet users having already embraced voice chat and webcam conferencing, Internet telephony, which also encompasses convergence of existing Internet and PSTN voice services, is the natural step forward. Although H.323 [14] has been around for a decade or so as the de facto signaling protocol over data networks, SIP (Session Initiation Protocol), which is being standardized under IETF RFC 2543 [3] and backed by powerful industry partners such as Cisco, Microsoft, Nortel, Sonus, and DynamicSoft, is gaining a lot of momentum in establishing itself as the voice over IP (VoIP) signaling protocol of choice. SIP is designed based on the general philosophy of IETF that protocols remain open and support decentralization of control over applications in an Internet environment. SIP is generally better developed than other telephony signaling protocols in areas such as distributed call control between end systems and inter-provider communications. These areas are known to be prone to feature interactions (FI's), thus many researchers believe feature interactions are more pronounced in Internet telephony than PSTN (Public Switched Telephone Network) [1].

This paper is organized as follows: Section 1 presents an enhanced classification of FI's for POTS (Plain Old Telephone Services) and an overview of the SIP protocol. Section 2 gives a detailed description on the SDL specification of SIP services. Section 3 discusses the semi-automated process of verifying traditional FI's that may exist in SIP. Section 4 presents a method for preventing traditional FI's in SIP. Section 5 shows new FI's that are potentially introduced by SIP and their mapping to the extended classification system. Finally, the paper ends with a conclusion and discussion of future work in Section 6.

1 Enhanced Classifications of Feature Interactions and Overview of SIP

Features in a telecommunication system are packages of incrementally added functionality that provide services to subscribers or administrators. In this paper, we do not make any distinction between feature and service; the two terms would be used interchangeably. There are many definitions of *feature interactions (FI's)*. In general, FI's are defined as undesirable side effects caused by interactions between features and/or their environment. The research community of FI's has generally categorized FI's by the nature and the cause of the interaction.

We believe the latter categorization by cause may be useful in classifying FI's, but the classification does not lead explicitly to a general scheme for detecting or resolving FI's. A classification process is most useful if it leads to methods of detecting, preventing, or resolving FI's. While understanding the causes of interactions is helpful in formulating potential resolution policies to prevent FI's, the causes of interactions are conceptually abstract from the actual feature specifications; for example, a resource contention scenario like presenting a voice greeting and call waiting tone to the same user may or may not be considered as a FI. Also how does a service designer know which "resource" to declare for checking resource contention? Thus, the causes cannot be construed as the most efficient means to derive methods for detecting FI's. We propose adding another type of categorization to the taxonomy of FI's, which is categorization by symptom or effect. This new categorization enables FI's to be associated to some of the well-known distributed system properties. As a result, we believe existing verification and validation techniques or tools for these distributed system properties may be used to facilitate detection of FI's, especially in IP Telephony such as SIP.

1.1 The Nature of Feature Interactions

The traditional categorization includes three dimensions: by kind, by the number of users, and by the number of network components. The combinations of these dimensions produce five types of FI's: single-user-single-component (SUSC), single-user-multiple-component (SUMC), multi-user-single-component (MUSC), multi-user-multi-component (MUMC), and customer-system (CUSY) FI's [2]. Detail discussion on this categorization can be found in [2].

1.2 Causes of Feature Interactions

Categorizing FI's by the cause has been suggested in [2]. The suggested causes are violation of feature assumptions, limitations of network support, and intrinsic distributed system problems. Instead of discussing these general causes, we consider below four detailed causes that are also mentioned in [2] and are specializations of the above general causes.

Resource contention (RSC) is definitely a well-known distributed system issue. It is defined as the attempt of two or more nodes to access the same resource. In the context of FI's, an accessing node may be the feature running on a network component. Resource is an abstract term; it needs not be a physical entity. For example, Call Waiting (CW) and Three Way Calling (TWC) in POTS may be considered as contending for the flash hook signal simultaneously.

Violation of Feature Assumptions (VFA) is defined as a set of assumptions that telecommunication features, much like any software features, operate under, and are

designed based on. There are many types of assumptions. One example from [2] that is particularly worth examining is assumptions about data availability. Features such as Terminating Call Screening (TCS) cannot function properly without the caller-id of the caller. If the caller-id were made unavailable in the case of Private Call (PC), TCS would permit the call request. In the case of Operating Assisted Call (OAC) and OCS, the hiding of the original caller-id by OAC allows the call to bypass the OCS restriction.

Resource Limitation (RSL) is also mentioned in [2], and is definitely a common cause of FI's in POTS, because most of the traditional POTS end-user devices (e.g. basic phones) have limited user interfaces and computational power. If the classical example of resource limitation were revised (e.g. Call waiting and Three Way Call), the confusion of the flash hook signal can also be attributed to the lack of separate buttons for the two features; thus that feature interaction can be also classified under resource limitation.

Timing and Race conditions (TRC) are also mentioned in [2], and are common, particularly in distributed real-time systems like POTS running on PSTN. A race condition is defined as a condition where two or more nodes have non-mutually exclusive read and modify access to a shared resource. As a result, the value or status of the shared resource may be undefined (different values in the same context) depending on the timing of the accesses between the nodes. A classical example is between Automatic Callback (AC) and Automatic Recall (AR). The AR feature makes a call on behalf of the original caller when the callee becomes idle again. Both AC and AR features depend on the busy signal of the callee. The timing of the busy signal received by the two features would either allow one of the calls to go through on the single line of the POTS phone, or reject both call requests because both ends are calling each other simultaneously.

1.3 Effect of Feature Interactions

This categorization, which is called categorization by effect, focuses on the effects or symptoms of the problems that are more specific and can be used to narrow down the search in detecting FI's. Common distributed system properties such as livelock, deadlock, fairness, and non-determinism are well-defined concepts. Tools such as Telelogic Tau [8] can verify some of these properties in a system. The category called incoherent interactions has also been introduced in the following subsection. The following diagram (Figure 1) illustrates the graphical representation of the extended taxonomy, which is called the "feature interaction tree" (FIT). Since a feature interaction can be associated to just one kind of interaction, but one or more cause-effect category pairs, a feature interaction may be defined as a spanning sub-tree within the following feature interaction tree. Two or more FI's may overlap with each other on the FIT (Figure 1). In addition, each effect-type interaction (e.g. livelocking (LLCK)) is associated to one or more preventive measures (not shown in the diagrams).

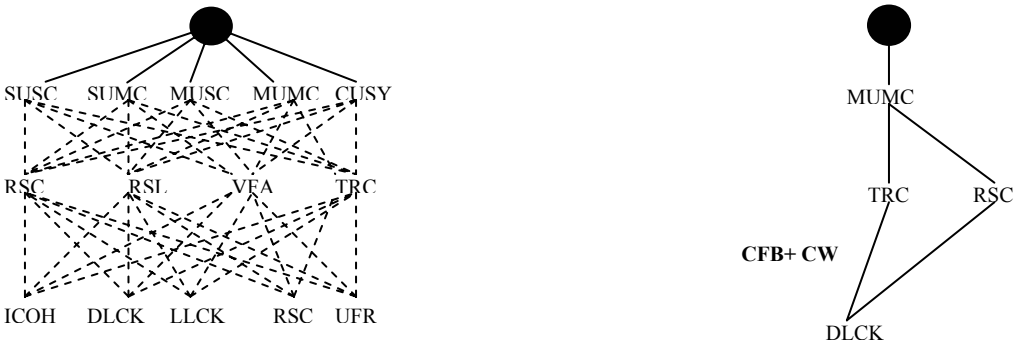

Figure 1: Feature Interaction Tree (FIT) and CFB+CW example

At this moment, the "FIT" is merely a graphical notation of the extended taxonomy to visualize the catalogue of FI's. We have not explored other interesting properties that the "FIT" may offer. For example, if deadlocking interaction could happen in CFB+CW pair and Call Transfer (CT) + CW pair separately, the question would be: could transitive property be applicable to "FIT"? That is, could deadlocking interaction also happen in CFB+CT pair? These questions are beyond the scope of this paper.

1.3.1 Livelocking (LLCK) and deadlocking (DLCK) interactions

Livelock is a well-defined computer science concept that occurs when the affected processes enter state transition loops and make no progress. In the case of Automatic Callback (AC) and Auto Recall (AR), if both features receive the busy signal from the callee and initiate the call simultaneously, both calls would not go through on single-line phones because both ends are busy making calls. Both features are in a livelock situation because they may always get the busy tone when they repeat the process and never complete the call. A livelocking interaction needs not be permanent; in most cases, the relative timing of the two processes leads to an eventual exit from the livelock loop.
Deadlock is another well-known distributed system concept that occurs when two or more processes are in a blocked state because they require exclusive access to a shared resource that belongs to the other, or wait for a message from the other process that will never be sent. An example is described in Figure 2 involving CW and CFB at A communicating with CW and CFB at B.

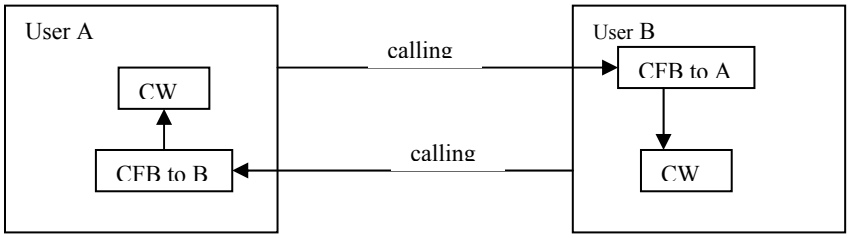

Figure 2: Call waiting and CFB at A versus Call waiting and CFB at B (deadlock)

If A and B call each other simultaneously and are programmed to forward to each other on busy, then both callers would keep hearing the phone ringing at the other end (and perhaps also hear the CW tone). They would be deadlocked at the ringing state and no progress would be made until one of them hangs up. If both ends do not subscribe to CW service, the call would be forwarded to each other on busy (or a call loop). Both users would be presented with a busy tone instead.

1.3.2 Incoherent interactions (ICOH)

An incoherent interaction is a form of a violation of feature assumptions (or properties). This term was first introduced in [9] to describe "the identification of specific incoherence properties" between the affected features. Furthermore, an incoherent interaction can be caused by resource contention and resource limitation. The two properties from the classical case of CFB and OCS is a good example; user A would successfully call user C via CFB even though an OCS entry at A is supposed to forbid any outgoing call to C. The CFB and OCS are programmed with contradictory assumptions.

1.3.3 Unfair interactions (UFR) & Unexpected Non-determinism (NDET)

Fairness is also a well-known distributed system concept in which processes of equal priority should be assigned fair scheduling so that they eventually proceed and have fair access to shared resources. A novel case of unfair feature interaction occurs between Pickup (CP) and Auto Answer (AA) (also known as Call Forward to Voicemail). If one of the CP destinations has subscribed to AA that would unconditionally forward any incoming calls to the voicemail, the call would always be answered by the destination with AA first.

Unexpected non-determinism is introduced here because it is an observable feature interaction. It occurs when a feature with non-deterministic behavior triggers other features that have normally deterministic behaviors, but due to this triggering behave in a non-deterministic fashion. The users are usually confused by the behavior of the affected features because they do not expect such non-deterministic behaviors. This is usually caused by timing and race conditions among the features. A case of this type of feature interaction is between Automatic Call Distribution (ACD) and Call Pickup (CP) (see Figure 3). The ACD feature allows incoming calls to the subscriber be redirected to one of the pre-programmed destinations (e.g. destination A or B). The redirecting policy can be a random selection or a deterministic algorithm. The CP feature allows the subscriber of the service to inform a list of destinations (destination A and B) that an incoming call has been put on hold, and is ready to be picked up by one of the destinations. It is conceivable that the switching element Y sends a call pickup message to all destinations and then another switching element called X with ACD would redirect the call to a particular destination (e.g. B). When destination A decides to pick up the call, the incoming call is no longer available because the call has already been redirected to destination B. In summary, the non-deterministic call redirecting policy of ACD affects the first-come-first-served call pickup policy of CP.

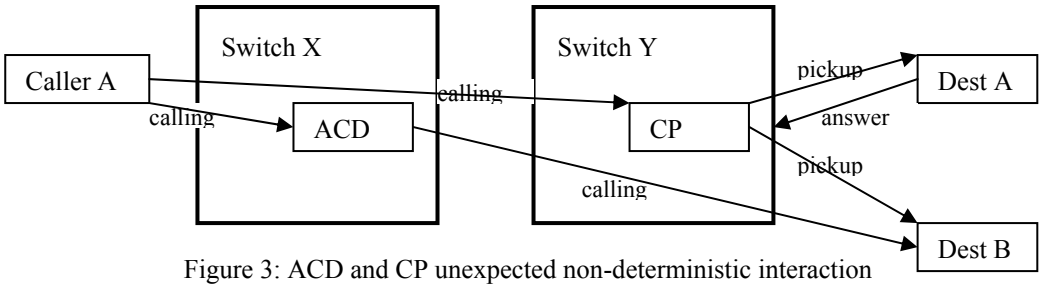

Figure 3: ACD and CP unexpected non-deterministic interaction

1.4 Overview of SIP

SIP (Session Initiation Protocol) is an application-layer multimedia signaling protocol standardized under IETF RFC 2543 [3]. Similar to most World Wide Web protocols, SIP has an ASCII-based syntax that closely resembles HTTP. This protocol can establish, modify, and terminate multimedia sessions that include multimedia conferences, Internet telephony calls, and similar applications. There are additional IETF drafts that describe other important extensions to SIP in efforts to realize VoIP deployment. However, these are beyond the scope of this paper.

In SIP terminology, a call consists of all participants in a conference invited by a common source. A SIP call is identified by a globally unique call-id. Thus, for example, if several people invite a user to the same multicast session, each of these invitations will be a unique call. However, after the multipoint call has been established, the logical connection between two participants is a call leg, which is identified by the combination of “Call-ID”, “To”, and “From” header fields. The sender of a request message or the receiver of a

response message is known as the client, whereas the receiver of a request message or the sender of a response message is known as the server. A user agent (UA) is a logical entity which contains both a user agent client and user agent server, and acts on behalf of an end user for the duration of the call. A proxy is an intermediary system that acts as both a server and a client for the purpose of making requests on behalf of other clients. A proxy may process or interpret requests internally or by passing them on, possibly after translation, to other servers.

What makes SIP interesting and different from other VoIP protocols are the message headers and body. Like HTTP, a SIP message, whether it is a request, response, or acknowledgement, consists of a header and a body. The Request-URI names the current destination of the request. It generally has the same value as the "To" header field, but may be different if the caller has a more direct path to the callee through the "Contact" field. The "From" and "To" header fields indicate the respective registration address of the caller and of the callee. The "Via" header fields are optional and indicate the path that the request has traveled so far. Only a proxy may append or remove its address as a "Via" header value to a request message. The "Record-Route" request and response header fields are optional fields and are added to a request by any proxy that insists on being in the path of subsequent requests for the same call leg. It contains a globally reachable Request-URI that identifies the proxy server. "Call-Id" represents a globally unique identifier for the current call session. The Command Sequence ("CSeq") consists of a unique transaction id and the associated request method. It allows user agents and proxies to trace the request and the corresponding response messages associated to the transaction. The body of a SIP request is usually a SDP [4], which contains the detail descriptions of the session.

The first line of a response message is the status line that includes the response string, code, and the version number. It is important to note that the "Via" header fields are removed from the response message by the corresponding proxies on the return path. When the calling user agent client receives this success response, it would send an "ACK" message back to the callee.

The message header fields that we have included in our SDL model are: "Request-URI", "Method", "Response Code", "From", "To", "Contact", "Via", "Record-Reroute", "Call-Id", and "CSeq". We believe these fields are most important to FI's because they convey the state of all the participants in the session. There are many header fields available in SIP and can become very complex. The associated RFC [3] is recommended for further details on SIP. We will discuss how the features are modeled in the next section.

2. Modeling SIP Services and Traditional Feature Interactions in SDL and MSC

In this section we describe how we modeled SIP services and their FI's. The Specification and Design Language (SDL) [6] was chosen as the modeling language for the following reasons: (1) the SIP protocol and services are state-oriented and map well to the communicating extended finite state machine model of SDL, (2) SDL is well supported by commercial software tool vendors like Telelogic [8] whose tools are used by many telecommunication software developers, and (3) various verification and validation techniques are available in SDL tools. Our approach to model SIP and its services starts with defining the use case and test scenarios using Message Sequence Charts (MSC) [7]. Next, we will describe the structural and behavior definitions of the SDL model. Then we will discuss the verification and validation process. We have described the basic SIP protocol and many additional services like CFB, CW, and OCS in SDL, but only selected diagrams will be presented in this paper.

2.1 Use Case Scenarios and Test scenarios

Since the IETF drafts have provided a call flow diagram or success scenario for each sample service in graphical notation [5], we translate these scenarios into message sequence charts. However, we note that these call flow diagrams only include the sequence of exchanged SIP messages at the protocol level. They do not represent service scenarios in the sense of use cases. Following standard practice of software engineering, we think that it is important to define service usage scenarios at the interface between the user and the system providing the communication service. We have therefore defined an abstract user interface which represents the interactions at the service level. These interactions between the users and the SIP system describe use case scenarios of SIP services. The users are the actors of the use case scenarios and are represented by the environment "env_0" in SDL.

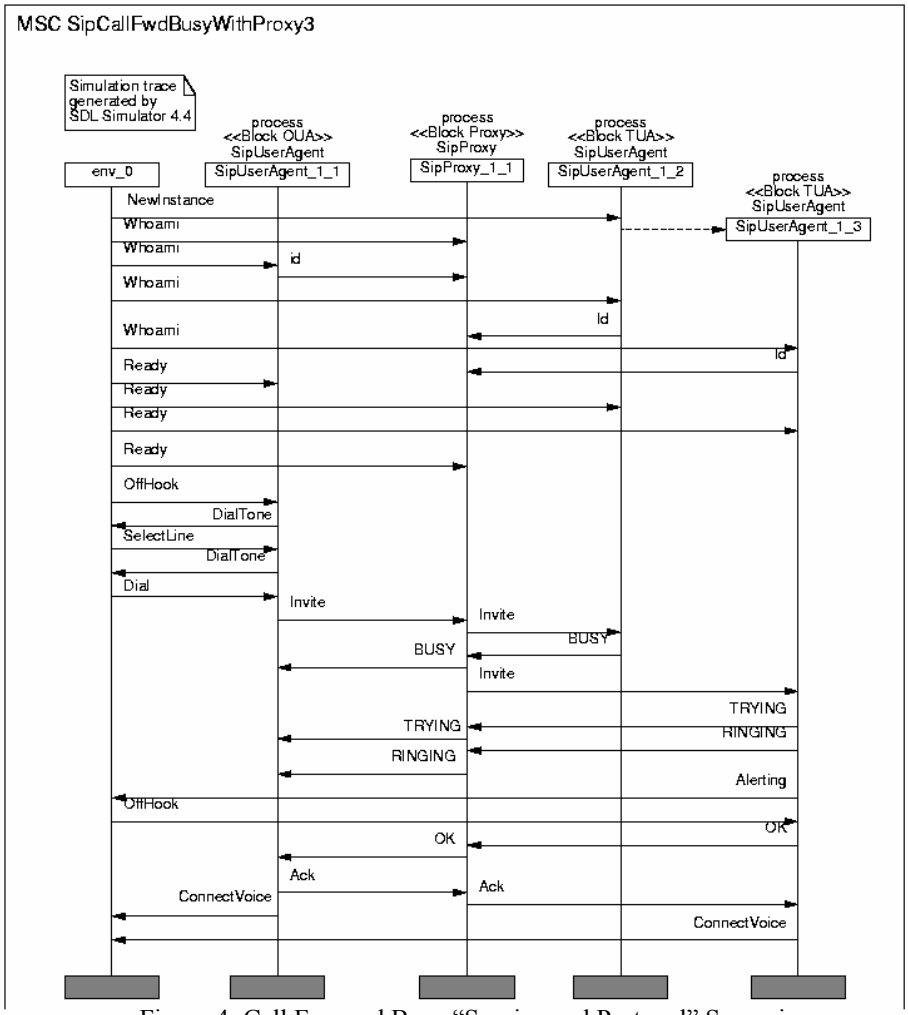

Figure 4: Call Forward Busy "Service and Protocol" Scenario

The sequence chart shown in Figure 4 represents the Call Forward Busy "service and protocol scenario". The sequence chart is the combination of the use case scenario with the corresponding scenario of exchanged SIP messages from [5]. It is written with response codes as message names and without message parameters so that the chart can fit in this paper. The complete MSC can be used as a test case for validating the SDL specification of the SIP protocol, as explained in the next subsection.

As a matter of facts, we have written one or more message sequence chart (service and protocol scenario) as the test case for each service because we use the Telelogic Tau's

Validator to verify our SDL model against the combined scenario.

2.2 Structural Definition

A system specification in SDL is divided into two parts: the system and its environment. An SDL specification is a formal model that defines the relevant properties of an existing system in the real world. Everything outside the system belongs to the environment. The SDL specification defines how the system reacts to events in the environment that are communicated by messages, called signals, sent to the system. The behavior of a system is the joint behavior of all the process instances contained in the system, which communicate with each other and the environment via signals. The process instances exist in parallel with equal rights. A process instance may create other processes, but once created, these new process instances have equal rights. SDL process instances are extended finite-state machines (FSMs). An extended finite-state machine (EFSM) is based on an FSM with the addition of variables which represent additional state information and signal parameters. The union of the FSM-state and additional state variables represent the complete state space of the process [12].

The relationship between modeled entities, their interfaces, and attributes are considered parts of the structural definition. A SDL system represents static interactions between SIP entities. The channels connected between various block instances specify the signals or SIP messages that are sent between user agents and/or proxies. Block and process types are used to represent SIP entity types such as user agent (SipUserAgentType) and proxy (SipProxyType).

A SIP User Agent contains both, what is called in SIP, a user agent client (UAC) and a user agent server (UAS). Since a user agent can only behave as either a UAC or UAS in a SIP transaction, the user agent is best represented by the inheritance of UAC and UAS interfaces. The inheritance relationship is modeled using separate gates (C2Sgate and S2Cgate) to partition the user agent process and block into two sections: client and server. The "Envgate" gate manages the sending and receiving of "Abstract User" signals between the user agent and the environment (see Figure 5). An instantiation of a block type represents an instance of a SIP entity such as user agent or proxy, and contains a process instance that describes the actual behavior of the entity. The process definition file contains the description of all the state transitions of the features to which the SIP entity has subscribed. In addition, each SIP entity must have a set of permanent and temporary variables for its operations. In the case of a user agent, the permanent variables store the current call processing state values of the call session. The temporary variables store the values of the consumed messages for further processing.

Similar to a user agent, a proxy consists of client and server portion. It tunnels messages between user agents but also intercepts incoming messages, injects new messages, or modifies forwarding messages on behalf of the user agent(s). A proxy is also a favorite entity to which features are deployed. A SIP Proxy has one gate that interacts with the environment (Envgate), and four gates that interact with user agents or proxies: client-to-proxy (C2Pgate), proxy-to-client (P2Cgate), server-to-proxy (S2Pgate), and proxy-to-server (P2Sgate) (see Figure 5).

In our SDL model, we use different SDL systems to represent different structural bindings between SIP entities and to simulate a particular set of call scenarios. The most complex system in our telephony model (see Figure 5) realizes the concept of originating and terminating user endpoints. It contains an originating user agent block, a proxy block, and a terminating user agent block. The originating block contains all the user agent process instances that originate SIP requests while the terminating block contains all the

user agent process instances that receive these requests. Upon receiving a request, a terminating user agent would reply with the corresponding response messages. It is important to note that only the originating user agent and proxy instances can send SIP requests (including acknowledgements).

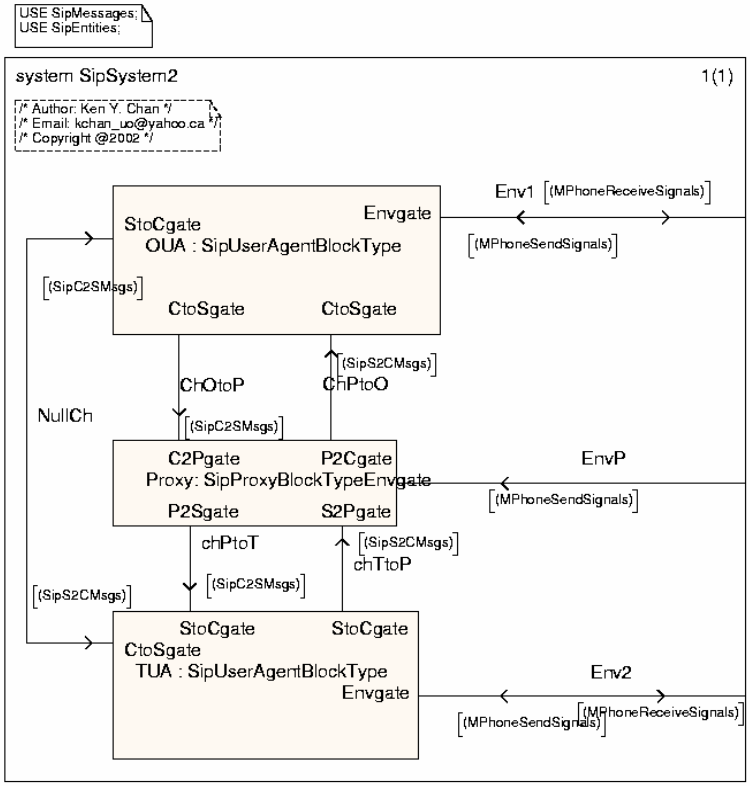

Figure 5: System Diagram of Proxy and User Agents

All blocks are initialized with one process instance. During the simulation, a 'NewInstance' "Abstract User" signal can be sent to a process instance to create a new process instance. Before the first "INVITE" message is sent, the environment must initialize each user agent and proxy instance with a unique Internet address by sending them a message called 'Whoami'. The user agent instances would in turn send an 'Id' message along with its Internet address and process id (PId) to the proxy. Thus, the proxy can establish a routing table for routing signals to the appropriate destinations during simulation. Similar to the user agent, a proxy has a set of permanent and temporary variables for its operations.

SIP messages are defined as SDL signals in the SIPMessage package. The key header fields in a SIP message are represented by the corresponding signal parameters. Since there is no linked list data structure in SDL, the number of variable fields such as 'Via' and 'Contact' must be fixed in our model. Complex data and array types of SDL have been tried for this purpose, but were dropped from the model because they may cause the Tau validation engine to crash. Instead, we used extra parameters to simulate these variable fields. A set of user signals, that is not part of the SIP specifications [3, 5], has also been defined here to facilitate simulation and verification. We call these user signals the "Abstract User interface". They represent the interface between the user and the local IP-telephony equipment in an abstract manner. The modeling of these user-observable behaviors is essential to describe FI's; however, the SIP messages described in the SIP standard do not describe these user interactions, concentrating only on the SIP messages exchanged between the different system components. For example, an INVITE message in SIP serves as more than just a call setup message; it may be used for putting the call on

hold in the middle of a call (mid-call features) [5]. The user interactions include such signals as Offhook, Onhook, Dial, SelectLine, Flash (flash hook), CancelKey, RingTone, AlertTone, TransferKey which relate to the actions that are available on most telephone units on the market.

2.3 Behavior Specification

In general, a SIP feature or service is represented by a set of interactions between users and the user agent processes, and possibly the proxy processes. Each process instance plays a role in a feature instance. A process instance contains state transitions which represent the behaviors that the process instance plays in a feature instance. In our model, we capture the behavior of a SIP entity in an SDL process type (e.g. UserAgentType). The first feature we model is the basic SIP signalling functionality, also known as the basic service. Each process type has state transitions that describe the basic service. In the case of a user agent, the process includes the UAC and UAS behavior. We define a feature role as the behavior of a SIP entity that makes up a feature in a distributed system. A feature role is invoked or triggered by trigger events. The entry states of a feature role in a SIP entity are the states for which the trigger events are defined as inputs. For example, the on-hold feature can be triggered in the 'Connect_Completed' (entry) state by an INVITE message with 'ONHOLD' as one of its parameters. After the invite message is sent, a response timer is immediately set. Then, the user agent is waiting for the on-hold request to be accepted. When the other user agent responds with an OK message, the requesting agent would reset the response timer and inform the device to display the on-hold signal on the screen.
In general, trigger events are expressed as incoming signals; pre-condition, post-conditions, constraints are expressed as enabling conditions or decision. Actions are tasks, procedure calls, or output signals. As a triggering event being consumed by the user agent process, the parameters of the event may be examined along with the pre-conditions of the feature. Then, actions such as sending out a message and modifying the internal variables may be executed. Post-conditions and constraints on the action may also be checked. Finally, the process progresses to the next state.

A state transition occurs when 1) an "Abstract User" signal is received from the environment, 2) a request or response message is received, or 3) a continuous signal is enabled. The so-called Continuous Signal in SDL is used to model the situation in which a transition is initiated when a certain condition is fulfilled. For example if the UAS is busy, the boolean 'isBusy' would be true in the 'Server_Ring' state. The UAS would immediately send the BUSY response to the caller. This way, we would not have to worry about the timer expiration because we do not need to send a busy toggling signal to simulate busy during a simulation. An asterisk '*' can be used in a state and signal symbol to denote any state or any signal; its semantics is equivalent to a wildcard. A state symbol with a dash '-' means the current state. Error handling, such as response timer expiration, can easily be modeled with SDL timers and a combination of '*' and '-' state symbols. For example, when the response timer expires, a timeout message would be automatically sent to the input queue of the user agent process. The expiration of the response message is generalized as the situation in which the receiving end does not answer the request in time. Thus, a 'NoAnswer' signal is sent to the environment. Finally, the process can either return to the previous state or go directly to the idle state.

Moreover, we can add additional features or services such as CFB, OCS, and other services, to the system. To add behaviors of additional features to a process type, we can subtype a "basic" process type such as UserAgentType. The derived type has the same interfaces and also additional state transitions [12]. We do not want to add new interfaces

(signal routes) to the process types because we do not want to change the interfaces of the block types. If a feature requires new SIP methods and response codes, we would not need to change the interfaces because method names and response codes are simply signals parameters in our model. Thus, we avoid the need to add new interfaces whenever a new feature is defined.

2.4 Verifying MSC against SDL Model

The SDL specification that has been discussed in the previous subsection was constructed using the Telelogic TAU tool version 4.3 and 4.4 [8]. TAU offers many verification or reachability analysis features: bit-state, exhaustive, random walk bit state exploration and verification of MSC. Bit-state exploration is particularly useful and efficient [10] in checking for deadlocking interactions because it checks for various reactive system properties without constructing the whole state space like the exhaustive approach does.

We verified our SDL model of SIP mainly by checking whether the model would be able to realize specific interaction scenarios which were described in the form of Message Sequence Charts (MSCs). In fact, we used the scenarios described informally in [3, 5], and rewrote them in the form of MSCs. Then we used the TAU tool to check that our SDL model was able to generate the given MSC. An MSC is verified if there exists an execution path in the SDL model such that the scenario described by the MSC can be satisfied. Thus, An MSC in Tau is considered as an existential quantification of the scenario. When "Verify MSC" is selected, Tau may report three types of results: verification of MSC, violation of MSC, and deadlock. Unless Tau is set to perform 100% state space coverage, partial state space coverage is generally performed and reported in most cases. Verification of an MSC in TAU is apparently achieved by using the MSC to guide the simulation of the SDL model. As a result, the state space of the verification has become manageable [11].

3. Detecting Feature Interactions

One of our objectives in this research is to explore the feasibility of using Tau to detect known and new FI's. Although SIP is fundamentally different from traditional PSTN signaling protocols, most of the traditional POTS FI's still exist in SIP, if the SIP services are designed and implemented with the mindset of POTS. (Note: Alternate design approaches to SIP services are discussed in the next section). We do not believe it is either feasible or practical to come up with an automated feature interaction detection scheme in Tau environment because Tau has the limited validation features in Tau and Tau's Validator frequently crashes. Instead, we write test cases in the forms of either MSC or Tau's Observer Process Assertions to verify whether traditional POTS FI's that were discovered by other feature interaction's researchers still exist or not [2,9].

In the previous sections, many FI's were identified as violations of distributed system properties: deadlock (a form of violation of safety) and livelock (a form of violation of liveness). The Tau tool offers various automated reachability analysis of SDL models and reports any deadlocks found during the analysis. Let us revisit the previous deadlock example on CFB and CW (see Section 1.3). If we have a SDL system with the corresponding SDL blocks or SIP entities which has a user agent process running with CFB and CW behaviors, we can select the "bit state exploration" option to ensure no deadlock would be found. Tau's Validator does not seem to offer reports on the liveness of the analyzed system in any of its reachability analysis functions. Thus, we use Tau's Observer Process features to detect live-locking FI's, as explained in Section 3.2.

Furthermore, Incoherent interactions are not easy to check with the Tau tool because certain interactions that involve contradictory properties cannot be expressed in a straightforward manner. There are two ways to approach this problem: (1) Use an MSC to specify incoherent properties, or (2) use an observer process offered by Tau to specify assertions. By examining the validation report (which shows the simulation trace in the from of MSC), we could trace back to the locations where the problem occurs; thus we could identify the interacting features. Both approaches have their advantages and disadvantages. However, the observer approach is the only practical approach in the current Tau release.

3.1 Specify Incoherencies as MSC

Many service designers like to use MSCs to capture important scenarios of a feature. If the sets of message sequence charts describing two features can be compared/verified against each other, then certain obvious incoherent FI's may be detected in this informal requirement specification stage. However, capturing key behaviors of a feature in MSC is not always possible. For example, the success scenario of OCS cannot be expressed directly as an MSC in the case of OCS and CFB interaction. How can we express in an MSC that user A cannot call user C? The concept of something that can "never happen" is usually expressed using universal quantification. Since verification of MSCs in Tau is based on an existential quantification of the scenario, a property that something should "never" happen can be expressed by negation. If m is a scenario that should never happen, then we could use the Tau tool to check whether the SDL model can satisfy this MSC m. If the result is that m cannot be satisfied by the model, this verifies the property.

If this concept is applied to OCS, "verifying user A can call user C" being false is equivalent to the truth of "user A can never call user C". Thus, the successful verification of the success call scenario from user A to user C concludes the violation of the OCS specification (see OCS MSC diagram). Since features like OCS apply to all calls including mid-call, the pre-enable condition of the MSC must be specified. The pre-enable condition must include all possible situations in which a call can be made to user C. There are very many possibilities; clearly, detecting incoherent interactions using MSC with the current version of Tau is almost impossible.

However, an extension to MSCs, called Live Sequence Chart (LSC) [16], seems to address the above concerns. First of all, an LSC allows the designer to specify messages that cannot be sent in a scenario. This is useful for specifying OCS type of services. Secondly, an LSC can be specified with either the universal or existential quantification of a scenario, which offers great flexibility in verifying scenarios that are contradictory to each other. Last but not least, a universal LSC allows an associated pre-enable condition which defines the scope of the scenario. This is essential to modeling services because such pre-enable conditions can be used as the triggering condition of a feature. Unfortunately, Tau does not support LSC but we believe LSC is a promising tool in this context. Details on LSC are beyond the scope of this paper and will be considered in future work.

3.2 Specify Incoherencies as Assertions in an Observer Process

Observer Processes with powerful access to signals and variables of various processes are provided by Tau as a tool which is useful for feature interaction detection. One or more observer processes can be included in An SDL system to observe the internal state of other processes during validation [8]. When the validation engine is invoked to perform state exploration, the observer processes remain idle until all the observed processes have made

their transitions. Then, each observer process would make one transition. All observer processes have access to the internal states of all the observed processes via the Access operators. Therefore, the conditions (e.g. the continuous signals, decisions and enabling condition) in which the observer process makes a transition may be considered for defining assertions for the observed processes. If the observer process finds that the assertion is violated, it will generate a report in the middle of the validation. The validation process would be stopped immediately and the report would be presented to the user.

Furthermore, we have experimented with assertions that verify certain liveness properties of the system. For example, if we have an integer counter for each user agent to keep track of the number of times each user agent has been in 'Ringing' state, we can monitor whether a user agent has been looping at 'Ringing' state or not. More specifically, an assertion, which verifies user agent UA1 and UA2 are not simultaneously in 'Ringing State' for more than three times ensures a certain level of liveness in the system. We have yet to experiment with converting well-known liveness detection algorithms into SDL assertions. However, since we are not aware that Tau supports temporal logic, we believe specifying liveness property without temporal logic would be non-trivial.

In general, we use our intuition to come up with useful assertions for each feature interaction category. Although this is not an automated process, we believe it is a practical approach to a subjective problem such as detecting "undesirable side-effects" between features (FI's) in Tau's environment. We believe the current features offered in Tau are very limited for detecting new FI's, but Tau meets our key objective; we could verify whether the well known FI's still exist in SIP or not. Thus, we could apply preventive measures to make SIP services more robust, as explained in the next section.

4. Preventing Feature Interactions

After a feature interaction is detected, the next natural step is to change the design to prevent it. In this section, we discuss corrective measures that can be incorporated in the design and implementation of a system to prevent FI's. These corrective measures map directly to the causes and effects of FI's. Preventing FI's may be done at design-time or at run-time. Run-time prevention is more difficult to exercise because the features may be running on different nodes and it is not easy to coordinate the actions that should be performed when a feature interaction is detected. Prevention strategies for different feature interaction categories are presented in the following subsections. In addition, the feature interaction examples described in Section 1 will be revisited again. The goal of this discussion is to provide a catalog of FI's to service designers, such that a designer can associate a feature with a set of prevention schemes that would reduce potential interactions with any unknown features. We will also see that it is easier to prevent FI's in SIP than in POTS, because SIP has extra call information (e.g. 'Via', 'Record-Route', 'Contact') in the message headers.

4.1 Preventing Resource Contention and Limitation – (CW and TWC)

The natural prevention strategy for resource limitation is to increase the number of available resources. IP Telephony services, particularly SIP, should face fewer resource limitation problems than POTS because SIP end-user devices tend to be more powerful: fast processor, a lot of memory, and flexible user interfaces and displays. For example, the semantics ambiguity of flash hook at the POTS user interface, which is the cause of both resource contention and resource limitation in the case of CW and TWC in the POTS

service, should never happen with SIP phones. SIP phones should be able to display the choice of two actions, namely putting the current call on hold and switching to the incoming call, or conferencing in the third party. The phone may assign one or more soft buttons for this purpose. However, if interactive user intervention is not possible, priority schemes (e.g. fuzzy policies [13]) may be used to resolve resource contention. Clearly, these strategies can be applied in both design-time and run-time feature interaction prevention.

4.2 Preventing Incoherent interactions – (OCS and CFB)

Incoherent interactions can either be direct or transitive [9]. Direct incoherencies are present when two features are associated with the same trigger signal but lead to different or contradictory results. A transitive incoherent interaction occurs when one feature triggers another feature, and the latter has results that are contradictory to the former feature. The word 'transitive' refers to the transitivity with respect to features that may lead to loops. Gorse has suggested a scheme to detect this type of interaction [9]. However, no prevention scheme has been presented. Since incoherencies lead to contradictory results, allowing only one of the features to be triggered would prevent the incoherence. However, transitive incoherencies such as OCS and CFB are difficult to prevent at run-time because the interaction may involve indirect triggering of a remote feature. How could we know that the incoming call has triggered CFB running on server Y which contradicts the assumptions of the OCS feature running on server X? Obviously, if the triggering condition of the first feature were made available to the other feature and vice versa, the two features could take advantage of the extra information and negotiate a settlement. Figure 6 illustrates an application of this scheme.

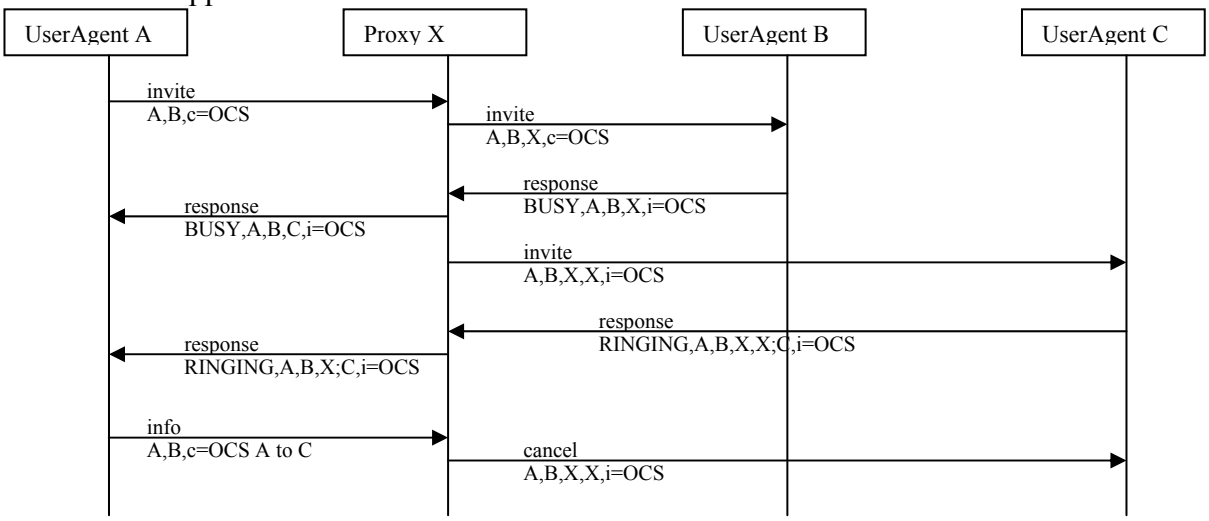

Figure 6: Preventing CFB and OCS incoherent interactions in SIP

In SIP, the 'Via' header indicates the path the request has traveled and can be used to prevent loops. The 'Record-Route' header indicates the proxy path that subsequent requests must follow. The formal specification of OCS and CFB can take advantage of header values in {From, To, Via, Contact, Record-Route} to determine whether a request has been forwarded to restricted destinations or not. For example, if OCS is activated at user agent A, the user agent may add an extra tag to some field (e.g. i=OCS) in the SDP of both the responses to A and all outgoing invitations, indicating that OCS is present at A (see Figure 6). When a downstream proxy or back-to-back user agent that is running CFB receives the invitation, it would add the 'Record-Route' header field with its proxy address and the final

destination as the tag to the response header. Upon the reception of the response from either the proxy or the destination, the OCS feature at the user agent A may deduce the call forward path and could determine whether a violation of its screening list occurs or not. If the answer is yes, then the OCS may send an "INFO" message [3] with some negotiation parameters to determine the appropriate action to be taken between CFB and OCS. If OCS was to have a higher priority than the CFB feature, the OCS user agent could send a cancel request instead of the final acknowledgement message, in order to cancel the original invitation. We believe this negotiation approach can be generalized but it is beyond the scope of this paper.

4.3 Preventing Unexpected Non-determinism & Unfair Interactions – (ACD & CP) and (CP & AA)

Unexpected non-deterministic interactions must be caused by at least one feature that has some non-deterministic behavior. The non-deterministic action of one feature typically triggers the other feature. Therefore, if we were to remove the triggering dependency of the second feature on the first feature, the problem would be solved. In the case of ACD and CP, an incoming call may trigger the non-deterministic invitation of ACD and the pickup invitation of CP to the same or different destination(s). If concurrent triggering of ACD and CP was detected and only one feature was activated, the problem would be solved.

An unfair interaction represents the violation of a fairness property. There might be tools that can perform reachability analysis on the system and verify that all processes are treated fairly. Such properties can be specified in temporal logic. We are not aware that Tau has the ability to deals with such properties. An unfair interaction can be avoided by relative priorities between the two features or by introducing randomization in the selection process.

4.5 Preventing Deadlocking & Livelocking Interactions - (CW & CFB) and (ACB & AR)

Since a deadlock between two features is caused by a mutual dependency on each other's resources, removing the cyclic dependency would typically resolve this problem. The dependency in the case of CW and CFB is on the Busy signal. The busy signal is intercepted by CW, therefore CFB at both ends would continuously ring each other. Loop detection can prevent the problem. Loop detection and prevention is a mandatory feature of the core SIP protocol. Thus, CFB features can examine the {From, To, Contact, Via, Record-Route} headers to determine whether a loop would occur or not. If the answer is yes, the subsequent forward would not be allowed and a response message with a response code of LOOP_DETECTED would be returned.

Livelock interactions, like deadlock interactions, also imply cyclic dependency on each other's resources. However, the result is that the system would fail to proceed. To break the lock or the loop, a randomized timer can be introduced to the triggering of the affected features. For example, ACB and AR may be stuck in a livelock if ACB and AR would initiate the callback and the recall simultaneously on single-line phones. If the triggering of one of these features was delayed, one of the calls would go through. However, this livelocking interaction is unlikely to happen because SIP phones usually have more than one line. Instead, ACB and AR would result in an incoherent interaction because both users would be presented with two calls between the same endpoints. Since two different SIP user agents at both ends handle the calls, loop detection in a user agent would normally not detect this interaction at run-time. However, the user agents that are

running in the same terminal may be designed such that they would share their route information with all their local user agents. As a result, this incoherent interaction could be prevented.

5. New Feature Interactions in SIP

Lennox introduced two feature interaction categories for IP telephony services called cooperative and adversarial interactions [1]. They are incidentally similar to, but not the same as, the concepts called cooperative and interfering interactions defined by Gorse [9]. In the following subsections, the SIP FI's described by Lennox are associated to our proposed taxonomy. The corresponding preventive measures will also be presented. Furthermore, new FI's that are unique to SIP will also be described.

5.1 Cooperative Feature Interactions

Cooperative FI's are multi-component interactions (SUMC or MUMC) where all components share a common goal, but have a different and uncoordinated way of achieving it [1]. Specific examples would be livelock, deadlock, unfairness, and unexpected non-deterministic multi-component interactions. The fact that features contend for a single resource or for each other's resources in order to establish call connections, resulting in violations of safeness and liveness properties, is a form of cooperative interactions. Let us examine the following SIP feature interaction proposed by Lennox.

Request Forking (RF) and Auto-answer (voicemail) is a potential cooperative FI. Request forking is unique to SIP; it allows a SIP proxy server P to attempt to locate a user by forwarding a request to multiple destinations in parallel [1]. The first destination to accept the request will be connected, and the call attempts to the others will be cancelled. The Auto-Answer (AA) or Call Forward to Voice Mail is designed to accept all incoming calls while the user is unavailable. Both features are intended to ensure that the user does not miss the call. However, since AA would answer the call request almost immediately, RF would always forward the call request to the same AA destination. Therefore, the user who may be closer to the other destinations would never be able to answer the call, and hence the intention of RF is ignored. To resolve this violation of fairness, one may introduce a delay timer to AA (e.g. answer after 4 rings) such that all destinations would be given a fair chance to answer the call.

5.2 Adversarial Feature Interactions

Adversarial FI's, by contrast, are multi-component interactions (SUMC, MUMC) where all components disagree about something to be done with the call (e.g. violation of feature assumptions) [1]. They are a form of incoherent interactions and are difficult to detect. The following new interactions are specific to SIP and have not been mentioned in [1].

Timed ACD and Timed Terminating Call Screening (TCS) is a potential adversarial FI. With IP telephony, service designers can rapidly create innovative features; for example, time or calendar-based forwarding and screening features. In this example, TCS restricts incoming calls that are originating from certain callers at the destination. ACD may be programmed to distribute different kinds of incoming calls to different sets of destinations depending on the time of the day (e.g. calls from A or B to destinations S or T in the daytime, and calls from C or D to destinations X or Y in the nighttime). However,

destinations S and T are programmed to screen calls from A or B in daytime and X and Y to screen calls from C and D at nighttime. Although ACD is intended to select the best route for incoming calls, TCS at various destinations is programmed to screen calls from these destinations at given time periods. As a result, their policies contradict with each other and no calls between these callers and callees can ever be completed in the conflicting time period. When time is involved in incoherent interactions, the solution to such conflicting policies is not trivial.

Call Screening and Register is another form of multi-component interaction unique to SIP involves dynamic address changing in SIP. A user agent or proxy may add, delete, or modify the current contact address of a user stored in the registrar. The address can be changed at any time by sending a "Register" request message to the registrar. Since a user cannot possibly stay current with the latest address change of another user, the call screening features would be rendered useless. It is a form of incoherent interaction. This interaction involves a debate of privacy for the call screener and the caller. These security issues are beyond the scope of this paper.

Dynamic Addressing and User Mobility and Anonymity is another adversarial FI that centers on the controversial issue on the balance between the lack of address scarcity in the Internet [1] and the correct programming of features that depend on reliable addresses. As it becomes the norm that a user owns more than one intelligent wireless and/or wired personal communication device, user mobility is a serious issue in designing reliable services. Furthermore, anonymous email accounts and email spamming are big problems in Internet. Could anonymous accounts and telemarketing spam calls emerge in the SIP world when SIP becomes popular? If SIP addresses were just as easy to obtain as email addresses, the call screening and forwarding feature in SIP would need to be more sophisticated than in POTS.

6 Conclusions and future work

The main contribution of this research described here is to provide a formal model of SIP services using the SDL language. The SDL model covers some of the important characteristics of SIP, such as caller-id, command sequence and, to a certain extent, most of the mandatory addressing header fields. The benefits and disadvantages of using a modeling and verification tool like Telelogic Tau are discussed. Our experience with verifying and validating our SDL model is also described. The second most important contribution is the discussion of new SIP FI's and some details on how to prevent the traditional FI's, and potentially new FI's, in SIP.

Another significant contribution is the discussion on how to detect FI's with the Tau tool. Although Tau has a reliable editor and simulation engine, we have run into many difficulties with the Tau validation engine. It frequently crashes with General Application Errors, which are unrecoverable in the Windows environment. We are not certain of the root cause of this problem but we suspect that the validation engine may not be able to handle very big state spaces or certain complex data structures. Hopefully, these issues would be addressed in future versions of the tool. We have used MSCs to characterize user interaction scenarios and used these scenarios to verify the SDL model of SIP. However, MSCs have limitations in terms of expressing quantification of instances and their behaviors. Live Sequence Chart (LSC) [16], which has not been discussed much in this paper, appear to be a promising extension to MSCs in this context. We have also used the observer processes provided by Tau to detect FI's. This seems to be a practical semi-automated approach, particularly to detect incoherent interactions and to verify liveness properties.

Finally, an extension to the classical feature interaction classification is presented to show the importance of categorizing FI's and the corresponding prevention strategies. As part of our future works, we would like to investigate potentially useful properties of FIT. In addition, we would like to modify our model to support the new SIP standard [15]. We intended to start the model with the latest standard but unfortunately the new RFC came to our attention only in the later stage of our project. We have some ideas on how to extend our model to support the new standard but it is beyond the scope of this paper.

Acknowledgements

This project would not be possible without the support of Communications and Information Technology Ontario (CITO) and Natural Science and Engineering Research Council (NSERC), Canada. The tools and the support we have received from Telelogic are appreciated. Furthermore, we would like to thank our colleagues at SITE, in particular Dr. Luigi Logrippo from Université du Québec en Outaouais and Dr. Daniel Amyot from University of Ottawa, for their insights to the domain of FI's.

References

[1] J. Lennox, and H. Schulzrinne, "Feature Interaction in Internet Telephony", Sixth Feature Interaction Workshop, IOS Press, May. 2000.

[2] E.J.Cameron, N.D.Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Shure, and H. Velthuijsen, "A feature interaction benchmark for IN and beyond," Feature Interactions in Telecommunications Systems, IOS Press, pp. 1-23, 1994.

[3] M. Handley, H. Shulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session Initiation Protocol", Request For Comments (Proposed Standard) 2543, Internet Engineering Task Force, Mar. 1999.

[4] M. Handley, and v. Jacobson, "SDP: Session Description Protocol", Request For Comments (Proposed Standard) 2327, Internet Engineering Task Force, April 1998.

[5] A. Johnston, R. Sparks, C. Cunningham, S. Donovan, and K. Summers, "SIP Service Examples", Internet Draft, Internet Engineering Task Force, June 2001, Work in progress.

[6] International Telecommunication Union, "ITU-TS Recommendation Z.100: Specification and Description Language (SDL)", ITU-TS, Geneva, Switzerland, 1999.

[7] International Telecommunication Union, "ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)", ITU-TS, Geneva, Switzerland, 1996.

[8] Telelogic Inc., "Telelogic Tau SDL & TTCN Suite", version 4.3 and 4.4, http://www.telelogic.com, accessed on Dec 20, 2002.

[9] N. Gorse, "The Feature Interaction Problem: Automatic Filtering of Incoherences & Generation of Validation Test Suites at the Design Stage" (Master Thesis), University of Ottawa, Ottawa, Ontario, Canada, 2001.

[10] Holzmann, G.J. (1988) 'An improved protocol reachability analysis technique', Software Practice and Experience, Vol. 18, No. 2, pp. 137-161.

[11] O. Hargen, "MSC Methodology", http://www.informatics.sintef.no/projects/sisu/sluttrapp/publicen.htm, accessed on Dec. 23, 2002, SISU, DES 94, Oslo, Norway, 1994.

[12] J. Ellsberger, D. Hogrefe, and A. Sarma, "SDL - Formal Object-oriented language for Communication Systems", Prentice Hall Europe, ISBN 0-13-621384-7, 1997.

[13] M. Amer, A. Karmouch, T. Gray, and S. Mankovskii, "Feature-Interaction Resolution Using Fuzzy Policies", Feature Interactions in Telecommunications and Software Systems VI, pp. 94-111, IOS Press, 2000.

[14] ITU, "Packet based multimedia communication systems", Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[15] J. Rosenberg, H. Shulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol", Request For Comments (Standards Track) 3261, Internet Engineering Task Force, June 2002.

[16] W. Damm, and D. Harel, "LSCs: Breathing Life into Message Sequence Charts*", Formal Methods in System Design, Kluwer Academic Publishers, pp. 19,45-80, 2001.