Towards Automated Detection of Feature Interactions*

Kenneth H. Braithwaite Joanne M. Atlee

Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1

Abstract

The *feature interaction problem* occurs when the addition of a new feature to a system disrupts the existing services and features. This paper describes a tabular notation for specifying the functional behavior of features. It also describes how four classes of feature interactions can be detected when features are specified in this new notation. Our goal is to develop a tool that can automatically analyze feature specifications and detect interactions at the specification stage of development.

Introduction

How does one add features to a system without disrupting the services and features already provided? A more difficult but related problem is: how can one ensure that combinations of independently developed services and features behave as expected? These questions, and other variations of the *feature interaction problem*, have plagued the telecommunications industry for several years [7]. More generally, they are problems that affect the development and evolution of all service-oriented software.

We use the following definitions of *service* and *feature* presented in [3]

- A *service* provides stand-alone functionality. For example, Plain Old Telephone Service (POTS) is a service.
- A *feature* provides added functionality to an existing feature or service; a feature cannot operate stand-alone. For example, *Call Waiting* adds functionality to POTS.
- A *feature interaction* occurs when one feature affects the behavior of another.

Sometimes feature interactions are desired, and one feature is explicitly designed to interact with another. In such a case, one wants to determine that the features not only interact, but that the interaction conforms to the specified behavior of the individual features [14]. In most cases, one simply wants to ensure that (supposedly) non-interacting features cannot interact.

*This research has been supported in part by the Natural Sciences and Engineering Research Council of Canada grant FSP137101, with matching funds from Bell-Northern Research Ltd.

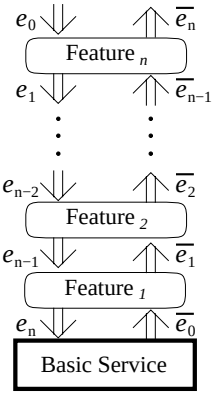


Figure 1: Information flow through layered state-transition machines.

We are investigating how to detect feature interactions during the requirements phase of development. Our goal is to develop a requirements notation that is flexible enough to support the specification of a wide variety of telephony events and properties but is rigorous enough to allow automated analysis. Automatic comparison of feature specifications is essential. For example, thousands of features have been implemented for telecommunication systems: the DMS-100[1] switch alone supports over 800 telephony features [10]. The number of possible combinations of telephony features precludes effective comparative analysis by humans. Automated analysis is also important when new features are created by third-party developers, since their designers are not likely to be intimately familiar with the base system and existing features.

At present, we have designed graphical and tabular notations for specifying the functional behavior of telephone services and features, and we have developed algorithms for detecting certain types of interactions among features. Using the taxonomy of feature interactions presented in [4], the classes of interactions we have been able to detect include interactions caused by call control manipulation, information manipulation, and resource contention; violations of feature invariants can also be detected.

In this paper, we describe the current status of our specification notation and the algorithms we have developed for detecting feature interactions. We demonstrate our method by showing how interactions can be detected among the specifications for features *Call Waiting*, *Three-Way Calling*, *Originating Call Screening*, *Call Forwarding*, *Calling Number Delivery*, and *Calling Number Delivery Blocking*. However, we anticipate that our method can be generalized and used to detect feature interactions in other service-oriented systems.

Modeling Services and Features

We are primarily interested in studying systems whose behavior can be modeled as layered state-transition machines. In such systems, the zeroth level machine specifies the system's basic services, and higher-level machines specify features that enhance the behavior of lower-level machines (see Figure 1). Information from the environment is

[1]DMS is a trademark of Northern Telecom.

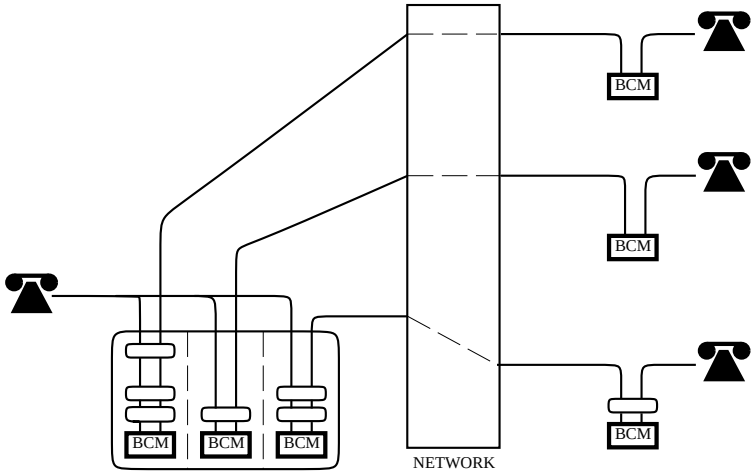


Figure 2: Representation of a call using single-ended call model.

input to the top-level machine and propagated down through the layers. At each level n, the n_{th} machine may either pass its input unaltered to the next machine; consume its input and perform some action; or consume its input, perform some action, and produce new information to be passed to the next machine. Examples of such systems include computer networks, operating systems, telephony systems [12], and robotics [2, 6]. The remainder of this paper will concentrate on telephony systems.

The basic service offered by a telecommunications system is to establish and maintain a communications link between two *agents*, which may be terminals or trunks. Traditionally, services and features have been defined in terms of a call model that encompasses both the originating and terminating agents. The advantage of this architecture has been improved performance, because all of the information a service or feature needs is locally available. The disadvantage is high coupling among the modules which implement services and features. As a result, service and feature modules are tightly interconnected, and it has become increasingly difficult and expensive to maintain and enhance the system.

To rectify the situation, and to emphasize changeability and reusability over performance, telephony systems are being re-designed using a single-ended call model that is based on agents rather than on connections (see Figure 2). The behavior of an agent's call is specified by its *call stack*: a set of layered state-transition machines. The zeroth-level machine of an agent's call stack specifies basic call processing, and the higher-level machines specify activated telephone features (e.g., *Three-Way Calling*, *Hold*, etc.). The behavior of a connection between two agents is the composition of their call stacks. If an agent is involved in more than one two-party call then it has multiple call stacks, one for each two-party call.

State-transition machines depicting the basic call processing service are shown in Figure 3. The state machine on the left specifies the *Originating Basic Call Model (OCM)*; it describes the behavior of a call that is initiated by the agent. The state machine on the right specifies the *Terminating Basic Call Model (TCM)*; it describes the behavior of a call that is being received by the agent. The figures are based on the basic call models defined in AIN Release 1.0.

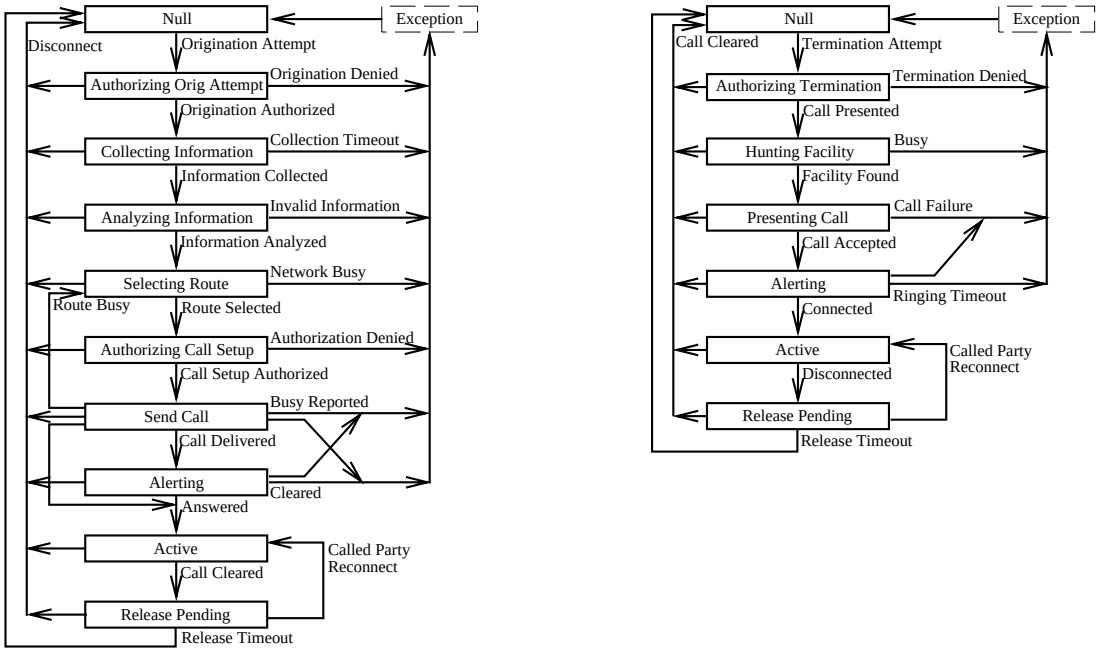


Figure 3: Basic Call Model as defined in AIN Release 1.0.

State-transition machines specifying activated features are layered on top of the agent's basic call model in order of their priority. The machine on the top of the stack has the highest priority because it has the first chance to act on incoming messages and has the last chance to modify outgoing messages.

In the model we propose, information that is passed up and down the call stack is represented by tokens. Some tokens carry no information other than their name, such as the **flashhook** token. Some tokens have a variable value, such as the **target** token, which represents the directory number of the called agent. Some tokens are compound, having other tokens as components: for example, the **Call Request** contains the set of tokens needed by a terminating agent to establish a connection with the originating agent; **Call Request** includes component token **target**.

Consider the telephony feature *Call Waiting*, which allows an agent (subsequently referred to as the *user* of the feature) to accept a call while on the phone with another agent. If a call comes in while the user is on the phone, *Call Waiting* generates a signal to notify the user that another call has arrived. At this point, the user has the option of putting the active call on hold and accepting the new incoming call. Figure 4 contains state-transition machine specifications for the *Call Waiting* feature. To improve the readability, modifiability, and reusability of feature specifications, we propose creating separate state machines for the different types of calls the feature can modify. Separate state machines are also created if the types of calls that the feature modifies can either be an originating call or a terminating call. Different tokens are passed up and down the call stacks of originating and terminating calls, so the tokens that a feature consumes, outputs, or uses to trigger state transitions depends on which underlying call model the feature is operating on.

In Figure 4, the top machine describes how *Call Waiting* affects the behavior of the incoming call. Since the new call must be a terminating call, only one state-transition machine is needed. However, the call that existed before *Call Waiting* was activated could

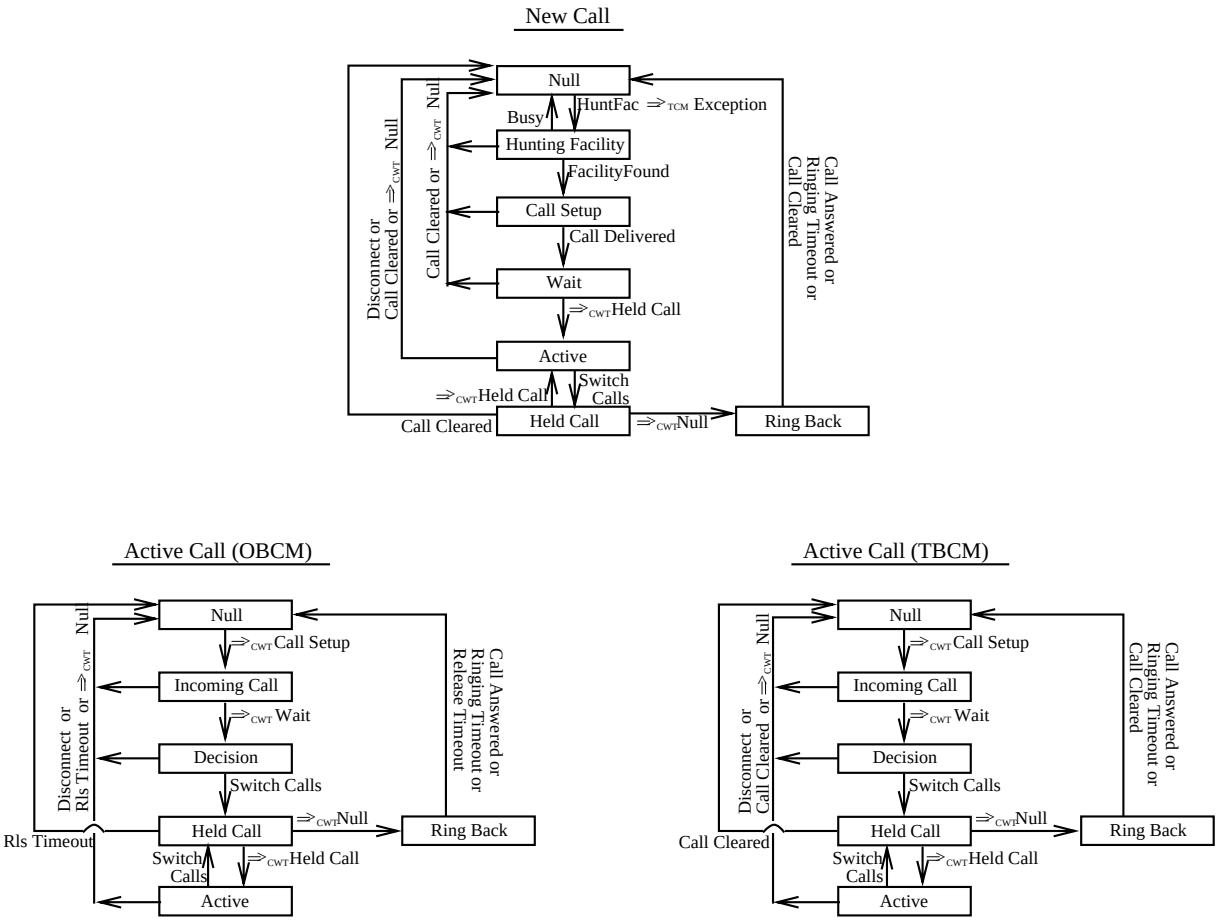


Figure 4: Call Waiting feature using split call model.

either have been initiated or received by the user. Thus, we create two state machines to describe how the feature affects the call that was already in progress when the new call comes in: the machine on the bottom left in Figure 4 describes how *Call Waiting* modifies an active call the user originated and the machine on the bottom right describes how *Call Waiting* modifies an active call the user received. The only difference between the two state-transition machines for *Call Waiting* on an active call is the token that the call model receives from the communication link indicating that the remote agent has hung up the phone: an originating call receives a **Call Cleared** token whereas a terminating call receives a **Release Timeout** token.

The graphical representations of state-transition machines shown in Figure 4 provide an intuitive understanding of how the *Call Waiting* feature affects normal call processing. A new call stack containing a terminating call model is created whenever a call arrives for an agent. *Call Waiting* is activated when the terminating call model determines that the agent is busy. At this point, the agent has two call stacks: one for the original call and one for the call that just arrived. When *Call Waiting* is activated, the feature machine for a new call is placed on the new call's call stack, and the appropriate feature machine for an active call is placed on the original call's call stack. The transition labels in the feature's graphical specifications are mnemonic names for the events that activate transitions in the state-transition machines. Labels prefixed with symbol $\Rightarrow_{CWT}$ indicate that the transition is activated by another transition in one of the feature's other machines

State	Input	Output	NewState	Assertion
NULL	$\Rightarrow_{CWT}$CALLSETUP		INCOMINGCALL	
INCOMINGCALL	$\Rightarrow_{CWT}$WAIT		DECISION	
	$\Rightarrow_{CWT}$NULL		NULL	
	$\Downarrow_A$**Disconnect**	$\ll$*forward msg*$\gg$	NULL	
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	
DECISION	$\Downarrow_A$**Switch Calls**	**Hold**$\Downarrow_A$	HELDCALL	
	$\Rightarrow_{CWT}$NULL		NULL	
	$\Downarrow_A$**Disconnect**	$\ll$*forward msg*$\gg$	NULL	
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	
HELDCALL	$\Rightarrow_{CWT}$HELDCALL	**Release Hold**$\Downarrow_A$	ACTIVE	
	$\Rightarrow_{CWT}$NULL	**Alert**$\Uparrow_A$	RINGBACK	
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	
ACTIVE	$\Downarrow_A$**Switch Calls**	**Hold**$\Downarrow_A$	HELDCALL	
	$\Rightarrow_{CWT}$NULL		NULL	
	$\Downarrow_A$**Disconnect**	$\ll$*forward msg*$\gg$	NULL	
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	
RINGBACK	$\Downarrow_A$**Call Answered**	**Release Hold**$\Downarrow_A$	NULL	
	$\triangleright$Ringing Timeout	**Disconnect**$\Downarrow_A$	NULL	
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	

Table 1: Call Waiting Specification for Active Originating Call.

running concurrently on a different call stack. For example in the *Call Waiting* machines for both the new call and the active call, the transitions into HELDCALL are activated by an event **Switch Calls** indicating the the user is switching from one call to another; the transitions into ACTIVE are activated by a transition into HELDCALL in the parallel machine.

Although graphical specifications provide an intuitive understanding of a feature's functional behavior, they are missing a lot of the detail needed to detect interactions. For example, the transitions have labels representing the event which causes the state transition, but there is no distinction between different types of events and there is no indication that the transitions have side-effects (e.g., that tokens are consumed or produced). We have designed a tabular notation for specifying a feature's functional behavior that is similar to the SCR tabular notation [8]. We intend feature specifications to be composed of both an intuitive graphical specification and a more precise tabular specification.

Tables 1 and 2 formally specify the behavior of the *Call Waiting* feature for an active originating call and an incoming call, respectively. A third table specifying the behavior of the feature with respect to an active terminating call is not shown; it would be the same as Table 1, with all occurrences of input $\Downarrow_N$**Call Cleared** replaced by input $\Uparrow_N$**Release Timeout**. Appendix A contains the tabular specifications of the originating and terminating call models.

Table 3 lists the type of input events, output events, and assertions that may appear in a tabular specification. We would like to emphasize that this is a preliminary list of events and assertions, and we expect that the list will grow as we gain more experience specifying

State	Input	Output	NewState	Assertion
Null	HuntingFacility$\Rightarrow_{TCM}$Exception		HuntingFacility	**uses bridge**
HuntingFacility	▷Facility Found	HuntingFacility$\Rightarrow_{TCM}$PresentingCall	CallSetup	
	▷Busy	HuntingFacility$\Rightarrow_{TCM}$Exception	Null	**!uses bridge**
	$\Rightarrow_{CWT}$Null	HuntingFacility$\Rightarrow_{TCM}$Null	Null	**!uses bridge**
	$\Downarrow_N$**Call Cleared**	HuntingFacility$\Rightarrow_{TCM}$Null	Null	**!uses bridge**
CallSetup	$\Uparrow_A$**Alert**	**Beep**$\Uparrow_A$	Wait	
	$\Rightarrow_{CWT}$Null		Null	**!uses bridge**
	$\Downarrow_N$**Call Cleared**	$\ll$*forward msg*$\gg$	Null	**!uses bridge**
Wait	$\Rightarrow_{CWT}$HeldCall	Call Answered$\Downarrow_A$	Active	
	$\Rightarrow_{CWT}$Null		Null	**!uses bridge**
	$\Downarrow_N$**Call Cleared**	$\ll$*forward msg*$\gg$	Null	**!uses bridge**
Active	$\Downarrow_A$**Switch Calls**	**Hold**$\Downarrow_A$	HeldCall	
	$\Rightarrow_{CWT}$Null		Null	**!uses bridge**
	$\Downarrow_A$**Disconnect**	$\ll$*forward msg*$\gg$	Null	**!uses bridge**
	$\Downarrow_N$**Call Cleared**	$\ll$*forward msg*$\gg$	Null	**!uses bridge**
HeldCall	$\Rightarrow_{CWT}$HeldCall	**Release Hold**$\Downarrow_A$	Active	
	$\Rightarrow_{CWT}$Null	**Alert**$\Uparrow_A$	RingBack	
	$\Downarrow_N$**Call Cleared**	$\ll$*forward msg*$\gg$	Null	**!uses bridge**
RingBack	$\Downarrow_A$**Call Answered**	**Release Hold**$\Downarrow_A$	Null	**!uses bridge**
	▷Ringing Timeout	**Disconnect**$\Downarrow_A$	Null	**!uses bridge**
	$\Downarrow_N$**Call Cleared**	$\ll$*forward msg*$\gg$	Null	**!uses bridge**

Table 2: Call Waiting Specification for Incoming (Terminating) Call.

Input events:

$\Downarrow_A$**token**	- information (interpreted as being from the agent) that is being passed down the call stack.
$\Downarrow_N$**token**	- information (interpreted as being from the call stack of connection's remote agent via the communications network) that is being passed down the call stack.
$\Uparrow_A$**token**	- information that is being passed up the call stack towards the agent.
$\Uparrow_N$**token**	- information that is being passed up the call stack towards the communications network (i.e., towards the call stack of the connection's remote agent).
$\Rightarrow_f$ S	- a signal from one of the feature's other machines, indicating that it has transitioned into state S. This signal is only visible to other state-transition machines associated with the same instantiation of the feature.
S1$\Rightarrow_{CM}$S2	- a signal that the underlying call model is transitioning from S1 to S2. This signal is passed down the call stack so that the activated features (in the order of their priority) have the opportunity to circumvent the transition and impose its own desired transition in the underlying call model. Imposed call model transitions are treated as original call model transitions and are passed down the call stack (again) so that the activated features have the opportunity to circumvent the new imposed transition.
S1$\Rightarrow_f$S2	- a signal that an underlying feature *f* is transitioning from S1 to S2. This signal is passed from the top of the call stack down to the feature making the transition so that higher-priority features have the opportunity to circumvent the transition and impose its own desired transition in the lower-level feature. As with modified call model transitions, modified feature transitions are passed down the call stack again, ending at the affected feature, so that higher-level features can change the modified transition.
$\triangleright$event	- a signal indicating the termination of internal processing. This signal is only visible to the machine performing the internal processing.

Output events:

token$\Downarrow_A$	- a **token** is sent from the feature to the call model, which lower-level machines will assume is from the agent.
token$\Downarrow_N$	- a **token** is sent from the feature to the call model, which lower-level machines will assume is from the communications network (i.e., from the call stack of the connection's remote agent).
token$\Uparrow_A$	- a **token** is sent from the feature to the agent, which will appear to be from the call model.
token$\Uparrow_N$	- a **token** is sent from the feature to the communications network (i.e., the call stack of the connection's remote agent), which will appear to be from the call model.
$\ll$*forward msg*$\gg$	- the input token is forwarded to the next level machine without alteration.
S1$\Rightarrow_{CM}$S2	- the feature imposes a new state transition from S1 to S2 in the call model.
S1$\Rightarrow_f$S2	- the feature imposes a new state transition from S1 to S2 in lower-level feature *f*.

Assertions:

connect(A,B)	- a connection is established between agents **A** and **B**.
uses X	- the feature acquires resource **X**.
Q(token)	- the feature asserts **Q** on the value of **token**.
$\models$**A**	- the feature raises assertion **A**, which continues to hold along all computation paths until it is lowered. **A** is a propositional logic formula, where the propositions are primitive assertions (e.g., **Q(token)**).
!A	- the feature lowers assertion **A**.

Table 3: Notation used in tabular specifications.

features. **Input events** include tokens from the environment to the call model[2], tokens from the call model to the environment, signals ($\Rightarrow_f$) between the feature's machines on different call stacks indicating the occurrence of state-transitions, notifications of state-transitions in the underlying call model or in lower-level features, and local internal signals ($\triangleright$) indicating the termination of internal processing. **Output events** include tokens that are output to either the call model or the environment and notifications of imposed state-transitions in the call model or in lower-level features. **Assertions** include established connections between agents, acquisitions of call-processing resources, dispossessions of resources, assertions on the values of tokens, and required relationships among more primitive assertions.

The tables specify the behavior of the feature in terms of functions and assertions. Each row in a table specifies a mapping from a state and an input event to a new state, a set of output events, and a set of raised assertions. Let T be the set of all tokens that the basic call models and features can generate and output to the call stack[3]; let inT be the set of input events denoting the receipt of a token $t \in T$ and let $outT$ be the set of events denoting the output of a token $t \in T$. Let N be the set of notifications of all state-transitions specified in the call models and features, and let inN and $outN$ denote the sets of events denoting the receipt or the output of a notification, respectively. For a given feature f, let S be the feature's set of states, let R be the feature's transition relation, let I be the feature's set of local internal signals, and let A be the set of assertions raised by the feature. Then formally, a tabular specification $\mathcal{T}$ represents a partial function from states and input events to states, sets of output events, and sets of assertions.

$$\mathcal{T} : (S \times (inT \cup inN \cup I \cup R)) \mapsto (S \times \mathcal{P}(outT \cup outN) \times \mathcal{P}(A))$$

Note that there are two methods by which a feature can affect the operation of the underlying call model. One method is to generate tokens on behalf of the agent or the communications network (i.e., on behalf of the remote agent associated with an established connection); these tokens are passed down the call stack towards the call model and are expected to cause the call model to change state. For example in the RINGBACK state of *Call Waiting* in Table 1, the feature has alerted the user that he left a call on hold when he hung up the phone; if the user does not answer the ringback before the $\triangleright$Ringing Timeout event, then the feature terminates the held call by sending a **Disconnect**$\Downarrow_A$token to the call model indicating that the user has hung up. The second method is to circumvent certain state-transitions in the call model or in lower-level features and replace them with new state-transitions. Notifications of all state-transitions are passed down the call stack so that features can act on them. Note that if a feature receives notification of transition S$\Rightarrow$T, it can only impose a new transition S$\Rightarrow$U if there exists a transition from S to U in the lower-level machine that was making transition S$\Rightarrow$T. For example in Table 2, *Call Waiting* is initially activated by the transition from HUNTINGFACILITY to EXCEPTION in the terminating call model (i.e., the feature is activated because the call model has determined that the user's line is busy). The *Call Waiting* feature consumes this state-transition

[2]To enhance readability, tokens from the environment that are being passed down the call stack are annotated with their source, which is either the agent associated with call stack or the communications network. Tokens from features or from the call model to the environment are annotated with their destination.

[3]Note that set T will grow as new tokens are needed to implement new features.

notification, performs it own HUNTINGFACILITY operation using the new information that *Call Waiting* has been activated, and outputs a new call model transition based on the results of its HUNTINGFACILITY computation.

Algorithm for Detecting Feature Interactions

Feature specifications are analyzed by composing feature machines and tracing the paths through the composite machine. There is a trace path for each possible call processing sequence. Feature interactions can be detected by testing the reachable states of the composite machine is analyzed. Some interactions are detected as a conflict between the information required in a state and the information actually present in that state, some are detected as a relationship between assertions in a particular state of the composite machine, and some are detected as a perturbation in the paths through a component machine when it is part of the composite machine. Equivalently, the reachability graph for the composition of features can be built and searched.

In this section we introduce four categories of interactions our analysis method can detect. We illustrate how the feature machines machines are composed and how the composite machines are traced. Finally, we show how these traces are used to detect the four types of interaction.

Classes of Feature Interactions

Our method is based on requirements and design specifications, so it can only be used to detect interactions that manifest themselves at the specification stage; implementation-dependent interactions and race conditions will not be detected. In addition, we have concentrated our efforts on detecting interactions among call processing features; we have not addressed interactions among features associated with operations, administration, and maintenance services, such as billing. The classes of interactions we have been able to detect include *call control interactions*, *resource contentions*, *information invalidations*, and *assertion invalidations.*

Some interactions arise because different features try to interpret the same signals simultaneously with conflicting results. In our model these *call control interactions* show up when the current states of different features on the same call stack each have a transition with the same token as an input event coming down the call stack. When this happens only one feature may receive the signal, thereby changing the expected behavior of the other.

Resource contentions arise when features try to share an unsharable resource R. If the feature G tries to acquire R after the feature F has already done so and before F has released R, then there is a resource contention. Currently we restrict our attention to resource contentions among features of a single agent. In our model a feature indicates that it needs to use a resource R by raising the assertion **uses R**. This adds **uses R** to the assertion list for the agent. When the feature is finished with the resource it raises **!uses R**. This removes **uses R** from the agent's assertion list.

If two features each need the resource R they will each raise **uses R** at some point. A resource contention is detected as a duplicate insertion of **uses R** into the agent's assertion list. The contending features need not be on the same call stack.

Recall that in our system information flowing up and down call stacks m is represented as tokens. An *information invalidation* arises when a feature alters information which a another feature subsequently uses for a decision. These interactions will be detected by examining the tokens in the states of the feature machines.

If the alteration is the insertion or removal of an information token, then the sequence of tokens the second feature receives may perturb the reachability graph of the second feature. These interactions are detected by comparing the reachability graph of the second feature viewed on its own with its reachability graph when viewed as a component of the composite machine. This type of analysis requires a graph inclusion algorithm. In this paper, we are only addressing analysis tests that require a linear trace through the composite machine's reachability graph; thus, we will not discuss this type of information invalidation, which we call *restriction invalidation*, any further.

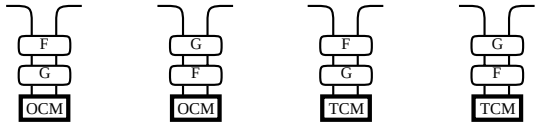
If the alteration of information consists of a change in the value but not the name of a token (such as changing the directory number of the caller or callee), then this is represented in the model by appending a prime (') to the token. Interactions caused by this kind of alteration are detected when the primed token (**token'**) arrives in a state of the second feature is expecting to receive **token**. In our model a state uses the information from a token if and only if it names that token in the input, output, or assertion portion of a transition leaving the state. Naming the token on a transition means that the feature will make a decision or take an action using the value extracted from that token as it passes through the state. If the token is arrives in the state primed then the decision or action is made on a changed value. Note that such interactions may be intended. Feature F may be explicitly designed to interact with feature G via changed tokens. We need to detect these interactions to help verify that they occur when they should.

In an information invalidation, a feature makes an incorrect assumption about a piece of information (such as that it has come from an OCM unchanged) that has been invalidated by a previous feature. The converse is also possible: a feature may make an assumption about a piece of information (such as that it will reach a the TCM unchanged) which is invalidated by a subsequent feature. We call this an *assertion invalidation.* A feature expresses its expectation about the future of a token by raising assertions about that token on transitions and raising requirements about which paths through the feature must satisfy the assertion. An assertion is a predicate. An assertion requirement is a logic formula that is tested using a truth table. An example requirement is $\models \mathbf{P} \rightarrow \mathbf{Q}$. The meaning of this is not that if **P** is true then **Q** can be inferred but that the the requirement is true on if whenever **P** has been raised then so has **Q**. The requirement for an assertion specifies those paths on which the assertion should hold. In our algorithm, assertion invalidations are manifested when an assertion required on all (specified) paths through a feature fails on a path in the composition. This failure is detected by testing the assertion and requirement in each state.

Composing Feature Machines

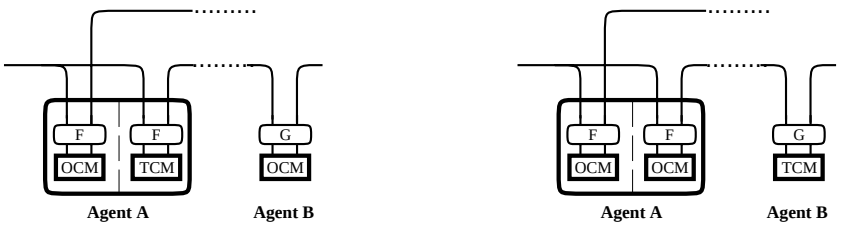
The composite machine for a set of features is built by executing all possible configurations of the component feature machines on all possible call models. Suppose we want to determine whether or not features F and G interact. We would need to compose the behaviors of the two features with respect to all possible call configurations that involve

both features. If the features can be invoked by the same agent, then all possible arrangements of the agent's call stack need to be traced. For example, if features F and G can be invoked on either an originating call model or a terminating call model, then we would need to search the following four call configurations:



If the features can be invoked by different agents (A and B) involved in the same telephone call, then there are two additional configurations that need to be traced:

Suppose further that F adds a new call to the configuration, thereby adding a new call stack to agent A's original call stack; then each of agent B's allowable call stacks must be composed with the allowable pairs of A's call stacks. Assuming that F creates a new originating call model on behalf of agent A, then there are two configurations in which the features are invoked by separate agents (the new call stack created by feature F is the leftmost call stack for agent A):



and there are six configurations in which the features are invoked by the same agent. All eight configurations need to be traced.

In general, the complexity of the composite machine will depend on whether the features operate on the same call stack, whether the features reside with the same user, wether the features can add new call stacks to the configuration, and whether the features can be invoked on Originating Call Models, Terminating Call Models, or both. Each stack-feature arrangement must be subjected to the tracing algorithm.

Tracing the Composite Machine

The reachability graph for the composition of two features can be built using a backtracking algorithm. Consider a trace on a call from agent A to agent B[4]. The trace starts

[4]This call may also involve originating or terminating calls from other agents depending on the features that A and B invoke.

with the OCM for A. If A's OCM proceeds to the point where a call request is sent up to agent B then a TCM for B commences. If at any time an event in a call stack triggers a feature, then that feature is placed on the stack in the appropriate state. If an event causes a transition in a feature then that transition occurs in the composite machine. Whenever a point of non-determinism is reached (where either one machine can make different transitions or the several machines can transition in different orders or on the same event) then backtracking is used to follow all the possible trace paths. The trace ends when both features have deactivated.

Some possible paths in the composite machine, or in each feature, can have arbitrary length due to cycles. This is especially true of features which respond to user commands. Consider an agent using *Three Way Calling* to alternately phone A and B while maintaining a call with C. If the feature's behavior does not depend on the number of times it traverses the cycle then it suffices to trace the cycle once. Hence for these features we need only trace acyclic or uni-cyclic paths and these are all of finite length. We have not yet discovered a feature whose behavior depends on the number of times a cycle is traversed.

The complexity of the tracing will depend on the complexity of each feature, on the amount of synchronization between the features, on whether they are in the call stacks of the same or different agents, and on how long the features' lifespans overlap. The number of paths will generally be small relative to the combinatorial possibilities because each feature will synchronize with its underlying call model and progress towards it deactivation.

Although the search space can explode exponentially in theory, we do not believe this happens in practice. The invocation requirements of a feature often reduces the number of call stack configurations that are possible. In addition most features can be represented by machines with few states, and most features have little branching and proceed almost linearly. Features represented by multiple machines executing in parallel usually have synchronized transitions. *Call Waiting* for instance is a complex feature, but the machine for each stack has only six states, and the graph is of low degree.

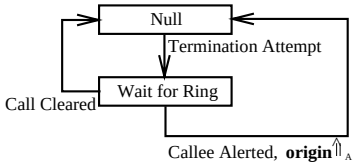
Detecting Feature Interactions

All the kinds of interactions discussed in this paper can be detected by testing states as opposed to paths in a composite machine's reachability graph. This means that with careful bookkeeping they can all be found by tests made while tracing the reachability graph.

Finding resource contentions involves checking the list of assertions for each agent. When a feature asserts **uses R** it is added to the assertion list of that agent. When **!uses R** is raised **uses R** is removed from the assertion list. If one feature tries to obtain resource R while another is using it then both features will raise **uses R**. A duplicate insertion of **uses R** in the assertion list of the same agent is a resource contention. There is an obvious generalization for sharing n copies of a resource.

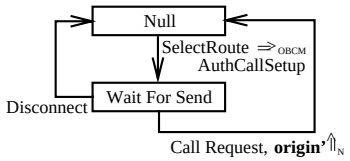
Call Control interactions are found by comparing the active states of the features on each call stack for a single agent. The comparison is made on the input events of the transitions out of the active states. If the same event occurs in the input portion of any two transitions then a call control interaction is detected. Both features are vying for the same event at the same time as it passes down the call stacks.

Information invalidations are found by looking at any primed token **t'** when it arrives



State	Input	Output	NewState
NULL	$\Downarrow_N$**Termination Attempt**	$\ll forward\ msg \gg$	WAITFORRING
WAITFORRING	$\Uparrow_A$**Alert**	$\ll forward\ msg \gg$ **Termination Attempt.origin**$\Uparrow_A$	NULL
	$\Downarrow_A$**Disconnect**	$\ll forward\ msg \gg$	NULL
	$\Downarrow_N$**Call Cleared**	$\ll forward\ msg \gg$	NULL

Figure 5: Specification of the *Calling Number Display* feature.



State	Input	Output	NewState
NULL	SELECTINGROUTE$\triangleright_{OCM}$AUTHCALLSETUP	$\ll forward\ msg \gg$	WAITFORSEND
WAITFORSEND	$\Uparrow_N$**Call Request**	**Call Request.origin'**$\Uparrow_N$	NULL
	$\Downarrow_A$**Disconnect**	$\ll forward\ msg \gg$	NULL
	$\Uparrow_N$**Release Timeout**	$\ll forward\ msg \gg$	NULL

Figure 6: Specification of the *Calling Number Display Blocking* feature.

in a state of a feature machine and seeing if that state has a transition which names **t** in its input, output, or assertion portion. Such transitions mean the feature wants the information in **t**. Since **t'** is present, that information is changed. This is an information invalidation.

Assertion invalidations are detected by testing assertion requirements in each state. All of the assertion lists for agents along a call must be examined for assertions and requirements. The set of assertions and assertion requirements in a state is the union of the assertion lists of all the agents in the call. An assertion requirement might be $\models$(**connect**(A,B) $\rightarrow$**X**(A). This can be tested by examining the assertion lists for **X**(A) and **connect(A,B)** to determine if **X**(A) is asserted whenever **connect(A,B)** is asserted.

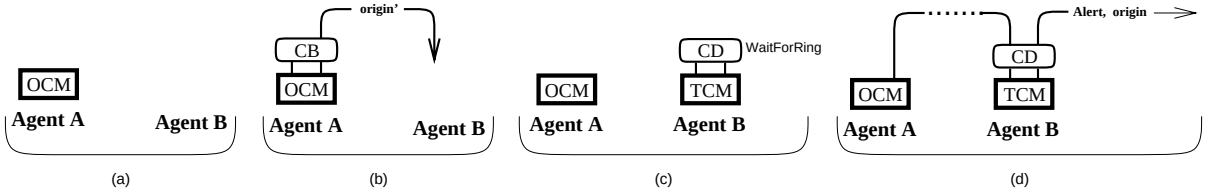


Figure 7: Example of an interaction due to information invalidation.

Case Studies

Example 1: Information Interaction

This example presents a fairly complete informal trace of the interaction of two features: *Calling Number Display* and *Calling Number Display Blocking*. *Calling Number Display* presents the directory number of the caller to the callee when the phone rings. *Calling Number Display Blocking* prevents the caller's directory number from being presented to the callee. The (intended) interaction between these features is that *Calling Number Display Blocking* will prevent *Calling Number Display* from properly displaying the caller's number.

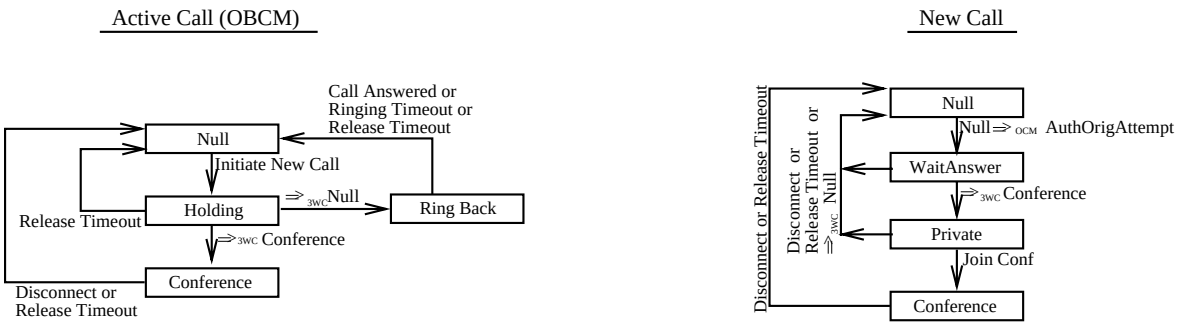
The feature machine for *Calling Number Display* has one active state. It records the originating caller's number when that token passes through the state, and presents the directory number to the user when the phone is rung. The feature machine for *Calling Number Display Blocking* also has only one active state. This state intercepts the token which indicates the caller's number and 'changes' it such that it can no longer be displayed.

Each feature operates on only one call model: *Calling Number Display Blocking* works on an OCM and *Calling Number Display* works on a TCM. Thus there is only one configuration of the call stacks to trace.

Figure 7(a) shows the first stage of the trace. The composite machine consists of a single call model: the caller's OCM. Next the OCM begins to set up a call. When it reaches state AUTHCALLSETUP, the feature *Calling Number Display Blocking* is activated and placed on the call stack. Upon activating, *Calling Number Display Blocking* immediately transitions into the WAIT state . If the call setup is authorized, the OCM produces the token $\Uparrow_N$**CallRequest** which is sent towards the destination agent B to initiate a call. **CallRequest** is a compound token with several fields; one of these fields is **origin**, which represents the caller's directory number (see Figure 7(b)). Token $\Uparrow_N$**Call Request** passes up the call stack to *Calling Number Display Blocking*. *Calling Number Display Blocking* forwards the message up the call stack with the **origin** marked as modified[5]. The modification is shown by appending a prime (') to **origin**. When the (modified) token is passed upwards, *Calling Number Display Blocking* transitions to NULL.

The **Call Request** token arrives at the destination agent as a **Termination Attempt** token. The arrival of this token activates both a terminating call model for the agent and the *Calling Number Display* feature (see Figure 7(c)). As it activates *Calling Number Display* transitions into the state WAITFORRING. One of the transitions from this state outputs the token **origin** (which is obtained from the compound token **Termination**

[5]In the real world implementation of this feature, it is actually a permission on the origin information that is modified.



State	Input	Output	NewState	Assertion
NULL	$\Downarrow_A$**Initiate New Call**	**New(OCM)** **Hold**$\Downarrow_A$	HOLDING	**uses bridge**
HOLDING	$\Rightarrow_{3WC}$CONFERENCE	**Release Hold**$\Downarrow_A$	CONFERENCE	
	$\Rightarrow_{3WC}$NULL	**Alert**$\Uparrow_A$	RINGBACK	
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	**!uses bridge**
CONFERENCE	$\Downarrow_A$**Disconnect**	$\ll$*forward msg*$\gg$	NULL	**!uses bridge**
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	**!uses bridge**
RINGBACK	$\Downarrow_A$**Answered**	**Release Hold**$\Downarrow_A$	NULL	**!uses bridge**
	$\triangleright$Ringing Timeout	**Disconnect**$\Downarrow_A$	NULL	**!uses bridge**
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	**!uses bridge**

Figure 8: Specification of *Three-Way Calling* on an active originating (OCM) call.

State	Input	Output	NewState	Assertion
NULL	NULL$\Rightarrow_{OCM}$AUTHORIGATTEMPT	$\ll$*forward msg*$\gg$	WAITANSWER	
WAITANSWER	$\Downarrow_A$**Answered**	$\ll$*forward msg*$\gg$	PRIVATE	
	$\Rightarrow_{3WC}$NULL		NULL	
	$\Downarrow_A$**Disconnect**	$\ll$*forward msg*$\gg$	NULL	
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	
PRIVATE	$\Downarrow_A$**Join Conf**		CONFERENCE	
	$\Rightarrow_{3WC}$NULL		NULL	
	$\Downarrow_A$**Disconnect**	$\ll$*forward msg*$\gg$	NULL	
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	
CONFERENCE	$\Downarrow_A$**Disconnect**	$\ll$*forward msg*$\gg$	NULL	
	$\Uparrow_N$**Release Timeout**	$\ll$*forward msg*$\gg$	NULL	

Figure 9: Specification of *Three-Way Calling* on the new call.

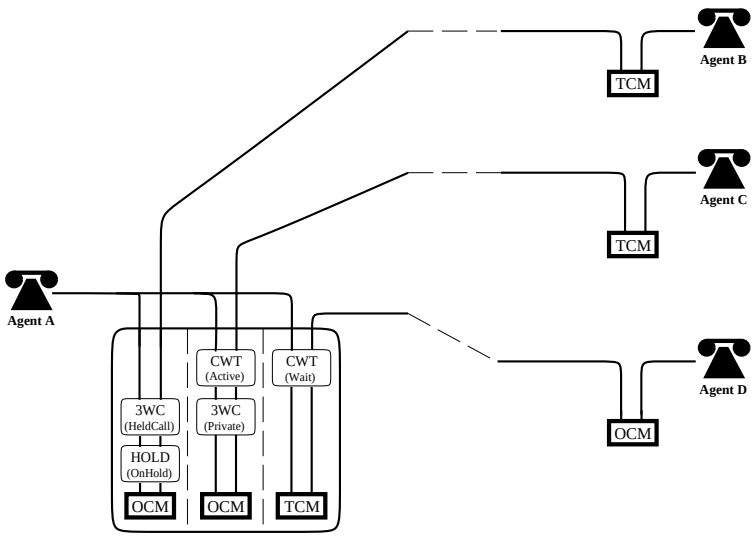


Figure 10: Example of an interaction due to call control interaction.

Attempt). As **Termination Attempt** passes through WAITFORRING the presence of the primed token **origin'** is noted and the information invalidation is detected at this point. Intuitively, *Calling Number Display* is attempting to record for future use information that has been changed[6]. *Calling Number Display* passes the **Termination Attempt** token down towards the underlying call model. When the TCM reaches the state ALERTING, it sends an **Alert** token up to the agent. When this token reaches *Calling Number Display*, it is forwarded up the call stack to the agent. In addition, a new token **Termination Attempt.origin** is created (see Figure 7(d)). Finally, *Calling Number Display* transitions to NULL and the trace is complete.

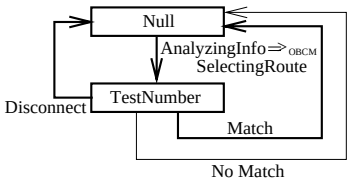
No other interactions are found. There are no assertions, so no assertion invalidations are detected. There are no **uses** so no resource contentions are detected. There were no call control interactions since no states of the two feature machines were ever active on the same call stack.

Example 2: Call Control Interaction

This example demonstrates the detection of a call control interaction between *Call Waiting* and *3-Way Calling*. *3-Way Calling* allows an agent who is engaged on a line to place that line on hold, receive a dial tone, and dial a third party. He may then speak privately with that third party. If he wishes, he may end the connection to the third party and return to the held call, or he may add the third party to the original conversation making it a 3-Way call.

The agent makes his choice by initiating an event. Each possible event can be mapped to one of several input signals [11, 15]. On a simple telephone the events to initiate the second call and to merge it with the first might be mapped to the flashhook. Similarly, the signal to accept an incoming call when *Call Waiting* has been activated might be mapped to the flashhook. In this case the features will interact.

[6]While our algorithm detects the interaction at this point, the effects of the interaction will not be apparent to the agent yet.



State	Input	Output	NewState	Assertion
NULL	$\Downarrow_N$ANALINFO⇒$_{OCM}$SELECTROUTE		TEST	
TEST	▷**Match**	ANALINFO⇒$_{OCM}$EXCEPTION	NULL	¬OCS(target)
	▷**NoMatch**	ANALINFO⇒$_{OCM}$SELECTROUTE	NULL	OCS(target)
	$\Downarrow_A$**Disconnect**	≪*forward msg*≫	NULL	
	$\Uparrow_N$**Release Timeout**	≪*forward msg*≫	NULL	

Figure 11: Specification of the *Originating Call Screening* feature.

For this example we will not produce a full trace, but simply exhibit the state of the compound machine where the interaction occurs. This state can be reached by the following series of events. Agent A, who subscribes to both features, is in a call with B. Agent A decides to call party C and uses *3-Way Calling* to do so. While A is engaged in a private conversation with C, he receives a call from party D, and accepts it using *3-Way Calling*[7]. The state of the composite machine is shown in Figure 10[8]. In this composite state, both state ACTIVE in *Call Waiting* and state PRIVATE in *3-Way Calling* have transitions labeled $\Downarrow_A$**flashhook**, indicating a call control interaction.

Note that it is not enough to simply check if both features accept the same input event coming down the stack. Two features can both accept and act on the same event without an interaction if in the features' composite machine, the features are never ready to accept the event at the same time.

Example 3: Logic Interaction

Originating Call Screening attempts to prevent a connection originating from the agent to any one of a specified set of directory numbers. *Call Forwarding* attempts, under conditions that vary from one version of the feature to another, to connect an incoming call to a new target. *Call Forwarding* can interfere with *Originating Call Screening*. If agent A has placed agent C's number in his *Originating Call Screening* screening list but has not placed agent B's number in the list, and if B has redirected calls from his number to C's number, then A may reach C by dialing B.

We model *Originating Call Screening* as follows. Certain transitions in the *Originating Call Screening* machine are labeled with $\models$**OCS(target)**. This indicates that the **target** (directory number) passed the test *Originating Call Screening* applied. **OCS(target)** can be thought of as a label applied to successful paths through the *Originating Call Screening*

[7]In reality, this situation cannot in fact occur due to the resource contention between these features which we describe in Example 4.

[8]When *3-Way Calling* is used to initiate a new call, the *3-Way Calling* places the active line on hold by invoking the feature *Hold* on the call stack of the active line.

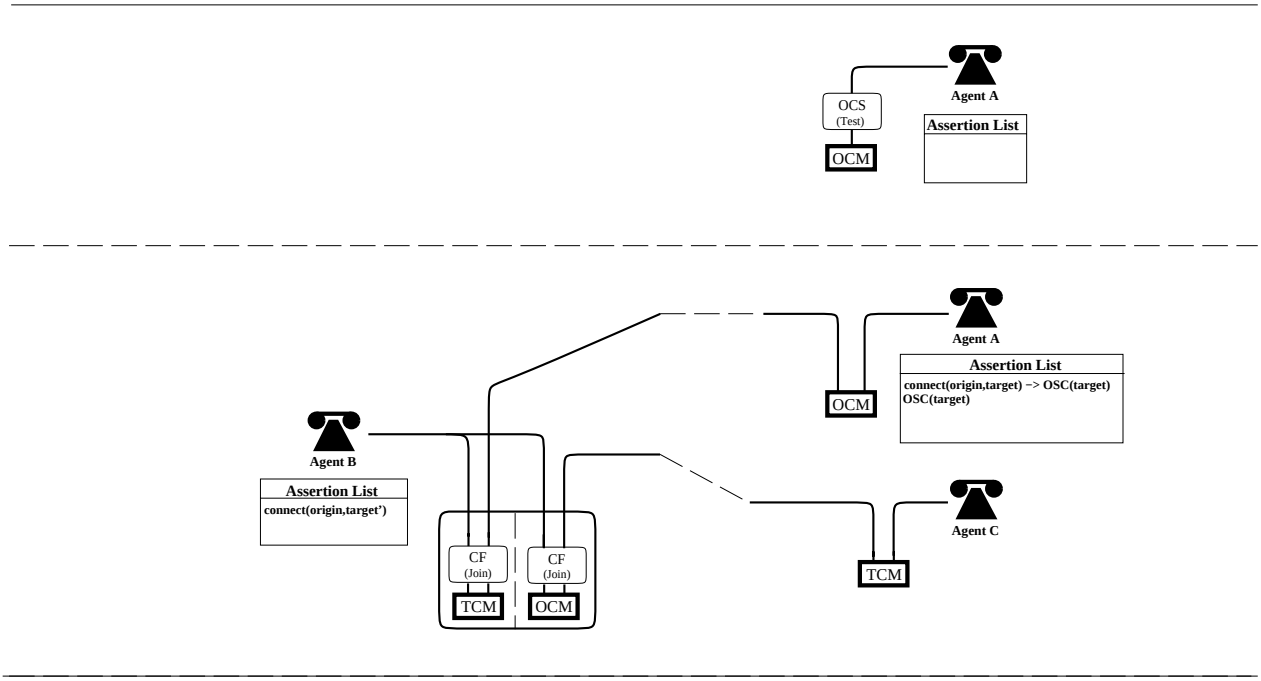


Figure 12: Example of an interaction due to an assertion invalidation.

machine. *Originating Call Screening* also asserts $\models$**connect(A, X)** $\rightarrow$**OCS(X)**, which states that along any path resulting in a connection originating at A and terminating at X, assertion **OCS(X)** must hold. (In the above assertion, A is the agent that invokes the *Originating Call Screening* feature, and X is a free variable.) The **connect** assertion between A and X may be raised by the tracing algorithm when the connection is made by the underlying call models, or may be raised by another feature that implements a virtual connection. This state of the composite machine is shown at the top of Figure 12.

When *Call Forwarding* is activated on an incoming call, *Call Forwarding* initiates a new call from the agent (who is the terminus of the incoming call) to a new destination agent, and forms a virtual connection from the originating agent of the first call (**origin**) to the new destination agent (**target'**). When *Call Forwarding* has done all the preliminary signaling necessary to form the virtual connection, it transitions into the JOIN state and raises the assertion $\models$**connect(origin, target')**.

The bottom of Figure 12 shows the interactions of these two features when party A invokes *Originating Call Screening* and party B forwards all of his calls to party C. Note that the interaction occurs after *Originating Call Screening* has deactivated. In this state of the composite machine, assertion **connect(origin,target')** has been raised but the required assertion **OCS(target')** has not.

Example 4; Resource Interaction

3-Way Calling and *Call Waiting* both require the use of a piece of hardware known as a bridge, but there is only one bridge available to an agent. If an agent attempts to use both features at the same time there will be a resource contention. Figure 13 shows an abbreviated trace. Initially, the agent is involved in a call. He receives an incoming call and accepts it using *Call Waiting*. When *Call Waiting* is invoked, it asserts **uses bridge**. This

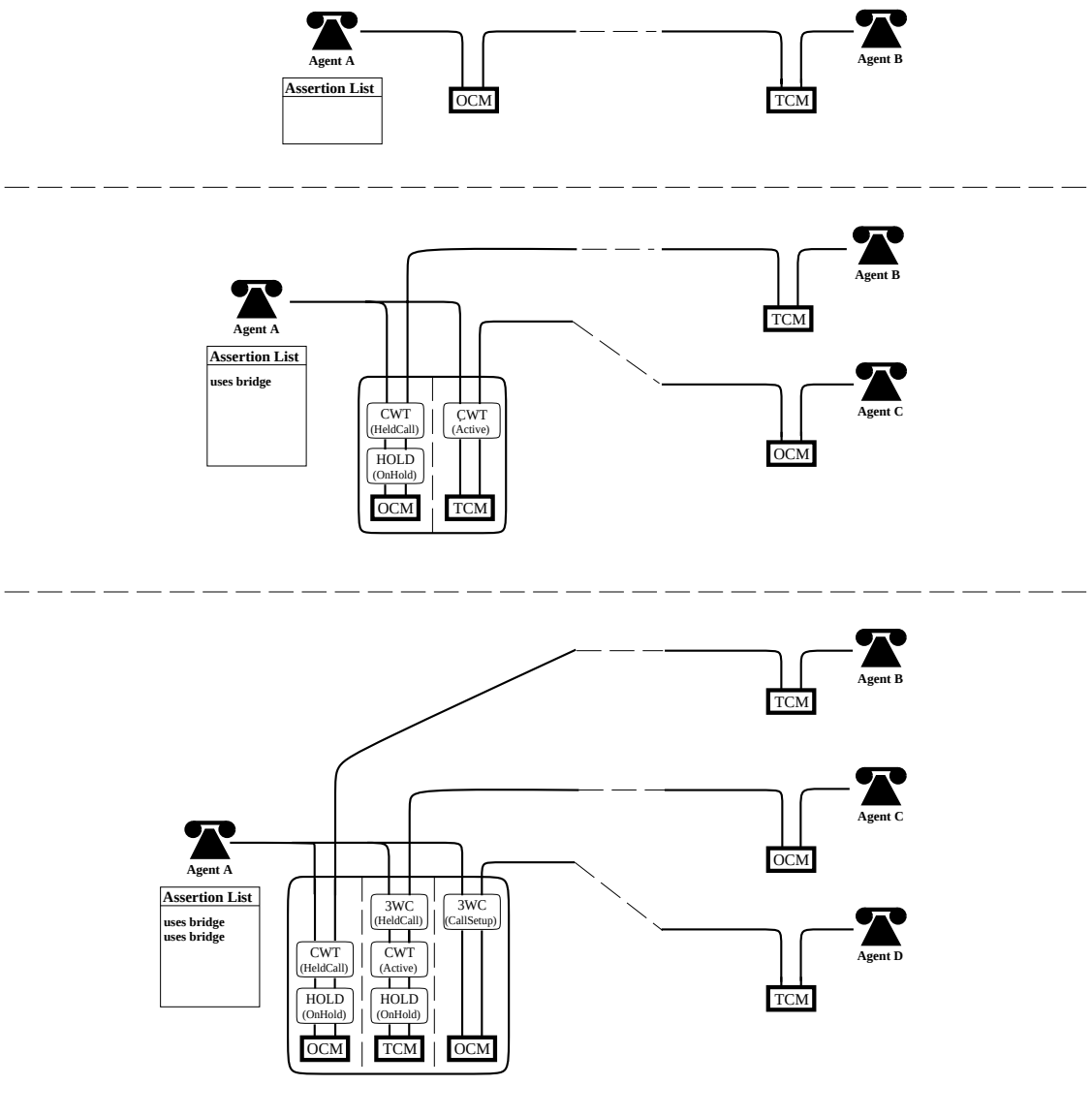


Figure 13: Example of an interaction due to resource contention.

is shown in the subscriber's assertion list in the middle diagram in Figure 13. When the subscriber subsequently invokes *3-Way Calling*, a second **uses bridge** is asserted (see the bottom diagram in Figure 13). This assertion will also go into the subscriber's assertion list. This duplicate insertion indicates a resource contention. The same contention would arise if the features were activated in reverse order.

Conclusion

The work described in [9] and [13] propose template specifications of features. One of the major advantages of a template notation is the power of its expressibility. As a result, the specifications contain enough information to be able to detect a number of different types of feature interactions.

The work described in [1], [5], and [14] propose formal specifications of features. One of the major advantages of a formal approach is that mathematical analysis techniques can be applied to the composition of feature specifications to determine if the features behave correctly. [1] describes a specification environment that provides automated support for formal specification, refinement, and simulation of telephone features.

We have taken a middle-of-the-road approach. In this paper, we have presented a tabular notation that is flexible enough to support the specification of a wide variety of telephony events and properties, but is rigorous enough to allow automated analysis. Using the call stack model of a telephone call [12], we have sketched algorithms for determining the set of composite machines associated with a pair of features and for tracing through the reachability graphs of the composite machines. We have also described how to detect four types of feature interactions (call control interaction, information invalidation, resource contention, and assertion invalidation) by examining the information known at each state of the composite machines' reachability graphs. Our next goal is to automate the algorithms presented in this paper.

References

[1] R. Boumezbeur and L. Logrippo. "Specifying Telephone Systems in LOTOS". *IEEE Communications*, 31(8):38–45, August 1993.

[2] R. Brooks. "A Robust Layered Control System for a Mobile Robot". *IEEE Journal of Robotics and Automation*, RA-2:14–23, April 1986.

[3] E.J. Cameron, N. Griffeth, Y. Lin, and H. Velthuijsen. "Definitions of Services, Features, and Feature Interactions", December 1992. Bellcore Memorandum for Discussion, presented at the International Workshop on Feature Interactions in Telecommunications Software Systems.

[4] E.J. Cameron, N. Griffeth, Y.J. Lin, M. Nilson, W. Schnure, and H. Velthuijsen. "A Feature Interaction Benchmark in IN and Beyond". Technical Report TM-TSV-021982, Network Systems Specifications Research, Bell Communications Research, September 1992.

[5] A. Fekete. "Formal Models of Communication Services: A Case Study". *IEEE Computer*, 26(8):37–47, August 1993.

[6] A. Flynn, R. Brooks, and L. Tavrow. "Twilight Zones and Cornerstones: A Gnat Robot Double Feature". Technical Report A.I. Memo 1126, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1989.

[7] N. Griffeth and Y. Lin. "Extending Telecommuncations Systems: The Feature-Interaction Problem". *IEEE Computer*, 26(8):14–18, August 1993.

[8] K. Heninger. "Specifying Software Requirements for Complex Systems: New Techniques and Their Applications". *IEEE Transactions on Software Engineering*, SE-6(1):2–12, January 1980.

[9] E. Kuisch, R. Janmaat, H. Mulder, and I. Keesmaat. "A Practical Approach to Service Interactions". *IEEE Communications*, 31(8):24–31, August 1993.

[10] Northern Telecom. *DMS-100 Meridian Digital Centrex Library*, 50039.08/12-92 issue 1 edition, 1992.

[11] D. Parnas and J. Madey. Functional Documentation for Computer Systems Engineering (Version 2). Technical Report CRL Report 237, Department of Electrical and Computer Engineering, McMaster University, 1991.

[12] G. Utas. "Feature Processing Environment", December 1992. Presented at the International Workshop on Feature Interactions in Telecommunications Software Systems.

[13] Y. Wakahara, M. Fujioka, H. Kikuta, and H. Yagi. "A Method for Detecting Service Interactions". *IEEE Communications*, 31(8):32–37, August 1993.

[14] P. Zave. "Feature Interactions and Formal Specifications in Telecommunications". *IEEE Computer*, 26(8):20–30, August 1993.

[15] P. Zave and M. Jackson. "Conjunction as Composition". *ACM Transactions on Software Engineering and Methodology*, 2(4):379–411, October 1993.

Appendix A: Specification of Call Model

Tables 4 and 5 contain the tabular specifications of the originating basic call model (OBCM) and the terminating basic call model (TBCM), respectively.

State	Input	Output	NewState
Null	$\Downarrow_A$**Origination Attempt**		AuthOrigAttempt
AuthOrig	$\triangleright$Originated	**Collect Info**$\Uparrow_A$	CollectingInfo
Attempt	$\triangleright$Origination Denied	**Origination Denied**$\Uparrow_A$	Exception
	$\Downarrow_A$**Disconnect**		Null
Collecting	$\Downarrow_A$**Info Collected**		AnalyzingInfo
Info	$\triangleright$Collection Timeout	**Collection Timeout**$\Uparrow_A$	Exception
	$\Downarrow_A$**Disconnect**		Null
Analyzing	$\triangleright$Valid Info		SelectingRoute
Info	$\triangleright$Invalid Info	**Invalid Info**$\Uparrow_A$	Exception
	$\Downarrow_A$**Disconnect**		Null
Selecting	$\triangleright$Route Selected		AuthCallSetup
Route	$\triangleright$Network Busy	**Network Busy**$\Uparrow_A$	Exception
	$\Downarrow_A$**Disconnect**		Null
Auth	$\triangleright$Call Setup Authorized	**Call Request**$\Uparrow_N$	SendCall
CallSetup	$\triangleright$Call Setup Denied	**Call Set Denied**$\Uparrow_A$	Exception
	$\Downarrow_A$**Disconnect**		Null
SendCall	$\Downarrow_N$**Call Delivered**	**Call Delivered**$\Uparrow_A$	Alerting
	$\Downarrow_N$**Route Busy**		SelectingRoute
	$\Downarrow_N$**Answered**	**Answered**$\Uparrow_A$	Active
	$\Downarrow_N$**Called Party Busy**	**Called Party Busy**$\Uparrow_A$	Exception
	$\Downarrow_N$**Call Cleared**	**Call Cleared**$\Uparrow_A$	Exception
	$\Downarrow_A$**Disconnect**	**Call Cleared**$\Uparrow_N$	Null
Alerting	$\Downarrow_N$**Answered**		Active
	$\Downarrow_N$**Called Party Busy**		Exception
	$\Downarrow_N$**Call Cleared**	**Call Cleared**$\Uparrow_A$	Exception
	$\Downarrow_A$**Disconnect**	**Call Cleared**$\Uparrow_N$	Null
Active	$\Downarrow_N$**Call Cleared**		ReleasePending
	$\Downarrow_A$**Disconnect**	**Call Cleared**$\Uparrow_N$	Null
Release	$\Downarrow_N$**Called Party Reconnect**		Active
Pending	$\triangleright$Release Timeout	**Release Timeout**$\Uparrow_N$	Null
	$\Downarrow_A$**Disconnect**	**Call Cleared**$\Uparrow_N$	Null
Exception	$\Downarrow_A$**Disconnect**		Null

Table 4: Specification of Basic Call Model for Originating Caller.

State	Input	Output	NewState
Null	$\Downarrow_N$**Termination Attempt**		AuthTermination
Auth Termination	▷Call Presented		HuntingFacility
	▷Termination Denied	**Call Cleared**$\Uparrow_N$	Exception
	$\Downarrow_N$**Call Cleared**		Null
	$\Downarrow_A$**Disconnect**	**Call Cleared**$\Uparrow_N$	Null
Hunting Facility	▷Facility Found		PresentingCall
	▷Busy	**Called Party Busy**$\Uparrow_N$	Exception
	$\Downarrow_N$**Call Cleared**		Null
	$\Downarrow_A$**Disconnect**	**Call Cleared**$\Uparrow_N$	Null
Presenting Call	▷Call Accepted	**Call Delivered**$\Uparrow_N$ **Alert**$\Uparrow_A$	Alerting
	▷Call Failure		HuntingFacility
	▷Call Rejected	**Call Cleared**$\Uparrow_N$	Exception
	$\Downarrow_A$**Connected**	**Answered**$\Uparrow_N$	Active
	$\Downarrow_N$**Call Cleared**		Null
Alerting	$\Downarrow_A$**Connected**	**Answered**$\Uparrow_N$	Active
	▷Call Rejected	**Call Cleared**$\Uparrow_N$	Exception
	▷Ringing Timeout	**Call Cleared**$\Uparrow_N$	Exception
	$\Downarrow_N$**Call Cleared**		Null
Active	$\Downarrow_A$**Disconnect**	**Call Cleared**$\Uparrow_N$	ReleasePending
	$\Downarrow_N$**Call Cleared**		Null
Release Pending	$\Downarrow_A$**Called Party Reconnect**	**Called Party Reconnect**$\Uparrow_N$	Active
	$\Downarrow_N$**Release Timeout**		Null
	$\Downarrow_N$**Call Cleared**		Null
Exception			Null

Table 5: Specification of Basic Call Model for Terminating Called Party.