Evaluation of a State-Based Model of Feature Interactions

Pansy K. Au Joanne M. Atlee*
Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1

Abstract. At the second workshop on feature interactions, we proposed a state-based model for specifying features and proposed reachability-analysis algorithms for detecting various types of feature interactions. This paper is an evaluation of our approach with respect to the Bellcore benchmark of feature interactions.

1 Introduction

A *feature* is a sub-program that adds functionality to an existing software system. A feature *interaction* occurs when the addition of a new feature affects the behavior of existing services and features. That is, there is an interaction between features f and g if feature f behaves one way when g is present and another way when feature g is absent. Most software developers associate 'feature interactions' with unintended interactions between features that were thought to be unrelated. However, since the purpose of a feature is to modify or enhance functionality, features by definition interact; at the very least, a new feature is expected to interact with those features and/or services whose functionality is intentionally modified by the new feature. Thus, the problem of detecting feature interactions is twofold: we want to validate specified interactions and to detect unspecified interactions.

At the second workshop on feature interactions [5], we proposed graphical and tabular notations, based on a state-transition model, for specifying the functional behavior of telephone services and features. We also presented algorithms for composing feature specifications and for detecting certain types of interactions. Since then, we have developed these ideas into a suite of prototype tools that support automated detection of feature interactions [1, 16, 15].

In this paper, we evaluate our approach to feature specification and interaction detection with respect to a benchmark of feature interactions, published by Bellcore [6]. We have only attempted to specify one of the six services listed in the benchmark: Plain Old Telephone Service (POTS). In addition, we found we were able to specify 15 of the 19 features used in the benchmark, and were able to detect 11 of the 23 interactions, with the hope of eventually being able to detect 15 interactions.

2 Specification Model

The behaviors of features are modelled as state transition machines (STMs), where each transition is triggered by a single input event [5]. Each feature is specified independently of other features. Multiple STM's can be composed together to form another STM, which represents the reachability graph [1, 16, 15]. During the composition, tests for detecting interactions are run in each reachable state.

*This research has been supported by Natural Sciences and Engineering Research Council of Canada, with matching funds from Nortel.

Table 1: Descriptions of Events and Assertions

Events and Assertions	**Descriptions**
Token Events	
$\Downarrow_A$token, $\Uparrow_A$token	- *Agent Token Events:* tokens passed down from or up to the agent
$\Downarrow_R$token, $\Uparrow_R$token	- *Remote Token Events:* tokens passed down from or up to the remote user
State Transition Events	
$\xrightarrow{O}_f$	- *Enable Event:* feature f is ready to become active
$\xrightarrow{O}_f$S(e)	- *Activation Event:* feature f has become active and is in state S
S1$\xrightarrow{R}_f$S2(e)	- *RequestStateChange Event:* feature f requests to transition from S1 to S2
$_g$S1$\xrightarrow{M}_f$S2(e)	- *ModifyStateChange Event:* feature g forces f to transition from S1 to S2
S1$\xrightarrow{O}_f$S2(e)	- *OccurredStateChange Event:* feature f has made a transition from S1 to S2
Sibling Events	
S1$\xrightarrow{O}_{P(f)}$S2	- *Parallel Transition Event:* feature f of the sibling machine is in state S2
$\Rightarrow_{P(f)}$token, $\Leftarrow_{P(f)}$token	- *Parallel Input Event:* tokens passed from or to sibling feature
NewCall$_f$(BCM)	- *NewCall Event:* start a new stack with basic service BCM and feature f
Internal Events	
$\triangleright_f$event	- *Internal Event:* internally generated event by feature f
$*$	- *Wildcard state or token*
$\vDash$A	- *Assertion raised:* property A is asserted to be true, and will continue to be asserted until explicitly un-asserted
!A	- *Assertion lowered:* property A is un-asserted

2.1 Specification Notation

Features are specified as State Transition Machines using a tabular notation. Each row in the table represents a transition from an old state to a new state, due to the occurrence of an input event. As a side effect of the transition, an output event might be issued or an assertion raised (e.g., the assertion of an assumption). A feature f can control the behavior of a second feature g in two ways: it can communicate with the second feature via signal passing (Token events). Alternatively, it can control behavior by monitoring, rejecting, or modifying state transitions made by the second feature (Activation event and StateChange events). Descriptions of all the events and assertions defined in the notation are given in Table 1.

In accordance with the Advanced Intelligent Network (AIN) architecture, the originating end of a call (called the Originating Call Model, or simply OCM) and the receiving end of a call (called the Terminating Call Model, or simply TCM) are modelled as separate, communicating STMs. If a feature f is intended to modify the behavior of a basic call, then we write a specification of f that modifies an OCM and a different specification f' that modifies a TCM.

The specifications of the Call Forwarding feature are given in Tables 2 and 3. Call Forwarding incoming (CF_1) modifies the behavior of a call received by a subscriber who has invoked the CF feature; that is, it modifies a TCM (Table 2). Call Forwarding outgoing (CF_2) spawns the forwarded call; it imitates the OCM in a call from the subscriber who invoked CF to the forwarded number. CF_1 suspends the TCM of the initial call in state *AuthTermination*, thereafter simulating the TCM's functionality. CF_2 suspends the OCM of the forwarded call in state *AuthOrigAtt*, thereafter simulating the OCM's functionality. Data is passed between CF_1 and CF_2 as parallel tokens. Thus, if a user A calls a subscriber B, who has forwarded all calls to user C, the CF machines act as intermediaries that pass signals between A and C.

2.2 Levels of Specification and Composition

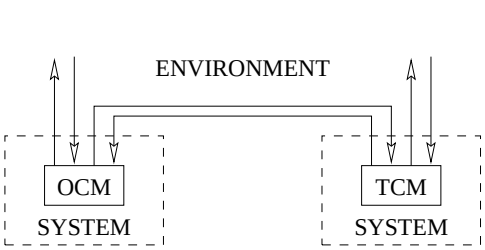
Architecturally, call configurations are modelled at four different levels: *Feature*, *Stack*, *Call*, and *Call-Group*. At each level, the model is divided into the *environment* and the *system*. The *system* refers to the part of the model that is of particular interest (enclosed in a dotted

State	Input Event	Output Event	NewState	Resource	Assertions
Ready	$\xrightarrow{O}_{TCM}$AuthTermination($\Downarrow_R$TerminationAttempt)		Receiving		
Receiving	AuthTermination$\xrightarrow{R}_{TCM}$HuntingFacility($\triangleright_{TCM}$CallPresented)	NewCall$_{CF}$(OCM), $\Uparrow_A$ShortAlert	WaitSibling		
	AuthTermination$\xrightarrow{O}_{TCM}$Exception($\triangleright_{TCM}$TerminationDenied)		Exception		
	$*\xrightarrow{O}_{TCM}$Exception($*$)		Null		
	$*\xrightarrow{O}_{TCM}$Null($*$)		Null		
WaitSibling	$*\xrightarrow{O}_{P(CF)}$Originated	$\Leftarrow_{P(CF)}$OriginationAttempt.s	Forwarding		
	$\Downarrow_A *$		WaitSibling		
	$*\xrightarrow{O}_{TCM}$Exception($*$)		Exception		
	$*\xrightarrow{O}_{TCM}$Null($*$)		Null		
	$*\xrightarrow{O}_{P(CF)}$Null	$_{CF}$AuthTermination$\xrightarrow{M}_{TCM}$Null($\Downarrow_A$Disconnect)	Null		
Forwarding	$\Rightarrow_{P(CF)}$CallDelivered	$\Uparrow_R$CallDelivered, $\Uparrow_A$Alert	Alerting		
	$\Rightarrow_{P(CF)}$Answered	$\Uparrow_R$Answered	Connected		!lconn(t,d)
	$\Rightarrow_{P(CF)}$CalledPartyBusy	$\Uparrow_R$CalledPartyBusy	Exception		
	$\Rightarrow_{P(CF)}$CallCleared	$\Uparrow_R$CallCleared	Exception		
	$*\xrightarrow{O}_{TCM}$Exception($*$)		Exception		
	$\Downarrow_R$CallCleared	$\ll$forward$\gg$	Null		
	$*\xrightarrow{O}_{TCM}$Null($*$)		Null		
	$\Downarrow_A *$		Forwarding		
Alerting	$\Rightarrow_{P(CF)}$Answered	$\Uparrow_R$Answered	Connected		!lconn(t,d)
	$\Rightarrow_{P(CF)}$CalledPartyBusy	$\Uparrow_R$CalledPartyBusy	Exception		
	$\Rightarrow_{P(CF)}$CallCleared	$\Uparrow_R$CallCleared	Exception		
	$*\xrightarrow{O}_{TCM}$Null($*$)		Null		
	$\Downarrow_A *$		Alerting		
Connected	$\Rightarrow_{P(CF)}$CallCleared	$\Uparrow_R$CallCleared	ReleasePend		
	$*\xrightarrow{O}_{TCM}$Null($*$)		Null		
	$\Downarrow_A *$		Connected		
ReleasePend	$\Rightarrow_{P(CF)}$CalledPartyReconnect	$\Uparrow_R$CalledPartyReconnect	Connected		
	$\Downarrow_R$ReleaseTimeout	$_{CF}$AuthTermination$\xrightarrow{M}_{TCM}$Null($\Downarrow_R$CallCleared)	Null		
	$*\xrightarrow{O}_{TCM}$Null($*$)		Null		
	$\Downarrow_A *$		ReleasePend		
Exception	$*\xrightarrow{O}_{P(CF)}$Null	$_{CF}$AuthTermination$\xrightarrow{M}_{TCM}$Null($\Downarrow_R$CallCleared)	Null		
	$*\xrightarrow{O}_{TCM}$Null($*$)		Null		
	$\Downarrow_A *$		Exception		

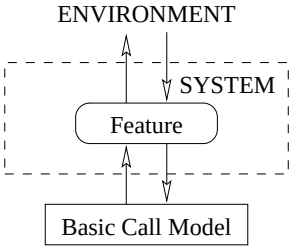
Table 2: Specification of Call Forward (All Calls) on the incoming leg of a call

State	Input	Output Event	NewState	Resource	Assertions
Ready	$\xrightarrow{O}_{OCM}$AuthOrigAtt(∗)		SetUp		
SetUp	AuthOrigAtt$\xrightarrow{R}_{OCM}$CollectInfo($\triangleright_{OCM}$Orig)		Originated		
	∗$\xrightarrow{O}_{OCM}$Exception(∗)		Exception		
	∗$\xrightarrow{O}_{P(CF)}$Null	$_{CF}$AuthOrigAtt$\xrightarrow{M}_{OCM}$Null($\Downarrow_A$Disconnect)	Null		
	∗$\xrightarrow{O}_{OCM}$Null(∗)		Null		
Originated	$\Rightarrow_{P(CF)}$OriginationAttempt.t		SelectRoute		
	∗$\xrightarrow{O}_{P(CF)}$Null	$_{CF}$AuthOrigAtt$\xrightarrow{M}_{OCM}$Null($\Downarrow_A$Disconnect)	Null		
	$\Downarrow_A$∗		Originated		
SelectRoute	$\triangleright_{CF}$RouteSelected		AuthCall		
	$\triangleright_{CF}$NetworkBusy	$\Uparrow_A$NetworkBusy	Exception		
	∗$\xrightarrow{O}_{P(CF)}$Null	$_{CF}$AuthOrigAtt$\xrightarrow{M}_{OCM}$Null($\Downarrow_A$Disconnect)	Null		
	$\Downarrow_A$∗		SelectRoute		
AuthCall	$\triangleright_{CF}$CallSetupAuthorized	$\Uparrow_R$CallRequest	SendCall		
	$\triangleright_{CF}$CallSetupDenied	$\Uparrow_A$CallSetupDenied	Exception		
	∗$\xrightarrow{O}_{P(CF)}$Null	$_{CF}$AuthOrigAtt$\xrightarrow{M}_{OCM}$Null($\Downarrow_A$Disconnect)	Null		
	$\Downarrow_A$∗		AuthCall		
SendCall	$\Downarrow_R$CallDelivered	$\Leftarrow_{P(CF)}$CallDelivered	Alerting		
	$\Downarrow_R$Answered	$\Leftarrow_{P(CF)}$Answered	Connected		⊨lconn(t,d), ⊨pconn(s,d)
	$\Downarrow_R$CalledPartyBusy	$\Leftarrow_{P(CF)}$CalledPartyBusy	Exception		
	$\Downarrow_R$CallCleared	$\Leftarrow_{P(CF)}$CallCleared, $_{CF}$AuthOrigAtt$\xrightarrow{M}_{OCM}$Null($\Downarrow_A$Disconnect)	Null		
	∗$\xrightarrow{O}_{P(CF)}$Null	$\Uparrow_R$CallCleared, $_{CF}$AuthOrigAtt$\xrightarrow{M}_{OCM}$Null($\Downarrow_A$Disconnect)	Null		
	$\Downarrow_A$∗		SendCall		
Alerting	$\Downarrow_R$Answered	$\Leftarrow_{P(CF)}$Answered	Connected		⊨lconn(t,d), ⊨pconn(s,d)
	$\Downarrow_R$CalledPartyBusy	$\Leftarrow_{P(CF)}$CalledPartyBusy	Exception		
	$\Downarrow_R$CallCleared	$\Leftarrow_{P(CF)}$CallCleared	Exception		
	∗$\xrightarrow{O}_{P(CF)}$Null	$\Uparrow_R$CallCleared, $_{CF}$AuthOrigAtt$\xrightarrow{M}_{OCM}$Null($\Downarrow_A$Disconnect)	Null		
	$\Downarrow_A$∗		Alerting		
Connected	$\Downarrow_R$CallCleared	$\Leftarrow_{P(CF)}$CallCleared	ReleasePend		
	∗$\xrightarrow{O}_{P(CF)}$Null	$\Uparrow_R$CallCleared, $_{CF}$AuthOrigAtt$\xrightarrow{M}_{OCM}$Null($\Downarrow_A$Disconnect)	Null		!lconn(t,d), !pconn(t,d)
	$\Downarrow_A$∗		Connected		
ReleasePend	$\Downarrow_R$CalledPartyReconnect	$\Leftarrow_{P(CF)}$CalledPartyReconnect	Connected		
	∗$\xrightarrow{O}_{P(CF)}$Null	$\Uparrow_R$CallCleared, $_{CF}$AuthOrigAtt$\xrightarrow{M}_{OCM}$Null($\Downarrow_A$Disconnect)	Null		!lconn(t,d), !pconn(t,d)
	$\Downarrow_A$∗		ReleasePend		
Exception	∗$\xrightarrow{O}_{P(CF)}$Null	$_{CF}$AuthOrigAtt$\xrightarrow{M}_{OCM}$Null($\Downarrow_A$Disconnect)	Null		
	$\Downarrow_A$∗		Exception		

Table 3: Specification of Call Forward (All Calls) on the outgoing leg of a call



(a) Basic Service



(b) Feature

Figure 1: Feature Level

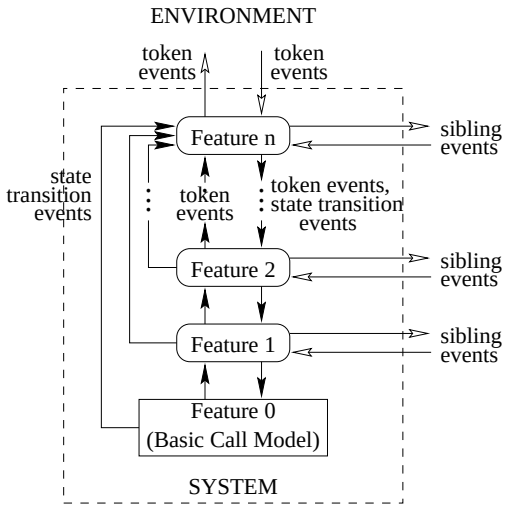


Figure 2: Stack Level

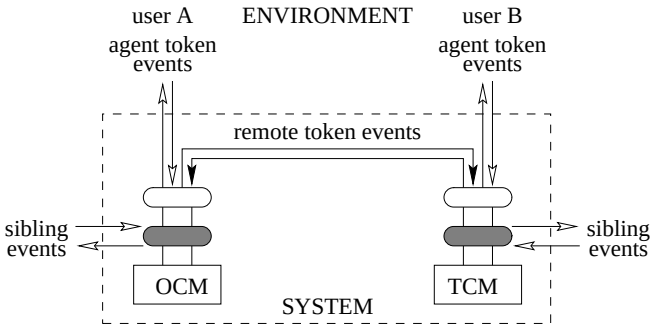


Figure 3: Call Level

rectangle in the diagrams, please see Figures 1a and 1b for examples) and the *environment* refers to the elements in the model with which the system interacts (everything outside of the dotted rectangle). Each architectural level focuses on a particular aspect of the configuration.

The *feature* level describes the behavior of an individual feature or service. A *service* is a stand-alone functionality that is complete in itself. Figure 1a shows our model of the Plain Old Telephone Service (POTS) which provides basic call processing activities. The systems of interest here are the two ends of POTS: OCM and the TCM. The environment for each system is the human user of the system (the agent) and the connection to the other service (the remote user). That is, each basic call model interacts with its agent and with its remote counterpart. A *feature*, on the other hand, is an addition or modification to an existing service; it cannot stand by itself. A feature modifies the behavior of the basic service by acting as a mediator between the basic call model's environment and the basic call model (Figure 1b). Information passed from the basic call model's environment is first seen and possibly modified by the feature before it is passed to the basic call model.

The *stack* level describes the behavior of a single end of a call and represents the composition of all the features activated on that end of the call (Figure 2). The feature at the top of the stack is said to have the highest priority in the stack, since it is the first to intercept inputs from the environment and the last to modify outputs to the environment. For the same reason, the feature at the bottom of the stack (the basic service) is said to have the lowest priority. Within the system of a stack, communication among the features consists of *token events* and *state transition events*. Token events represent data passed between adjacent features in the stack. State transition events represent requests to transition to a new state or notifications of a transition made (higher priority features can control lower priority features by rejecting or

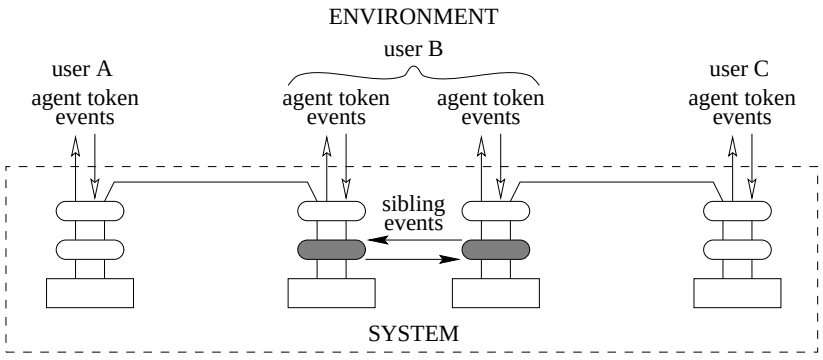


Figure 4: Call-Group Level

modifying state transition requests). At the stack composition level, communication among features in a stack is processed until all issued token events and state transition events are resolved.

The *call* level describes the behavior of a call. It is the composition of two stacks, each representing an end of the call, connected by a communication channel (Figure 3). The caller and the callee communicate by passing *remote token events*, which are designated for the remote user. Remote token events from one stack are passed as input to the other stack.

Finally, the behavior of multiple calls involving common users is depicted as a *call-group*, and represents the composition of several call specifications (see Figure 4). A feature involved in multiple calls, such as Call Waiting or Three-Way-Calling, would have a representative feature specification in each of the user's stack (shown in grey in Figures 3 and 4); such features are called *sibling features* and can communicate via *sibling events*.

3 Detection of Interactions

During the composition, each reachable state is tested to determine if an interaction can occur at that state. Composition and detection algorithms have been implemented, and we are able to automatically detect four types of interactions: control modification, data modification, resource contention and unreachable interactions. We have provided a basis for detecting a fifth type of interaction, assertion violation, for which an analysis algorithm is currently being implemented. Examples of these interactions are discussed below. The examples are taken from the Bellcore Benchmark [6] (discussed in Section 4) and the example numbers given in [6] are referenced.

3.1 Control Modifications

A control modification interaction occurs when one feature affects the flow-of-control in another feature. This can be done explicitly, by forcing the second feature to transition to a new state (which is only possible if the two features execute in the same feature stack); or implicitly, by intercepting, modifying or introducing data sent to the second feature.

3.1.1 Control of tokens

Features can communicate with each other by passing tokens. Since token events can trigger features' transitions, behaviors of features can be altered by intercepting the tokens passed to the features. When a feature in a stack receives a token, it can either pass it unchanged to the next feature, pass a modified token to the next feature, or consume the token by not passing it on. Passing a received token allows other features to react to the token. Consuming a token in effect prevents features from reacting to this token. For example, using the HOLD feature, a user can press the hold button and then put the receiver on hook without disconnecting the

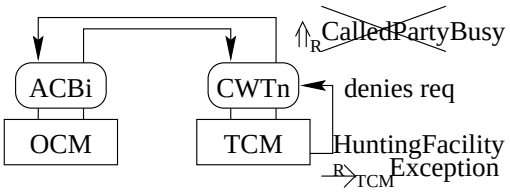


Figure 5: Control Modifications (No. 13)

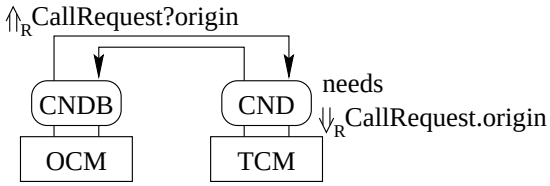


Figure 6: Data Modifications (No. 16)

call. Essentially, HOLD intercepts the Disconnect token, thereby preventing the Basic Call Model from receiving the Disconnect request and ending the call. This type of interaction is detected when more than one feature in a feature stack is ready to input the same token, a high-priority feature consumes or alters the token before it reaches other features in the stack.

3.1.2 Control of features' state-transitions

A feature can also control another feature's behavior by denying or modifying a transition request. Before a feature makes a transition, permission must be obtained from all features that have higher priority. When a higher priority feature receives a state-transition request from a lower priority feature, it can either grant permission, deny permission, or force the requesting feature to make an alternate transition. This type of interaction is detected whenever a state-transition request is denied or modified. Since state-transition requests and events are only passed among features executing in the same stack, we only search for this interaction during composition of a stack.

A side effect of disallowing a state-transition is that the output events (i.e. output tokens) associated with the transition do not occur. This type of interaction is detected when a denied or modified state-transition would have output one or more tokens.

Example : Call Waiting and Automatic CallBack (No. 13)
Call Waiting (CWT) enables a user to receive and answer a second call while the phone is busy. This is achieved by preventing the Terminating Call Model (TCM) of the second call from entering state Exception and sending a Busy token to the caller. Instead, CWT will determine whether facilities exist to support the call, and if so, instruct the TCM to set up the call. Automatic CallBack (ACB) helps a caller to eventually establish a connection to a party that is currently busy. If a user A calls B and gets a busy signal, then A can activate ACB, which will monitor B's phone line. When B hangs up, the switch will ring A; if A picks up, the switch will automatically ring B and set up a connection.

Suppose user A has ACB and user B has CWT, as shown in Figure 5. A calls B while B is talking to another user C. Although B is busy, A will not be able to activate ACB because A's feature stack will not receive a Busy token from B's stack. There are actually two control modification interactions here. One is the explicit interaction of CWT denying a state-transition in the TCM (the transition to the Exception state). The second is an implicit interaction that occurs because the Busy token, which would have been output by the denied state-transition, is not sent.

3.2 Data Modifications

A data modification interaction occurs when a token received by one feature has been modified by another. When a feature receives a token, it has the ability to modify its values. A question mark '?' is used to indicate when a token value has been modified. A data modification interaction is detected when a feature accepts a token that contains symbol '?'.

Example : Call Number Delivery Blocking and Call Number Delivery (No. 16)
Call Number Delivery (CND) allows the callee to see the caller's phone number. Call Number Delivery Blocking (CNDB) modifies the caller's number in the CallRequest to-

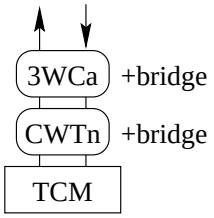


Figure 7: Resource Contentions (No. 2b)

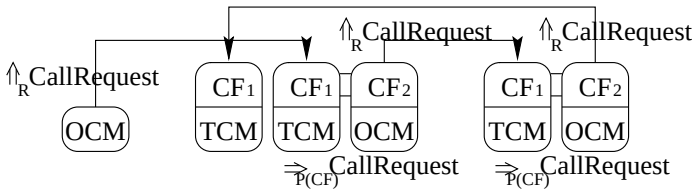


Figure 8: Reachability (No. 17)

ken so that the real number is not delivered. Suppose user A, who has CNDB, calls user B, who has CND. The CNDB feature modifies A's number so that B's stack receives token $\Downarrow_R$CallRequest?origin instead of token $\Downarrow_R$CallRequest.origin (please see Figure 6). At this point, a data modification interaction is detected.

3.3 Resource Contentions

A resource contention interaction occurs when the number of requested resources is greater than the number of resources available. This is detected by counting the number of resources currently used by each user during the call process. When the number of allocated resources is equal to the number of available resources, an additional request will result in a resource contention interaction.

Example : Three-Way-Calling and Call Waiting (No. 2b)
Both Three-Way-Calling (3WC) and Call Waiting (CWT) require the use of a resource known as a bridge. 3WC uses a bridge to support origination of a second call, and CWT uses a bridge to support the receipt of a second call. However, the switch only allocates 1 bridge to each telephone line card. Suppose a user A subscribes both CWT and 3WC (please see Figure 7). Suppose further that A is talking with B, and uses CWT to accept a phone call from C. A resource contention will occur if A subsequently attempts to invoke 3WC.

3.4 Reachability

A reachability interaction occurs when a state in a feature's specification becomes unreachable after the composition. This is detected by comparing the set of reachable states before and after the composition. A reachable state might appear to be unreachable if it becomes an intermediate state in the composition. More commonly, a state is unreachable because the transition(s) into the state has been explicitly denied or its input event never occurs.

Example : Call Forwarding and Call Forwarding (No. 17)
Suppose user A has Call Forwarding (CF) and has forwarded their calls to user B. Suppose B also has CF and has forwarded their calls to A. If a user calls A, the call is forwarded to B, then forwarded to A, then forwarded to B and so on (see Figure 8). This interaction is detected as a reachability interaction because many of the normally reachable states in the underlying call models are never reached.

3.5 Assertion Violations

An assertion-violation interaction occurs when the set of currently raised assertions is unsatisfiable. For each reachable state, the composition algorithm collects the set of assertions raised by the different calls in the same system. We are currently defining our assertion language and implementing an algorithm that checks the truth of a set of assertions.

Example : Originating Call Screening and Call Forwarding (No. 9 and 12)
Originating Call Screening (OCS) checks a dialed number against a screening list: if the

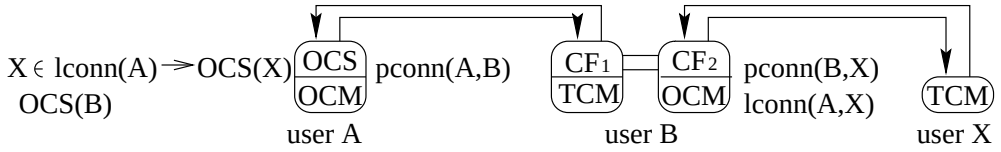


Figure 9: Assertion Violation (No. 12)

number is in the list, the connection will not be established. Call Forwarding (CF) forwards incoming calls to another number. Suppose a user A subscribes to both OCS and CF (see Figure 9). A can forward its calls to a number X on the screening list of OCS. A can then call itself, and through CF, be connected to the forbidden number X. Similarly, a user B who subscribes to CF can forward its calls to X. If A calls B, the call is forwarded to X. In both cases, OCS is thwarted, since the number dialed and tested is not on the screening list.

In the specifications, CF helps to establish physical connections between A and B (assertion pconn(A,B)) and between B and X (pconn(B,X)), and a logical connection between A and X (lconn(A,X)). OCS screens B's number and assumes that if user A is involved in a logical connection with another user, then the number of that second user has been screened

$$(A \in dom(lconn) \rightarrow \forall x(x \in lconn(A) \rightarrow OCS(x))) \wedge$$
$$(A \in ran(lconn) \rightarrow \forall x(A \in lconn(x) \rightarrow OCS(x)))$$

Assertion OCS(X) has not been raised, so the set of assertions does not evaluate to true.

4 Bellcore Benchmark of Feature Interactions

Cameron, Griffeth, Lin, Nilson, Schnure and Velthuijsen proposed a benchmark of feature interactions [6], hereafter called the Bellcore Benchmark, which is intended to be used to evaluate and compare different approaches to the feature interaction problem. These interactions are classified by their complexity (number of users, number of network components involved) and by their causes (violation of assumptions, limitations on network support and problems in distributed systems).

4.1 Specification of Benchmark Services and Features

We distinguish between a feature and a service in this paper: a service provides stand-alone operations, whereas a feature provides added functionality to a service and cannot operate by itself [1]. Our approach was directed towards the problem of specifying switch-provided customer features; that is, features that extend the functionality of POTS. Other services and features which provide Operations, Administration and Maintenance (OAM) (e.g., billing), networking, as well as ISDN features are out of our scope. Moreover, our intention was to concentrate on the functional behavior of the features. Thus, timing requirements and data values are specified abstractly. For example, there exist a number of timing requirements for Ringing Timeout, Dialing Timeout, Release Timeout, etc. events in call processing. We specify these as ordinary Internal events without explicit timing-constraint values.

We are able to specify one of the six services and 15 of the 19 features described in the Bellcore Benchmark using our specification notation. The behaviors of these features are based on the descriptions in *DMS-100 Meridian Digital Centrex Library* [13] and from our understanding of the features' behavior on our university telephone system. The features used in the benchmark and the services which they modify are listed in Table 4.

[1] For the purposes of the benchmark study, we classified CENTREX, Voice Mail, Personal Communication Services (PCS), calling from hotel rooms, and billing as services. Long distance calling is considered to be part of POTS. We classified the rest of the examples as features of various services.

Table 4: List of Features and Services Used in the Benchmark (subscripts used in Abbreviations are: c - caller, o - operator, i - invoke, r - recall, a - active, 1 - 1st leg, 2 - 2nd leg, n - newcall)

Feature	**Abbreviation**	**Service**
911	911c, 911o	POTS
Answer Call	AC	POTS
Area Number Calling	ANC	POTS
Automatic CallBack	ACBi, ACBr	POTS
Automatic ReCall	ARCa, ARCr	POTS
Call Forwarding	CF_1, CF_2	POTS
Call Number Delivery	CND	POTS
Call Number Delivery Blocking	CNDB	POTS
Call Transfer Operator feature	CT_1, CT_2	POTS
Call Waiting	CWTa, CWTn	POTS
Credit-Card Calling		POTS
Distinctive Ringing	DR	POTS
Multi-location Business Service -Extension Dialing	MBS-ED	POTS
Originating Call Screening	OCS	POTS
Terminating Call Screening	TCS	POTS
Three-Way-Calling	3WCa, 3WCn	POTS
CENTREX extension dialing		CENTREX
AIN feature billing		billing
Message Rate Charge		billing
		calling from hotel rooms
		Personal Communication Services
		Voice Mail Services

4.2 Benchmark Study

The results of our benchmark study are summarized in Table 5. The examples appear in the same order as given in the Bellcore Benchmark: examples 1-5 are interactions involving a single user and a single component; examples 6-8 involve a single user and multiple components; examples 9-10 involve multiple users and a single component; examples 11-18 involve multiple users and multiple components; and examples 19-22 are interactions between customer features and system features.

The '*Description of Interactions*' column indicates the level of specification and describes how the example is modelled using our notation, if possible. For example, in No. 1, the interaction between Answer Call (AC) and Call Waiting (CWT) is detected at the stack level, with the Terminating Call Model (TCM) at the bottom of the stack, AC in the middle, and CWT at the top of the stack; in this configuration, CWT has highest priority. In No. 11, the example is modelled at the call level. User A, with Originating Call Screening (OCS), calls user B who has Distinctive Ringing (DR). The direction of the arrow indicates the direction of the call, from caller to callee, and the dotted line separates different users. Similar notation is used to depict a call-group level configuration. In No. 9, user A, who has OCS, calls themself, and this call is forwarded by Call Forwarding (CF) to user X. In this call-group, there are two calls involved, A calls A and then A calls X.

The last four columns indicate whether we are able to specify the features used in the example and whether we are able to detect the interaction. For each example, the type of interaction is given if it can be detected, and a reason is given if it cannot be specified or detected. The abbreviations for the names of the features are given in Table 4.

4.3 Evaluation of Our Approach

Of the 15 Bellcore Benchmark interactions we can detect, six are control modification interactions (No. 1, 2a, 3, 13, 14, 15), one is a data modification interaction (No. 16), one is

Table 5: Benchmark of Interactions Detected

No.	Description of Interactions	Detect Now	Detect Later	Cannot Specify	Cannot Detect
1	*stack:* TCM/AC/CWTn	Control			
2a	*stack:* OCM/CWTa/3WCa	Control			
2b	*stack:* TCM/CWTn/3WCa	Resource			
3	*stack:* OCM/3WCa/911c	Control			
4	*stack:* TCM/ARCa/TCS	Reachability			
5	*stack:* OCM/OCS/ANC	Reachability			
6	*group:* OCM/OCS → TCM/CT1 (A); OCM/CT2 → TCM (Oper. X)		Assertion Violation		
7	Credit-Card Calling and Voice Mail Services			Out of Scope	
8	MBS-ED and CENTREX-ED			Out of Scope	
9	*group:* OCM/OCS → TCM/CF1 (A); OCM/CF2 → TCM (X)		Assertion Violation		
10	CWT and Personal Communication Services			Out of Scope	
11	*call:* OCM/OCS → TCM/DR (A, B)		Assertion Violation		
12	*group:* OCM/OCS → TCM/CF1 (A); OCM/CF2 → TCM (B, X)		Assertion Violation		
13	*call:* OCM/ACBi → TCM/CWTn (A, B)	Control			
14	*group:* OCM → TCM/CWTn (C); OCM/CWTa → TCM/CWTa (A); TCM/CWTn → OCM (B, D)	Control			
15	*group:* OCM → TCM/CWTn/3WCa (C); OCM/3WCn (A); OCM/CWTa → TCM/CWTa, TCM/CWTn (B)	Control			
16	*call:* OCM/CNDB → TCM/CND (A, B)	Data			
17	*group:* OCM → TCM/CF1 (X); TCM/CF1 (A); OCM/CF2 → TCM/CF1, OCM/CF2 (B)	Reachability			
18	*call:* OCM/ACBi → TCM/ARCa (A, B)				race condition
19	Long Distance Calling and Message Rate Charge			Out of Scope	
20	Calling from hotel rooms			Out of Scope	
21	AIN feature billing			Out of Scope	
22	AIN services and POTS			Out of Scope	

a resource contention (No. 2b), three are reachability interactions (No. 4, 5, 17), and four are assertion violations. Descriptions of these types of interaction were given in Section 3. Of the remaining eight interactions, seven are not analyzable because we cannot specify the services or features used in the example. The other interaction (No. 18) has been analyzed, but the specifications do not contain the necessary information to reveal the interaction.

Specification Notation and Model

As mentioned in Section 4.1, we developed our notation to specify customer switch-based features. Interactions No. 10, 19-22 involve features and services that our specification notation is not designed to specify. For example, in No. 19, the different segments in long distance calling is abstracted away in our model. Thus, we cannot detect interaction between segment connections and long distance billing. Example 8 involves two features that provide similar functionality but for different services: CENTREX extension dialing is provided for CENTREX service which we do not specify.

Abstraction of Timing Constraints

The advantage of specifying timing requirements abstractly is that specification and analysis is more efficient. However, if timing constraints do not specify the duration of time, then the reachability analysis might analyze unreachable paths. For example, when a callee hangs up, the underlying call models are in state ReleasePending until either the callee picks up the phone again, or until a timeout occurs. Since the reachability analysis considers all possible paths of execution, the analysis includes paths in which features perform many transitions and react to many events before the timeout event occurs. In reality, there is a limit to the number of tasks the other features can perform within this time period. Therefore, the analysis may include a number of unrealistic paths in the composition.

Example 18 describes an interaction between Automatic CallBack (ACB) and Automatic ReCall (ARC). ACB monitors the line of a busy callee and notifies the caller when the callee hangs up. ARC allows a user to call back the last caller. Suppose user A has ACB, user B has ARC, A calls B while B is busy, and A activates ACB. When B hangs up, ACB will notify A. If B activates ARC at the same time, both ACB and ARC will attempt to establish a connection between A and B. The features will determine that the parties they are trying to reach are busy, and the users have a choice as to whether or not they want to reactivate ACB and ARC. If they both re-activate, then the features may again try to establish the connection. This scenario can occur repeatedly due to the race condition that A and B are calling each other simultaneously. Since we do not specify timing constraints, the reachability analysis does not detect race conditions, and we cannot detect this kind of interaction with our model.

Lack of GoTo Construct

Although a feature has the ability to force a lower priority feature to make a specific state transition, there are limitations to this capability: the controlling feature must know the current state of the controlled feature and can only ask the controlled feature to make a transition that exists in its specification. In other words, there is no concept of a GoTo construct, whereby a controlling feature can request that the controlled feature transition to an arbitrary state in its specification. The primary reason for this decision is that a GoTo construct would violate the semantics of state-transition machines, on which our model is based.

One of the consequences of this decision is that there are features we cannot specify. Consider the credit card calling feature (example 7). A user can press # to originate another call without entering the credit card password again. In our model, the specification for credit card calling would need to force the Originating Call Model (OCM) to make a transition from *Active* back to *CollectingInfo*. This transition does not exist in the OCM specification, thus, the credit card calling service cannot force OCM to make such a transition.

Input Event Synonyms

In our analysis tools, we have incorporated a mechanism that allows us to define token events with different names but are semantically the same. Since multiple designers often introduce different terminologies for the same signal, this becomes a potential cause of interactions. This synonym mechanism helps to resolve the double-definition terminology problem.

Allowable Events

One difficulty we have encountered is that our notation does not provide for the specification of non-allowable events [3]. For example, in Terminating Call Screening (TCS), while the number is being screened, internal events of the underlying call model should not be accepted by the feature stack since the call should not proceed until the screening is done. The events $\triangleright_{TCM}$CallPresented and $\triangleright_{TCM}$TerminationDenied are out of the context of the current environment and we must explicitly disallow them in the specification. Explicitly prohibiting such inputs at each state is cumbersome. This also increases the complexity of the resulting specification. For example, the specification of Call Forwarding (CF_1) (see Table 2) requires an additional six transitions to explicitly ignore prohibited token events from the agent. It would be useful to incorporate a construct for defining the set of non-allowable events at each state into our notation. This may be addressed in the future.

Architectural Model

The advantage of specifying features independently is that they can be combined in any order for analysis. New features can be analyzed with all existing features in the system without re-specifying the whole system. Only the new features need to be specified. With our stack, call and call-group models, different aspects of call processing can be analyzed with respect to a single end-user, two connecting end-users, or multiple inter-related end-users.

State- (and Transition-) Explosion Problems

The advantage of performing reachability analysis is that every possible sequence of actions is analyzed. However, there is an explosion in the number of states and transitions in the reachability graph, versus the number of states and transitions in the individual specifications. The composition of features that have few interactions with one another will produce a large reachability graph. For example, the degree to which 3WC and CWT interact with one another depends on whether they are invoked by the same user. If user B has 3WC and user C has CWT, then the number of transitions in the composed reachability graph is enormous, since there exist many possible interleavings between B's 3WC and C's CWT: B's 3WC can be activated at any time while connected to C (independently of C's CWT), and C's CWT can switch between the two calls regardless of what state B's 3WC is in. The number of transitions in the resultant call-group composition is over 5000, whereas the number of transitions in a call-group involving only one of the features would be 200 with 3WC, and 1000 with CWT. One technique for reducing the search space would be to incorporate partial orders into our reachability analyzer [8].

5 Related Work

There are several state-transition approaches [2, 3, 4, 9, 14] in the literature that deal with the feature interaction detection problem. Each feature is associated with a set of state description primitives, and its behavior is described in terms of the effects each event has on the primitives using a set of rules. The state primitives and the rules together make up the state-based model. None of the approaches has a mechanism for inter-feature communication. Events are only generated outside of the system. Some sort of rule-application mechanism is used when non-deterministic situations occur during composition. This way, behaviors of

other features may be changed. The main difference between these approaches and ours is that we have a communication protocol between features and between end-users. Signals are passed from the environment to the system, between different parties within the system, and from the system to the environment. This enables us to detect data modification interactions and resource contention interactions, whereas these approaches cannot. The type of interactions they detect are deadlock, consistency and problems that arise from the rule-application mechanism. We suspect that these correspond to reachability interactions and some of the control modification interactions in our model. Kawarasaki and Ohta [9] also propose manual verification methods for dealing with interactions due to timing, number of retries and the use of different words for events and primitives that are semantically the same. We can detect the latter two types of interactions using the same mechanism we have for tracking resource usage, and using input-event synonyms, respectively. We may be able to incorporate their manual inspection methods for detecting timing interactions in our specifications.

The LOTOS specification model consists of feature specifications and feature intentions [7, 17]. Each system of features and each intention verifier is viewed as a process in process algebra and they can be joined together in parallel composition. Feature intentions are similar to our assertions: they describe the asserted assumptions and properties that the features expect to hold. An interaction is detected when a feature intention is violated in the system. Their method also uses reachability analysis to generate execution sequences to check for violated intentions. However, due to the state explosion problem, search strategies must be given to the tools. Interactions are detected by manual inspection of the traces produced from the tools. We also experience a state explosion problem, but our automated detection method is feasible for the size of the call-group configurations we have needed to analyze so far.

The Building Block approach [12] is very similar to ours. Both adopt the ideas of specifying features independently and using individual features as building blocks to build different call scenarios. In their building block approach, procedural-level specifications describe how a feature should operate (similar to our State Transition Machine specifications), and behavioral-level specifications describe what properties a feature must exhibit using temporal logic formulas (similar to our assertions). Model checking is used both to verify desired behavior in the procedural specifications and to check that temporal logic properties still hold after composition. The types of interactions detected include reachability, control modification, and assertion violation interactions.

Researchers at BT Laboratories use SDL to model features and detect interactions [10, 11]. A *service plane model* provides a user's view of the system, and is used to validate feature specifications and identify interactions. A *Centrex network model* is a composition of service and feature specifications, similar to our call-group, that is used to inspect the concurrent activation of features by multiple users (though only one user can activate more than one feature). Identification of interactions appears to be manual inspection of simulated executions of the models. Although simulation is not as thorough as reachability analysis, they have successfully identified previously unknown interactions using their technique [10].

6 Conclusion

Our approach has been geared to the specification and analysis of the functional behaviors of customer, switch-based features. We have used the Bellcore Benchmark [6] as a means for evaluating our approach. We are able to specify all of the customer, switch-based features discussed in the Bellcore Benchmark with our notation; and we are able to detect 11 of the 23 benchmark interactions, with the hope of detecting four more once we finish the implementation of the assertion violation analysis.

Given that we initially chose to concentrate on detecting interactions in switch-based features, and given the variety and complexity of the interactions in the Bellcore Benchmark,

we were pleasantly surprised that we were able to detect as many of the interactions as we have. However, it is not clear that we could detect many of the remaining interactions even if we were to specify some of the other basic services used in the benchmark examples. Our approach is limited due to the inability to specify and analyze data values, timing constraints, and assumptions about the behavior of the environment. In addition, we are concerned about the degree to which our requirements specifications contain design decisions. We would like to turn our attention towards assertion based approaches.

References

[1] P. Au. An evaluation of a state-based model of feature interactions. Master's thesis, Department of Computer Science, University of Waterloo, 1997.

[2] J. Blom, B. Jonsson, and L. Kempe. Using Temporal Logic for Modular Specification of Telephone Services. In *Feature Interactions in Telecommunications Systems*, pages 197–216, 1994.

[3] J. Blom, R. Bol, and L. Kempe. Automatic Detection of Feature Interactions in Temporal Logic. In *Feature Interactions in Telecommunications Systems*, pages 1–19, 1995.

[4] M. Boström and M. Engstedt. Feature Interaction Detection and Resolution in the Delphi framework. In *Feature Interactions in Telecommunications Systems*, pages 157–172, 1995.

[5] K. Braithwaite and J. Atlee. Towards Automated Detection of Feature Interactions. In *Feature Interactions in Telecommunications Systems*, pages 36–59, 1994.

[6] E. J. Cameron, N. D. Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen. A Feature Interaction Benchmark for IN and Beyond. In *Feature Interactions in Telecommunications Systems*, pages 1–23, 1994.

[7] M. Faci and L. Logrippo. Specifying Features and Analysing Their Interactions in a LOTOS Environment. In *Feature Interactions in Telecommunications Systems*, pages 136–151, 1994.

[8] P. Godefroid. Using Partial Orders to Improve Automatic Verification Methods. In *CAV*, 1990.

[9] Y. Kawarasaki and T. Ohta. A New Proposal for Feature Interaction Detection and Elimination. In *Feature Interactions in Telecommunications Systems*, pages 127–139, 1995.

[10] B. Kelly, M. Crowther, and J. King. Feature Interaction Detection using SDL Models. In *IEEE Globecom*, 1994.

[11] B. Kelly, M. Crowther, J. King, R. Masson, and J. DeLapeyre. Service Validation and Testing. In *Feature Interactions in Telecommunications Systems*, pages 173–184, 1995.

[12] F. J. Lin and Y.-J. Lin. A Building Block Approach to Detecting and Resolving Feature Interactions. In *Feature Interactions in Telecommunications Systems*, pages 86–119, 1994.

[13] Northern Telecom. *DMS-100 Meridian Digital Centrex Library*, 50006.08/12-92 issue 4 edition, 1992.

[14] T. Ohta and Y. Harada. Classification, Detection and Resolution of Service Interactions in Telecommunication Services. In *Feature Interactions in Telecommunications Systems*, pages 60–72, 1994.

[15] K. Pomakis. Reachability analysis of feature interactions in service-oriented software systems. Master's thesis, Department of Computer Science, University of Waterloo, 1995.

[16] K. Pomakis and J. Atlee. Reachability analysis of feature interactions: A progress report. In *ISSTA*, 1996.

[17] B. Stepien and L. Logrippo. Representing and Verifying Intentions in Telephony Features Using Abstract Data Types. In *Feature Interactions in Telecommunications Systems*, pages 141–155, 1995.