Daniel Amyot High Level Design and Prototyping of Agent Systems Department of Systems and Computer Engineering, Carleton University email: damyot@csi.uottawa.ca

Abstract. Functional scenarios describing system views, uses, or services are a common way of capturing requirements of telecommunication systems. However, integrating individual descriptions of telephony features in different ways may result in different kinds of unexpected or undesirable interactions. Appropriate integration techniques can hopefully lead to fewer such interactions. In this report, we first present how a collection of features integrated visually through causal scenarios called Use Case Maps (UCMs) may help generating high-level LOTOS specifications. Integrating UCMs together helps avoiding trivial and artificial interactions before any prototype is generated. Then, we use the powerful testing concepts and tools of LOTOS to detect remaining undesirable interactions. To illustrate these concepts, we capture and validate a subset of the telephony features from the First Feature Interaction Contest. We discuss the results of this experiment, as well as strengths and weaknesses of our methodology.

*Key words*. *Causal scenario, feature interaction, integration, LOTOS, specification-level validation, testing, Use Case Maps.* 

# 1 INTRODUCTION

A *feature* is a collection of services packaged together that can be commercialized. Undesirable interactions between features still represent nowadays a complex problem that telecommunication systems designers must face [18][31], and this situation is likely to remain challenging in the future. By definition, features interact with each other and with the basic system services, the so-called *Plain Old Telephone System* (POTS). However, a feature might be prevented from working properly according to its intent because of some unexpected interactions with other features in the system. This is at the heart of the feature interaction (FI) problem. Similar challenges can be found in the agent community where agent goals might be conflicting and impossible to fulfil simultaneously [16][24]. For the last decade, many partial solutions have been suggested to avoid, detect, analyze, and solve feature interactions at design time and run time. Our proposal is one of avoidance at design time, and one of detection at design time with the help of an executable prototype. Avoidance of trivial interactions is achieved through the visual integration of scenarios expressed with the *Use Case Map* (UCM) notation [10][17]. Detection is done by using a process algebra, the *Language Of Temporal Ordering Specification* (LOTOS) [26] in our case, and formal V&V techniques.

LOTOS has been used for years for the specification and validation of telephony systems ([7][19][20][23]) and for the detection of interactions between telephony features ([21][22] [31][32][38][39][40][41]). Research is still ongoing as to its application to real-size problems. Use cases were utilized for the analysis of interactions in [33]. More recently, UCMs have also been used to tackle the problems of feature interactions and resolution of conflicts in multi-agent systems ([11][12][13][14][15][16]). The UCM notation helps designers with the visualization of

problematic situations and their avoidance at a high level of abstraction. An approach where UCMs are transformed into LOTOS specifications and test cases has been applied to a number of examples in the areas of distributed systems and telephony ([2][3][4][5][6]).

With such knowledge and experience available, a methodology that would make use of the best features of UCMs (e.g., visual description and integration of features) and LOTOS (e.g., powerful theory and tools for validation and verification) for the avoidance and detection of feature interactions in telephony systems seems a natural evolution. Herein, we use such an approach (Section 2), and we illustrate it using some of the features described in the first feature interaction contest [25]. We present UCMs for selected features in Section 3. These UCMs were captured and integrated by Petriu in [35]. We discuss the synthesis and the validation of the LOTOS specification in the following section (Section 4). When integrating UCM scenarios (features) together, some trivial interactions can be avoided. However, for the remaining undesirable interactions, we use traditional LOTOS techniques and tools (Section 5). We discuss this methodology with three other approaches in Section 6 and then provide general conclusions.

### 2 METHODOLOGY

### 2.1 Rigorous Approach Based on Scenarios

We believe that the usage of UCMs in a scenario-based approach represents a judicious choice for the description and the design of reactive and distributed systems. Scenarios fit well in approaches that intend to bridge the gap between (informal) requirements and the first system design.



Figure 1 Scenario-Based Approach Used in this Experiment

Figure 1 introduces a scenario-based approach for designing telephony systems that are free of undesirable feature interactions. It is adapted from a more generic and rigorous approach discussed in [4][6]. We observed several advantages to this rigorous approach, the most important being related to the separation of the functionalities from the underlying structure, fast prototyping, test cases generation, and documentation of the requirements and of the high-level design.

In our case study, the start point is a collection of individual features described as Chisel diagrams [1]. Each feature is then captured as a Use Case Map (1). In the literature, this phase is often referred to as *scenario elicitation*, although in our case the requirements were already in the form of operational scenarios. The responsibilities in the UCMs are bound to components in the

underlying structure, which is common for all scenarios in this specific example. UCMs can then be integrated together to produce a global UCM that covers all cases (2). Sequential, alternative, and parallel composition<sup>1</sup> are used as integration operators, as well as more subtle abstraction and composition mechanisms that make use of stubs and plug-ins. Once the global UCM is available, it can be used to synthesize a LOTOS specification, which becomes the executable prototype (3).

Concurrently with these steps, validation test cases can be generated from the Chisel diagrams (4) to ensure that the specification conforms to POTS and to each individual feature, when only one is active at a time. We can create further test cases, built on top of the test cases for individual features, in order to detect undesirable interactions between pairs of features integrated according to the global UCM (5). All test cases are described in the same language as the specification, i.e., LOTOS.

Probes can be inserted in the specification in order to measure how much of the structure of the specification is covered by the test suite and to ensure that the whole specification has been exercised by at least one test case (6). The new specification then contains the probes, to which we add the test cases for individual features and those for pairs of features (7).

Once the specification has been tested against all the test cases (8), results and statistics (9) can be obtained from the resulting trees (*Labeled Transition Systems* — LTSs). One of the following verdicts will occur:

- At least one test case from the individual feature set has failed. Since it does not work properly on its own, the specification of this feature has been incorrectly synthesized from the global UCM, or this UCM does not conform to the Chisel diagram. In the latter case, the capture or the integration of this scenario might be the cause.
- At least one test case from the feature interaction set has failed. The specification of the two features involved is incorrect w.r.t. their integration in the global UCM, or there is a *feature interaction*, i.e., an unforeseen and undesirable result.
- At least one probe has not been visited by the entire test suite. Some part of the specification is unreachable, or the test suite is incomplete and does not cover a case that the specification considers, or the specification covers a case that should not be considered.
- The test suite has passed successfully, and all probes have been covered. The specification conforms to the requirements (Chisel diagrams), and no feature interaction was detected. We then have a good level of confidence in the global UCM, in the LOTOS specification, and in the test suite.

Following the verdict, modifications may be required to the UCMs, to the test cases, and/or to the specification. In fact, the approach of Figure 1 is iterative. It is also incremental as new features may be integrated at a later time.

<sup>1.</sup> Composition is a much overloaded term. In this report, we use *integration* when we refer to the process of merging several UCM scenarios, while we use *composition* to represent the different constructs used in such integration. Composition refers also to the way plugins are linked together in a stub, and to the way LOTOS concurrent processes interact with each other.

# 3 USE CASE MAPS FOR FEATURES

# 3.1 Use Case Maps in a Nutshell

UCMs are a visual notation we utilize for capturing the requirements of reactive and distributed systems. They describe scenarios in terms of *causal relationships* between *responsibilities*. UCMs put emphasis on the most relevant, interesting, and critical functionalities of the system. They can have internal activities as well as external ones. Usually, UCMs are abstract (generic), and could include multiple traces (called *routes*). With UCMs, scenarios are expressed above the level of messages exchanged between components, hence they are not necessarily bound to a specific underlying structure. They provide a path-centric view of system functionalities and improve the level of reusability of scenarios.

Figure 2 shows a simple UCM where a user  $(U_1)$  tries to establish a connection with another user  $(U_2)$  through some network.  $U_1$  first sends a connection request (**R**) to the network. The latter verifies (**V**) whether or not the called party is free. If she is, then there will be some status update (**F**) and a ring signal (**S**) will be activated on  $U_2$ 's side. Otherwise, the network status will be updated differently (**O**) and a message stating that  $U_2$  is not available (**M**) will be sent back to  $U_1$ .



Figure 2 Simple Connection UCM

A scenario starts with a triggering event or a precondition (filled circle labeled R) and ends with one or more resulting events or postconditions (bars), in our case S and M. Intermediate responsibilities (V, F, O) have been activated along the way. In this picture, the activities are allocated to abstract components ( $U_1, U_2, Network$ ). The notation allows for alternative paths (the fork in the figure), concurrent paths, and for explicit synchronous/asynchronous interactions between paths. For a detailed description of the notation, refer to [17].

The construction of a UCM can be done in many ways. Usually, one starts by identifying the activities that are to be performed by the system. They can then be allocated to scenarios and/or to components. Components can be discovered along the way. Eventually, the two views are merged to form a *bound UCM*, like the one in Figure 2.

# 3.2 Overview of the FI Contest Content

In the FI contest description (see [25]), a network was modeled as a collection of black boxes communicating with each other via defined interfaces. Definitions for the POTS service and the twelve features were given as sequences of (synchronous) events taking place on these interfaces. Interactions were to be detected between pairs of features.

### Network Structure

The left half of Figure 3 shows that the network consists of end-user equipment (telephones A, B, and C), a switch, a Service Control Point (SCP) that processes IN features [28], an Operations System (OS) that does billing, and a global clock (not on the figure). The network interfaces are

the interface between a user and the switch (on which the telephone is used for signaling); the interface between the switch and the SCP (on which IN messages are used); and the interface to the billing system (for tracking the beginning and end of each call).



Figure 3 The Network and the UCM Structure

This network was transformed in an abstract structure (right half of Figure 3) on which UCMs that capture the Chisel diagrams are to be drawn. The switch, the SCP, and the OS were mapped onto abstract components. The phones were split into two sets of (replicated) components based on the user's role in a call, i.e., Originator or Terminator. The interfaces were left out as they are usually part of a more refined level of abstraction than the one addressed by UCM structures.

### Features

On top of POTS, the first phase of the contest described ten features, but this report mainly focuses on four of them:

- *Calling Number Delivery* (CND): allows the called telephone to receive a calling party's Directory Number and the date and time. The number is delivered whenever an idle called party receives the Ringing event.
- *IN Freephone Billing* (INFB): allows the subscriber to pay for incoming calls. Call routing, although normally part of this feature, has been dissociated into another feature.
- *IN Teen Line* (INTL): restricts outgoing calls based on the time of day (i.e., hours when homework should be the primary activity). This can be overridden on a per-call basis by anyone with the proper identity code.
- *Terminating Call Screening* (TCS): restricts incoming calls. Calls from lines that appear on a screening list are redirected to a vague but polite message.

The six remaining features were IN Freephone Routing (INFR), Call Forwarding Busy Line (CFBL), Three-way Calling (3WC), IN Call Forwarding (INCF), Call Waiting (CW), and Charge Call (CC). The second phase contained two additional features, namely Cellular pays (Cell) and Return Call (RC). The UCMs developed in this phase considered a third additional feature as well, namely Automatic Call Back (ACB) [35].

# **3.3 UCM Capture from Chisel Diagrams**

This section discusses the step (1) in Figure 1. Chisel diagrams are used to define requirements for communications services and service features. Since its design originated at a usability workshop involving practitioners, the language Chisel is intended to reflect current practice for writing these requirements [1]. The authors of this language claim that it is unambiguous, that it applies to a variety of network technologies, and that it has a sound basis for translation to commonly used

formal software specification languages. The purpose of Chisel diagrams is to improve communication between the diverse people and organizations involved in the telecommunications service creation process.

The Chisel diagram for INTL is given in Figure 4. Sequences and alternatives of events on the network interfaces are supplemented by variables and conditions. Each node in the tree has a unique identifier, an events name, and a list of parameters. Nodes are also allowed to have multiple events, separated by |||, that can occur in any order. Some leaves are followed by references to a specific node in the POTS Chisel diagram, and variables in that diagram are assigned values from the INTL diagram. We will not dwell further in the explanation of Chisel diagrams, nor will discuss the meaning and the correctness of the INTL feature in Figure 4.

Usually, requirements do not come in such formal and operational form. In our case, since these Chisel diagrams are at a somewhat lower level of abstraction than UCMs, the scenarios to be captured will be more abstract. This is generally not the case because requirements are often described in prose form, i.e., in an informal and non-operational form.



Figure 4 Chisel Diagram for IN Teen Line (INTL)

The Chisel diagrams are based on events that are shared between entities (the network components), whereas UCMs are described in terms of responsibilities performed by components. This first issue has been resolved by assigning these events to the component in which they will most likely be observed. Hence, events that are unobservable by the user become local to the system components (Switch, SCP, OS). Figure 5 shows a partial UCM for the INTL feature of Figure 4. Some events become responsibilities local to the switch (like setting the busy status of the originator), others become responsibilities that the user can observe (like getting an announcement "Ask for PIN"), and others remain events that the user can trigger (like off-hook). Responsibilities are marked with a cross, and event names are associated to start and end points. They are bound to their respective network components (see Figure 3). Obviously, some responsibilities will be refined as events or messages between components at a lower level of abstraction (like Chisel diagrams, Message Sequence Charts (MSCs) [29], ObjecTime design, or LOTOS specifications), but UCMs delay this kind of decision to a next refinement stage, possibly with another and more appropriate notation.

Resources and responses (on the SCP-Switch interface) were not put on the UCM because they are basically messages and they are hidden from the user's point of view (from which we describe the scenarios). However, their existence is somewhat implied by the path crossing the SCP-Switch boundary on several occasions. Resource and response messages represent only one way to implement the causal relationship shown in the UCM and the checking of the conditions in the SCP. This is in fact the refinement chosen by the producers of the original Chisel diagrams, which are more detailed (and hence less loose) than UCMs. UCMs provide a description similar to a *service specification* in the OSI model, where we can specify abstract actions and components that are not always visible to the user, but without committing too soon to an implement-oriented solution.

Note also that the conditions are simplified to the point where they become simple (italicized) labels on the paths. The conditions themselves should be expressed with another notation, more suitable for dealing with data.



Figure 5 Partial UCM for INTL

This UCM is incomplete and focuses on the behaviour specific to INTL in the context of POTS. It then continues just like the POTS UCM would (although it is not shown in this report as an individual UCM). The INTL feature, as defined in the contest description, refers to POTS for common behaviour. This also means that disconnections need to be managed by our UCMs. One of the assumptions in the contest was that a hang-up could occur only at some specific points in the scenarios. These occur where end points (bars) are inserted in the UCM. Therefore, a disconnection could happen instead of dial PIN or dial, or after reject. Hence, a disconnection UCM, not shown here, is implicitly composed with the INTL UCM at each of these locations on the map. Its triggering would prevent the other events to occur and terminate the call connection(s).

# **3.4 Integration of UCM Scenarios**

Individual scenarios are useful for understanding the behaviour of one feature, but they can also be integrated together to form a *global UCM* (step (2) in Figure 1). The assumption here is that performing the integration at this level of abstraction provides early insights in possible conflicts between features expressed as scenarios. Integration helps to ensure early consistency between individual maps. For instance, events and responsibilities that are not labeled correctly, that are omitted, or that are not at the same level of abstraction or in the same order become hard to integrate. Hence, they indicate that some individual maps might need to be fixed. Integration also helps to avoid ambiguous situations, the most common of which is non-determinism. A path segment that is a prefix to two different scenarios might imply the need for a way to decide which alternative to take in a global scenario. Merging several path segments together might also indicate that variables and data are required to distinguish between the different cases, similarly to multiplexers in circuit design. Many such design decisions can be made at this level.

### Root Map

The following root map and plugin maps result from the integration of the thirteen features enumerated in Section 3.2. This integration was done with the *UCM Navigator* tool [34], a UCM editor developed in our research group, which outputs the next few figures. The *root map* (Figure 6) represents the global context in which sub-maps are plugged in. The diamonds in this UCM are called *stubs* and they serve as placeholders for *plugin maps*. The diamonds with filled lines (e.g., **post-dial**) are *static* stubs and they contain only one plugin map. They are basically used as an abstraction mechanism and for path refinement. The diamonds with dashed lines (e.g., **pre-dial**) are *dynamic* stubs and they may contain several plugin maps from which one or more are selected at run-time depending on the satisfyability of their associated preconditions. Plugins are maps that can also contain their own stubs.



Figure 6 Root Map for Global UCM

### Binding of Plugins to Stubs

One constructs a complete scenario by recursively selecting appropriate plugins for the stubs. Many figures in this section present plugins created for the FI contest. They are bound to the stub by associating the entry and exit points of the stub to the start and end points of the plugin map. The first stub in the root map, **pre-dial**, one entry point (*IN1*), and two exit points (*OUT1*, *OUT2*), as shown in Figure 7.



Figure 7 Entry and Exit Points on a Stub (pre-dial)

Each stub has its set of entry and exit points that may be bound to plugins. For instance, the default plugin for pre-dial is basically a straight path, whose start point is POTS and end point is dial, and which does nothing but connect *IN1* to *OUT1*. Hence, the binding is {(POTS, *IN1*), (dial, *OUT1*)}. *OUT2* remains unbound, and therefore this path (leading to reject in the root map) will never be followed when the default plugin is selected. This same stub has a second (and much more complex) plugin, illustrated in Figure 8.



**Figure 8** INTL Plugin for Pre-dial Stub in the Root Map<sup>1</sup>

Its binding is {(INTL, *IN1*), (dial, *OUT1*), (rejected, *OUT2*)}. With this plugin, it is possible to reach the second exit point that leads to a reject end point (itself leading to an eventual disconnection due to the implicit composition at each end point in the root map, as discussed in Section 3.3).

The INTL plugin of Figure 8 differs in other ways from the default plugin for pre-dial. Their preconditions are mutually exclusive, i.e., the user must be subscribed to INTL for this plugin to be selectable, and the user must not be subscribed to INTL for the default plugin to be selectable. Hence, the two plugins can never be active simultaneously. This alternate composition within the stub results from the nature of the individual features and from how they were integrated together. In essence, INTL is the only feature that deviates from all the others between the update of the busy status (busyOrig) and the dial tone (DT).

When a user is subscribed to INTL only, the flattening of the root map with the INTL plugin in the **pre-dial** stub and default plugins in the other stubs results in the individual UCM of Figure 5.

#### Other Relevant Plugins

To obtain a complete picture of the system with the four features that interest us (CND, INTL, INFB, and TCS), we now give an overview of the remaining appropriate plugins. Bindings will not be discussed unless they are not obvious from the figure.

<sup>1.</sup> The empty circles on the paths are *empty points* and are used for path transformations in the UCM Navigator. They are not part of the UCM notation as such.

The **post-dial** static stub in the root map contains by definition only one plugin, which is shown in Figure 9 (where **R** means Ringing, and **RR** stands for the remote AudibleRinging). In this UCM, several path segments are concurrent, as explicitly stated by the ||| operator in the Chisel diagrams. Some slight differences were introduced at this point due to the distributed nature of our system that cannot be so easily abstracted from with paths that cross components. For instance, an **RR** could occur at the originator after the terminator has picked up the phone, thus representing the fact that the system might take time to consume the **off-hook** event before deciding to stop the **R** and **RR** activities. This behaviour, which might reflect the real system better, contains the behaviour described in the Chisel diagrams, but also allows for other global scenarios.



Figure 9 Plugin for Post-dial Static Stub in the Root Map

The default plugin for the **process-call** stub of Figure 9 is illustrated in Figure 10 (a). This is the point where the system checks whether or not the terminator side is busy. If so, the **busy** path is selected. Otherwise, the **idle** path is selected and the terminator status is set to busy (**busyTerm**). The binding is {(**POTS**, *IN1*), (**idle**, *OUT1*), (**busy**, *OUT3*)}.





Figure 10 Default Plugin for Process-call Stub in Figure 9

The INFB plugin (Figure 11) does not override the default one, but occurs before. It simply analyzes some IN information in the SCP and then sets the called party as the paying party. The binding is {(INFB, IN1), (callB, OUT4)}.



Figure 11 INFB Plugin for Process-call Stub in Figure 9

The TCS plugin of Figure 12 is similar in nature to the default one, except that it first checks whether or not the originator party is on the screening list. If so, then the call is rejected. It overrides the default plugin when the terminator party has subscribed to TCS. The binding in this case is {(TCS, *IN1*), (idle, *OUT1*), (TCS-reject, *OUT2*), (busy, *OUT3*)}.



Figure 12 TCS Plugin for Process-call Stub in Figure 9

The **busy** stub in Figure 9 has one plugin that concerns us, the other being related to the features not discussed in this report. The default plugin in this case simply connects the entry point to the path leading to the **busy** event. The last stub in this figure, **display**, also has a straightforward default plugin that does nothing but connecting the entry point to the exit point. When the terminator side has CND active, the plugin shown in Figure 13 is used instead of the default one in order for the originator's number/name to be displayed.



Figure 13 CND Plugin for Display Stub in Figure 9

Finally, the billing stub in the root map (Figure 6) contains two mutually exclusive plugins selected according to whether or not the terminator has subscribed to INFB. If so, then the terminator party (also referred as B) is charged with the incoming call, otherwise the originator (A) pays.



(a) Default Plugin

(b) INFB Plugin

Figure 14 Default and INFB Plugins for Billing Stub in Root Map

### 3.5 Avoiding Feature Interactions

We claim that an integration of scenarios at the level of UCMs helps to avoid some trivial or artificial interactions between features. For instance, many potential interactions between INTL, INFB (or TCS), and CND are avoided because the features in each possible pairwise combination are allowed to proceed independently in the map. They are integrated using a sequence of three different stubs that encapsulate the features from their environment.

Important design decisions still need to be made at integration and composition time, something that cannot be easily automated. For example, interactions between features in one stub (e.g., INFB and TCS) are still possible, depending on the composition/decision mechanism used within the stub (**process-call** in our case). Maps with stubs show how localized the impact of a feature can be. They can be represented by only one plugin (INTL in **pre-dial**), or by several plugins along one or many paths (INFB in **process-call** and **billing**). This helps focusing on issues related to how a plugin (i.e., dynamic behaviour) is selected in one or more dynamic stubs. Since only a limited number of smaller UCMs have to be considered in a stub, it becomes easier to check that they have mutually exclusive but complete preconditions (to avoid non-determinism and unspecified behaviour), or that priorities need to be established. Hence, the design decisions are simpler. The integration becomes an interesting and useful step in a design process that includes UCMs, and it cannot be as trivial as the composition of states suggested by the Chisel approach. However, the composition of plugins in a stub should not be done at the UCM level, which is not an adequate notation for such details. A more appropriate notation, such as LOTOS or some agent meta-models, should be used instead.

Chisel diagrams specify normal behaviour, but they do not distinguish between what should be obliged and what should be permitted or even forbidden in a feature. For instance, in their respective Chisel diagrams, CND displays the incoming call and charges the call to the originator, while INFB does not display and charges the call to the terminator. Although this appears to be an interaction, it is somewhat artificial since these two features are obviously compatible in telephony systems. That is because CND *obliges* the display and *allows* for the terminator to pay (it is not forbidden), whereas INFB *allows* the display (it is not forbidden) and *obliges* the terminator to pay. Stated like this, these requirements, which are acted upon only at integration time, lead to a global system without such interactions between CND and INFB. This kind of information (modalities on the alternatives) would help to determine what stubs are required and how the default behaviour (POTS) is overridden. In our example, we had to infer this knowledge manually from our understanding of the *intent* of these features. A notation like the OPI model (Obligation-Permission-Interdiction) would make this distinction explicit in the description of a feature [7]. Supplemented with OPI concepts, UCMs could be used to better capture the intent of features in terms of scenarios, and not in terms of properties as it is usually the case.

### 4 LOTOS SPECIFICATION

### 4.1 LOTOS and the Synthesis & Validation Approach in a Nutshell

#### **Overview** of LOTOS

For the last decade, we observed that formal methods, such as LOTOS, SDL, MSCs, and Estelle, have proven their usefulness in capturing descriptions of complex, concurrent, and distributed systems. LOTOS is an algebraic specification language standardized by ISO [26]. Using LOTOS, the specifier describes a system by defining the temporal relations along the actions that constitute the system's externally observable behavior. Data abstractions can also be described by using *Abstract Data Types* (ADTs).

LOTOS is powerful at describing and prototyping distributed systems at many levels of abstraction through the use of *processes*, *hiding*, *parallel composition* and *multiway synchronization*. LOTOS is suitable for the integration of behavior and structure in a unique executable model. LOTOS models allow the use of many validation and verification techniques such as step-by-step execution (simulation), random walks, testing, expansion, model checking, and goal-oriented execution. Many tools can be utilized for the automation of these techniques, and several development cycles based on stepwise refinement are available.

#### Synthesis of Specifications from UCMs

The synthesis of LOTOS specifications, illustrated by our example scenario in Figure 15, allows for the rapid generation of prototypes that implement UCM scenarios. The behaviour of each component is translated into a LOTOS process that preserves the internal causality relationships between the responsibilities and events that are part of path segments crossing this component (right half of Figure 15). The architecture itself is converted to a structure (left half of Figure 15) where the processes are composed together through shared communication *channels*<sup>1</sup> (LOTOS gates). The causal relationships between the components are also considered during the construction of the processes. Decisions related to the nature of the message exchanges must then be made and documented.

<sup>1.</sup> We use the generic term *channel* to denote a communication link between two entities, not necessarily a SDL channel (a FIFO queue).



Figure 15 Synthesis of a LOTOS Specification from a UCM

# LOTOS Validation

Since the synthesis is not automated, it becomes necessary to validate the specification against the UCMs, which correspond to the (informal) requirements. Four of the most common approaches to the validation of a LOTOS specification are simulation, equivalence checking, model checking, and functionality-based testing.

Simulation is the step-by-step execution of a specification. The designer takes the role of the environment, provides events to the specification, and observes the results (the next events). Although useful for debugging, simulation is probably the weakest validation technique available.

Equivalence checking usually requires a formal representation of (part of) the requirements, seldom available in the early stages of the design process. However, this approach is most useful when checking the conformity of one specification against another, after some refinement or modifications.

Model checking aims to validate a specification against safety, liveness, or responsiveness properties derived from the requirements. These properties can be expressed, for instance, in terms of temporal logic or  $\mu$ -calculus formulas. In the LOTOS world, this technique usually requires that the specification be expanded into a corresponding model, which is some graph representation (labeled transition system, finite state machine, or Kripke structure) of the specification's semantics. On-the-fly model checking techniques, where the whole model does not have to be generated a priori, exist as well. Often, the languages used to define properties are very flexible and powerful, yet they can be quite complex; it is a difficult problem to determine whether a property really reflects the intents of informal requirements.

Functionality-based testing is concerned with the existence (or the absence) of traces, use cases, or scenarios in the specification. These scenarios reflect system functionalities, usually in terms of operational or user-centered instances of intended system behaviour. They can easily be transformed into black-box test cases that can be composed with the specification for validating the latter against requirements. Test cases are often more manageable and understandable than

properties, and they relate more closely to informal requirements. However, they are usually less powerful and expressive than liveness or safety properties expressed in temporal logic. For example, a test suite that passes successfully does not prove the absence of errors in any way.

Among these four approaches, we favored functionality-based testing for the validation of the features and the detection of interactions. Simulation is not sufficient because there are just too many global sequences of events possible in the system. Equivalence checking is not possible because we aim to produce a first high-level specification from the scenarios. Since these requirements are expressed mostly operationally, UCMs and test cases are easier to extract than properties, so model checking should not be used at first. It could be used later on, however the state explosion problem can hardly be avoided in our case.

### LOTOS Testing from UCMs using LOLA

LOLA (LOtos LAboratory) is a state exploration tool with application in simulation, testing, and transformation of LOTOS specifications [37]. It has the particularities of accepting Full LOTOS and of being available on several platforms (including SunOS, Linux and DOS). Its testing strategy is consistent with the *Testing Equivalence*. The LOTOS testing theory has a test assumption stating that the implementation (the specification in our case) communicates in a symmetric and synchronous way with external observers, the test processes. There is no notion of initiative of actions, and no direction can be associated to a communication.

In the following, we assume that *Success* is a special gate, not part of the specification under test, which is used in the test cases to indicate a successful execution. LOLA expands the composition of the specification and a test process in order to analyze whether the executions reach the success event or not. Three *verdicts* can occur after the execution of one test case:

- **Must pass**: all the possible executions (called *test runs*) were successful (they reached the *Success* event).
- **May pass**: some executions were successful, some unsuccessful (or inconclusive according to a depth limit).
- **Reject**: all executions failed to reach *Success* (they deadlocked or were inconclusive).

In the real world, test cases must be executed more than once when there is non-determinism in either the test or the implementation (under some fairness assumption). However, LOLA avoids this problem because it determines the response of a specification to a test by a *complete* state exploration of the following composition [36]:

SpecUnderTest[EventsSpec]
|[EventsSpec \cup EventsTest]|
Test[EventsTest \cup {Success}]

LOLA analyzes all the test terminations for *all possible evolutions* (test runs). The successful termination of a test run consists in reaching a state where the termination event (*Success*) is offered. A test run does not terminate if a deadlock or internal livelock is reached.

Validation test cases are usually derived from the UCMs in order to detect errors, incompleteness and inconsistencies. For most distributed systems, including telephony systems, the high (if finite) number of global states makes the generation of an exhaustive test suite impossible. Hence, it becomes essential to carefully select a small and finite set of validation test cases. To do so, we can base our strategy is based on the exploration of UCM paths, similarly to white-box approaches used for sequential programs. Depending on the targeted coverage, the critical nature of paths, and the cost associated to their traversal, we can choose to explore some paths, all combination of paths, some or all the temporal sequences resulting from concurrent paths, etc. For each selected abstract sequence of events/responsibilities (UCM routes), *acceptance* test cases (whose expected verdict is **Must pass**) and *rejection* test cases (whose expected verdict is **Reject**) can be generated.



Figure 16 Derivation of Validation Test Cases from UCMs

Our sample scenario is reused again in Figure 16, to demonstrate the derivation of a set of test cases with the goal of covering all paths in the UCM. Each path linking a start point to an end point then becomes an abstract sequence that will be translated into a LOTOS test process (while considering the observable messages and data types defined during the synthesis). In this example, the rejection test cases were generated from the abstract sequence where a mutation was applied on the last event (a fault model called *off-by-one*).

Although this general test derivation approach could be used for validating our features, we chose instead to use a more detailed model that was available to us, namely the Chisel diagrams. These diagrams are described at a somewhat lower level of abstraction than UCMs, and therefore they bring more precision to the definition of the tests for individual features. In general, when one starts from informal requirements, such detailed description is not yet available. Hence, validation test cases are usually derived from UCMs, not from the requirements (contrarily to transition (4) in Figure 1). However, since the Chisel diagrams were given to us in the contest description, their use seemed appropriate (see Section 4.3). Moreover, we limited our scope to acceptance tests only (with a **Must pass** verdict expected). Rejection test cases are left for future work.

# 4.2 Synthesis

The current section relates to step (3) in Figure 1. Following the synthesis approach introduced in the previous section, we are now about to generate a LOTOS specification from the global UCM (Figure 6) and its plugins. This specification, presented in Appendix A, will serve as the basis for the validation of individual features against their requirements and for the detection of interactions.

This section provides general explanations about the synthesis of our LOTOS specification. We first discuss the data types needed to support the parameters, databases, and preconditions. We follow with the representation of the network (Figure 3) as a structure of LOTOS components.

Finally, for some components of the network, we present the construction of the processes' internal behaviour from the UCM paths that have responsibilities bound to these components.

### Data Types

The abstract data types are mostly derived from the tabular descriptions in the contest description [25], except for the basic data types and operations (Boolean, NaturalNumber, FBoolean, Element, and Set), which are ADTs simpler than the ones in the International Standard (lines 80 to 227). They were simplified in order to become more efficient in our tools. The ADTs specific to the features are as follow (lines 228 to 853):

- Time: discrete time, counted in tics.
- Address and AddressList: user's address, and list thereof.
- Cadence: Ring or SpecialTone (not used by our restricted set of features).
- PIN: validPIN or invalidPIN, instead of a real personal identification number.
- Message: used for announcements.
- TriggerName and ResponseType: IN triggers and their responses.
- LogType, LogRecord, and Log: for the list of log records in the OS.
- Feature and FList: for lists of features.
- SInfo and SDB: for the database of subscriber information in the switch.
- SCPit, SCPinfo and SCPDB: for the database of feature parameters in the SCP.
- StatItem, Stat and Status: for the database of status items in the switch.
- StubPath and SPList: entry/exit points of each stub in the maps, and list thereof.

These abstract data types support the representation and the manipulation of information for the thirteen features described as UCMs in [35], and not only for the four features on which we focus in this report.

#### Structure

LOTOS gates were used to represent individual events shared between the network components (Figure 3). These components are represented as LOTOS processes and are synchronized on common gates. Each event in the Chisel diagrams of the contest description (i.e., each responsibility in the UCMs) is mapped onto a unique gate. Therefore, instead of using gate splitting for representing the on-hook and off-hook events on the user/switch interface (as in user2switch!onHook and user2switch!offHook), we have two individual gates (onHook and offHook). Having individual gates permits more specific compositions between processes and, more importantly, between the specification and the test processes.

Since we are designing the system from the user's viewpoint, some events will be observable while others will remain hidden within the system. Hence, the observable events are the ones on the switch-to-user and user-to-switch interfaces, and are enumerated in lines 59 to 79. The hidden events are those on the switch-to-SCP, SCP-to-switch, and to-OS interfaces (lines 860 to 870).

We also created four additional events. The hidden event Time is used by the switch to get the current time from a global clock. We use three other observable events to improve the testability of our specification. Init allows the initialization of all the databases used by the network components with users' values (likely to come from a test case). CreateUser is used to create users (originators and terminators) and specify their initial state. Finally, Query's purpose is to allow a test case to verify the log in the OS at the end of the test. The top-level process structure itself is derived from the network components and the way they interact with each other (lines 875 to 897). Figure 17 illustrates this structure with titled boxes for components, local variables and databases (with their type) between parentheses, and lines for the LOTOS synchronization operator (|[...]|). Because this is a binary operator, artificial groupings (boxes without titles) become necessary, and they may not have any logical meaning. Each of these lines represents the set of common gates on which the two sides may synchronize. The **GlobalTime** process stands for the global clock mentioned in the contest description, which is queried by the switch on several occasions.



Figure 17 Top-Level Process Structure

The dashed boxes for **User** indicate that these processes are created dynamically within **UserFactory** and that they interleave (|||) with each other. Figure 18 presents a MSC that illustrates how we can create two users with their own identity and subscribed features.



Figure 18 How the UserFactory Process Works

Components to which stubs are bound have sub-processes, one for each stub. Moreover, dynamic stubs may themselves have multiple sub-processes, one for each plugin. The stub pro-

cess is then used to specify the composition between the possible plugins. Each of these processes receives a list of entry/exit points (type SPList) as input and then outputs another such list upon termination.

#### Process Behaviour

As illustrated in Figure 15, UCM paths define the behaviour of the components over which they pass. Components are thus responsible for the events and responsibilities bound to them, and for the implementation of their causal relationships. For the construction of process behaviour, we only consider the four features that interest us, and the others are left to future work. Moreover, for simplification purposes and for conformance to the Chisel diagrams, the specification considers only one call session, i.e., it is not possible to initiate a sequence of call sessions (the behaviour of the switch is not totally tail-recursive). We cannot possibly explain all the synthesis decisions that were made, but we illustrate the main concepts with three examples.

The User process has multiple path segments to take care of. The originator and terminator roles are merged together to form this unique process. Their integration results in seven alternatives between different multi-sequences (trees) of events. As an example, consider the path segment from INTL that crosses the originator in Figure 8. The abstract sequence <ask for PIN, dial **PIN**> has to be implemented somehow in the process. The resulting multi-sequence is specified in lines 1003 to 1012. AskForPIN is an announcement received from the switch through the Announce gate. At this point, we need to note that the generation of this announcement has to be reflected symmetrically in the Switch process. This event is then followed by two alternatives, the first one corresponding to the event in the abstract sequence, i.e., Dial with PIN as a parameter (to be provided by the test case, hence the ? instead of the !), followed by a recursive instantiation of the User process. The second one, although not part of the abstract sequence, comes from the fact that Dial is a point where a disconnection may occur (Section 3.3), hence the OnHook followed by a **stop**. The userId parameters are used to distinguish between different instances of the User process. Again, these events have to be generated from the synchronizing process, which is switch in this case. Notice that this multi-sequence corresponds to the states 4, 5, and 13 in the Chisel diagram for INTL (Figure 4).

The second example, also from INTL, relates to the behaviour of the SCP. INTL has two paths crossing this component: the first one "implicitly" checks whether or not we are in the restricted *TeenTime* period (known by the SCP database), and the other checks the validity of the PIN (again from the SCP database). A call-return mechanism implements these "implicit" checks: they are caused by a Resource event and result in a Response event. Appropriate parameters for Resource include user and time for the checking of *TeenTime*, and the Response contains the boolean value resulting from the evaluation of the IsInTeenTime predicate (lines 1465 to 1468). For the PIN validation, user and pin are needed as input parameters, and this result in a Response containing the CONTINUE message when the PIN is valid. When the PIN is invalid, then a SEND\_TO\_RESOURCE message is sent back, followed by the reception of a Resource request resulting in a DISCONNECT Response (lines 1474 to 1487). Note the following points:

- These multi-sequences correspond to the states 1, 6, 8, 9, 11, and 13 in the Chisel diagram for INTL (Figure 4). Since there are two states numbered 13 in this diagram, we have to specify that we are referring to the one in the lower-left corner.
- These Resource requests and Response must be mirrored in the Switch (see lines 1200, 1201, and 1216 to 1226).

- The Switch is really the component that decides what to do with the result provided by the SCP for the checking of *TeenTime*. Therefore, the OR-fork (where paths split) should probably be located in the Switch. In general, UCMs do not claim or intend to specify where decisions are made, but it is always better to have the OR-forks reflect these locations when they are known.
- State 9 in Figure 4 regroups two events that can occur concurrently (or in any order). The Switch has to synchronize on these two events. Lines 1222 and 1223 of the specification in Appendix A specify that the Switch prescribes one ordering. We used only one possible refinement in order to reduce the state space during validation. We refined in this way many Chisel states that contained the ||| operator.

For the last example, we look at the specification of a simple stub, namely display (Figure 9). This stub has a process (DisplayStub) that is instantiated by process User at line 979, concurrently with the start of the ringing (StartR). Within this stub (described in lines 1029 to 1044), the two plugins (default and CND, see Figure 13) are specified as mutually exclusive alternatives. Since these are quite simple plugins, no other sub-processes seemed necessary. As soon as the terminator subscribes to CND, the CND plugin is selected. Note that this process has an inPaths parameter of type SPList, which allows the calling process to indicate from which entry points in the stub the events are coming. In our case, the stub has only one entry point and the process is instantiated with the value inDisp1. Upon successful termination, the process exits with another SPList that contains the list of exit points in the stub that should be activated. Again, since there is only one exit point, both plugins exit with outDisp1 for this result. These values are then used by the calling process to reason about what happened within the stub (lines 981 to 984).

These examples have illustrated some of the basic concepts used to synthesize the LOTOS specification:

- Components are implemented as processes synchronized on their common channels/ gates.
- Because of their reactive nature, most components are specified with implicit recursive behaviour.
- Hidden gates are used for what is not observable by the user.
- Path segments in one component are integrated together, often as alternatives (could also be integrated as concurrent multi-sequences, depending on the UCM context).
- UCM activities are implemented as gates or as messages exchanged between components.
- Composition with the disconnection phase is applied to specific points in the global UCM.
- ADTs are used to represent databases and operations, and to evaluate conditions.
- Symmetry is enforced in synchronized actions (actions in one process must be mirrored in the other synchronized processes, unless locally hidden).
- Chisel states with the ||| operator are refined into simpler sequences, for the reduction of the state space.

Several additional rules for define for the specification of the stubs:

- Components with stubs have sub-processes, one for each stub.
- Dynamic stubs may have multiple sub-processes, one for each plugin.
- The stub process is used to *specify the type of composition* between the possible plugins.

• Each stub process receives a list of entry/exit points as input and then outputs another such list upon termination.

These concepts have been used throughout the construction of the specification, although we deviated from them on several occasions while debugging the integration.

### 4.3 Testing

We are now about to derive test cases for validating POTS and the individual features against their requirements, the Chisel diagrams (step (4) in Figure 1).

#### Structure of the Test Suite

In LOTOS, the testing is done through the composition of test processes with the specification [9]. Often, LOTOS test processes are sequential, monolithic, and deterministic in nature. However, through process instantiation, LOTOS test processes can be built on top of each other, hence reusing part of previous test processes in new ones. We make use of this capability in our strategy. We will define shared processes that represent sub-sequences of test cases. We call these processes *common behaviours*. In the conformance testing framework used in telecommunication systems [27], these common behaviours correspond in a way to *test steps*, which may be instantiated from multiple test cases and other test steps.

Figure 19(b) shows the bottom level of LOTOS test processes, composed solely of common behaviour processes for POTS. They are reused by the POTS test cases, and also by common behaviour processes for individual features. On top of the latter, we construct test processes for individual features, and also for each pair of features. Common behaviour processes then become reusable by many test cases, which simplifies the generation of test suites and increases the consistency among test cases.



Figure 19 Construction of the Test Suite

Figure 19(c) presents the typical code structure in common behaviour processes. They are mainly composed of simple expressions that terminate with an *exit code* (exit(n)). With LOLA, test cases do not need to be sequential or deterministic, so alternatives and explicit non-determinism are allowed in common behaviour processes. Note that many alternatives are preceded by the internal action i. This non-determinism ensures, under LOLA, that all branches in the test case will by selected and covered at testing time.

In Figure 19(a), the typical code structure illustrates that a test case provides the system configuration and verifies the exit codes. More specifically, the system is first initialized (by Init), users are created according to the mechanism shown in Figure 18, and the test cases themselves are performed by instantiating common behaviour processes. The exit code is then captured and used to validate the log against its predicted value.

During the testing, a deadlock in a test case for POTS or for an individual feature indicates that there is a bug that needs to be fixed. When all these test cases pass successfully, a deadlock in a test case for a pair of features indicates an unexpected interaction. Interactions will be covered in Section 5.

#### POTS Common Behaviour

We constructed a tester for POTS using six processes (lines 1534 to 1647). They have two parameters, representing the originator and terminator users, whose values are provided by the test processes. POTS states 1, 2, 4, 5, 13, and 15 were defined because they were referred to by the Chisel diagrams of other features. This is one of the main reasons why such common behaviour can be so easily reused.

A LOTOS *canonical tester* is a process that tests all of the behaviour of an implementation for conformance to a specification [9]. Inspired from this theory, we used the Chisel diagrams in order to obtain a reduced set of test cases, while maintaining a good validation power. In essence, a canonical tester has the same traces as the specification it aims to check, but it forces the coverage of all alternatives when the environment has a decision to make. For instance, Figure 20(a) shows a simplified Chisel diagram for POTS, for which a LOTOS interpretation is provided in (b). In its test process (c), the addition of an internal action i before Dial and OnHook (corresponding to the dark area) forces the composition to check both alternatives, which are both valid user inputs from the system's point of view (Figure 20(b)). In a similar way, if the system makes an internal decision, by using guarded behaviour or with hidden events, then the test process has to accept all possible outputs accordingly. The light shaded area presents a case where the system offers either Ring or BusyTone depending on its internal information about users' status. A real LOTOS canonical tester would also take care of all possible values associated to the parameters. We chose not to follow this strict rule because we wanted to generate common behaviour processes where the parameters are set during the initialization phase in a test case. Our test cases do not check all possible configurations, while a canonical tester would (which is seldom feasible).



Figure 20 Example of a Canonical Tester for a Chisel Diagram

Note that we assigned an exit code to the leaf at the end of each branch in a Chisel diagram. This allows test processes to determine what branch has been selected, in order later to check the validity of the log collected by the OS.

Interleaved events in a Chisel diagram should, according to the canonical tester theory, require that all possible combinations of events be covered explicitly. For instance, state 9 in Figure 4 has Announce ||| Resource. Let's rename this A ||| R and assume these are events provided by the user to the system. Following the LOTOS expansion theorem, this expression is equivalent to  $A_{iR}$  []  $R_{iA}$ , hence the canonical tester would need to be  $i_{iA_{iR}}$  []  $i_{iR_{iA}}$ . However, we will leave A ||| R as is in the tester for two reasons:

- On several occasions, we implemented only one alternative in the system (see the end of the previous section). By this refinement, the system has already made the decision, and thus the user needs to accept it.
- Leaving the ||| operator leads to simpler expressions.

Processes POTS\_1 to POTS\_15 represent the lowest layer of common behaviour, and will now be used, directly or indirectly, by almost all of the other test processes.

### POTS Test Cases

Often, more than one test case will be required to cover a Chisel diagram, because initial states and conditions are necessary. POTS has only one *precondition*: whether or not the terminator side is busy. Hence, two test processes can cover all the states in the Chisel diagram (lines 1648 to 1714). Process tPOTS1 tests the cases where the terminator side is not busy, whereas tPOTS2 takes care of the cases where the terminator is busy. They both use POTS\_1 as their start point. Note that the names of all test processes start with a lowercase t, while common behaviour processes start with a lowercase c (except for the POTS common behaviour processes). Test processes for pairs of features are prefixed with fi.

#### Test Cases for Individual Features

These tests check that each feature acts properly when being the only one active (lines 1715 to 2123). The previous test suite (for POTS) still needs to be checked because, in the absence of any active feature, what remains must be the regular POTS behaviour.

Table 1 presents the 10 test processes used for the coverage of INTL, CND, INFB and TCS, according to their respective Chisel diagram. Each test was created by providing an initial configuration (according to the conditions shown in the Chisel diagrams and the individual UCMs) and by calling the appropriate common behaviour process. These tests were applied to the specification, and results were collected (steps (7), (8), and (9) in Figure 1).

For each test, we included its purpose (according to the preconditions that need to be satisfied by the initial configuration), the common behaviour process it uses<sup>1</sup>, and how many unique global sequences were generated by its composition with the specification. Each of these traces could be represented as unique message sequence charts from the user's point of view. Some nondeterminism inside the system (which would create many more global sequences) has been abstracted from in this experiment; on-the-fly reduction techniques, which preserve testing equivalence, have been used while testing with LOLA. All of them were successful, therefore we do not

<sup>1.</sup> A common behaviour processes can call other such processes. For instance, POTS\_1 calls POTS\_2, which in turn calls POTS4 and POTS\_15. POTS 4 calls POTS\_5 and POTS\_13.

have any indication that POTS and the individual features are faulty in our system. The validation of the system then continues with the detection of unexpected interactions between pairs of features.

Feature	Test Process	Purpose According to Preconditions	Used Common Behaviour	Number of Global Sequences
INTL	tINTL1	TeenTime not restricted: allow call.	POTS_1	29
	tINTL2	TeenTime restricted, valid PIN: allow call.	cINTL1	30
	tINTL3	TeenTime restricted, invalid PIN: do not allow call.	cINTL2	2
CND	tCND1	Terminator idle: display.	cCND1	84
	tCND2	Terminator busy: do not display.	POTS_1	2
INFB	tINFB5	Terminator idle: affect billing.	POTS_1	29
	tINFB2	Terminator busy: do not affect billing.	POTS_1	2
TCS	tTCS1	Terminator idle, A not on Screened B: allow call.	cTCS1	29
	tTCS2	Terminator busy, A not on Screened B: busy tone.	cTCS2	2
	tTCS3	A on Screened B: announce screened message.	cTCS3	2

 Table 1 Description of Test Processes for Individual Features

# 5 DETECTING FEATURE INTERACTIONS

# 5.1 Test Cases for Detecting FI

The tests generated in this section come from step (5) in Figure 1. In theory, if the same type of integration used for merging the individual UCMs and if the same composition used in the stub processes are used again during the generation of the test cases for pairs of features, then we should not find any inconsistencies, and perhaps not a single unexpected interaction. In practice however, integrating two features in a test sequence is much easier than integrating *n* features in a system (where n > 2). This is one of the main reasons why tests for pairs of features are necessary. Although they cannot cover everything there is to check, they represent a pragmatic and efficient way of attacking the problems of conformance to the requirements and interoperability between features.

Having a set of four features, we have to check  $n^*(n-1)/2 = 4^*(4-1)/2 = 6$  different pairs of features<sup>1</sup>. We developed a test suite composed of six test processes (lines 2124 to 2864), described in Table 2. Each process contains many test cases that have different initial configurations. Their number can be found in the same table, as well as an enumeration of the common behaviour processes used, and the number of global sequences generated by LOLA using the *TestExpand* command.

We do not intend to explain the purpose of each of the 25 test cases. Comments in the code of the tests provide that information. As an example, we nevertheless present the purpose of the three test cases that validate the pair INTL-CND.

<sup>1.</sup> In this study, the assumption is that a feature cannot interact with itself. This is however an incorrect assumption in general. Hence, we would also need to cover the pairs INTL-INTL, INFB-INFB, CND-CND, and TCS-TCS. The numbers of pairs would become  $n^*(n+1)/2 = 4^*(4+1)/2 = 10$ .

FI Test Process	Number of Test Cases	Used Common Behaviour	Number of Global Sequences
fiINTL_CND	3	cCND1, cINTL2	170
fiINTL_INFB	3	POTS_1, cINTL1, cINTL2	61
fiCND_INFB	2	cCND1, POTS_1	86
fiINTL_TCS	9	cTCS1, cTCS2, cTCS3, cINTL1, cINTL2	74
fiCND_TCS	4	cCND1, cTCS2, cTCS3	90
fiINFB_TCS	4	cTCS1, cTCS2, cTCS3	35

 Table 2 Description of Test Processes for Pairs of Features

Among the four combinations of its two preconditions, INTL has only 3 cases to check (TeenTime not restricted, TeenTime restricted and valid PIN, TeenTime restricted and invalid PIN), whereas CND has two other cases (terminator busy, terminator idle), unrelated to the ones of INTL. A Cartesian product would give us a total of 6 global cases. However, the 3 cases where the terminator is busy are not interesting to us. For these cases, CND acts exactly like POTS would. Hence the pair INTL-CND would act like the pair INTL-POTS, or in other words simply INTL, already covered by tINTL1, tINTL2, and tINTL3. With this purified domain partitioning, the three remaining cases provide new constraints on the values to be used in the preconditions attached to the global UCM (Figure 6). In essence, this UCM specifies the way these two features are integrated together, and the test cases have to reflect their end-to-end behaviour accordingly:

- Terminator idle, TeenTime not restricted (case 1): the end-to-end UCM acts like cCND1 from the user's point of view.
- Terminator idle, TeenTime restricted and valid PIN (case 2): the end-to-end UCM acts first like cINTL1 (part about the request for the PIN) and then like cCND1 (display of the number) from the user's point of view. Unfortunately, because our common behaviour processes specify all events until the end of a scenario, cINTL1 cannot be used as is and must be partly duplicated in fiINTL\_CND, and then cCND1 can be used.
- Terminator idle, TeenTime restricted and invalid PIN (case 3): the end-to-end UCM acts like cINTL2 from the user's point of view.

If we had explicitly tested all the events that are currently hidden in the system, it would have been much more difficult to define reusable common behaviour. Having the system specified as a black box increases the reusability of these processes, although they do not ensure that what happens inside the system corresponds to what would be expected. We can only assume that if the end result is fine, then the system behaved properly.

# **5.2 Unexpected Interactions**

With our first specification, all our test cases passed successfully, except for fiINFB\_TCS (steps (7), (8), and (9) in Figure 1). LOLA returned three different traces that led to unexpected dead-locks. The first trace, presented below, is related to the first test case in fiINFB\_TCS: the idle terminator (B) has subscribed to INFB and TCS, and the originator is not on the screening list. In this scenario, the originator (A) on-hooks first, but it is also billed instead of the terminator.

```
init ! insert(sub(usera,noflist,undefined,undefined,noaddlist,validpin),
        insert(sub(userb,insert(tcs,insert(infb,noflist)),undefined,undefined,
                                       insert(userc,noaddlist),validpin),nosdb))
      ! nostatus ! noscpdb ! inittime;
offhook ! usera;
dialtone ! usera;
dial ! usera ! userb;
startar ! usera ! userb;
startr ! userb ! usera;
offhook ! userb;
stopar ! usera ! userb;
stopr ! userb ! usera;
i; (* time ! inittime *)
i; (* logbegin ! usera ! userb ! usera ! inittime *)
onhook ! usera;
disconnect ! userb ! usera;
i; (* time ! tic(inittime) *)
i; (* logend ! usera ! userb ! tic(inittime) *)
onhook ! userb;
i; (* exit (0) *)
stop
```

Since we know on which network interfaces these events occurred, we can represent such LOTOS traces as synchronous MSCs, a form more appropriate for illustration of linear scenarios. The MSC for this trace is shown in Figure 21.



Figure 21 First FI, Originator Billed Instead of the Terminator

The error in the billing was detected when the test case queried the log from the OS and could not synchronize on the expected value. The problem here is that TCS was selected, but not INFB. Hence, the person to be billed was the default one, i.e., the originator.

The second interaction trace is similar in nature, but this time the terminator on-hooks first (Figure 22).



Figure 22 Second FI, Originator Billed Instead of the Terminator (Who On-hooks First)

The last interaction trace is related to the fourth test case in fiINFB\_TCS: the idle terminator (B) has subscribed to INFB and TCS, and the originator is on the screening list (Figure 23). The call should be blocked by TCS, but it goes through because INFB was selected and not TCS. The deadlock occurs when the test case expects a specific announcement (ScreenedMessage) while the switch attempts to send something else (StartAudibleRinging or StartRinging).



Figure 23 Third FI, Call Should Be Blocked but Is Not

Appendix B presents the erroneous part of our original specification. The choice between the TCS plugin and the INFB plugin in the **process-call** stub, which both override the default plugin, was left open (i.e., non-deterministic). When integrating the UCMs, we did not know if other types of constraints were necessary for these two features to work properly together. Even from a UCM perspective, a mutual exclusion would cause problems, but this is a detail that was somewhat buried down in the composition within the stub. This is why a more precise and rigorous detection technique appears necessary once the integration is completed.

A sensible solution to this problem would be to give a sequential priority to TCS over INFB in the stub, i.e., INFB would be selected only if TCS allows it. The specification in Appendix A implements this solution. In the end, all of our test cases (POTS, individual features, and pairs of features) passed successfully, and hence no expected interactions seemed to remain in the global specification.

### Fixing the UCM

Giving TCS priority over INFB (and over the other features in the **process-call** stub) can be reflected back at the UCM level in different ways. One simple way, which does not necessarily reflect the stub structure in the current LOTOS specification, would be to move the TCS checking at a higher level that what it used to be in **process-call**. Therefore, we move the first condition from Figure 12 to Figure 9 and we remove the TCS plugin. As a result, the paths around this stub would be as prescribed by Figure 24:



Figure 24 New Surroundings of Process-call Stub in Figure 9

In this figure, it is important for the feedback loop (used by call-forwarding features) to go back *before* the TCS screening list is checked. This is to ensure that intermediate originators in a forwarded call will be screened as well. Fixing a UCM could result in new types of interactions that were not present previously. For instance, the checking of the condition *on*-*TCS* is meaningful only if the terminator has subscribed to the TCS feature (otherwise, TCS becomes mandatory). This is why the test suite needs to be reapplied to the new resulting specification (this is called regression testing).

# **5.3** Ensuring Coverage with Probes

The generation of test cases from scenarios is an *a priori* approach to validation. We assume that the functional coverage is achieved when all tests execute as planned. However, the quality of the specification and of the test suite may be enhanced by using a syntactic approach called *structural coverage*. If some required coverage is not reached, new tests can be added *a posteriori*.

Probe insertion is a well-known white-box technique for monitoring software in order to identify portions of code that have been exercised, or to collect information for performance anal-

ysis. A program is instrumented with probes (generally counters initially set to 0) without any modification of its external functionality. Test cases "visit" these probes along the way, and the counters are incremented accordingly. Probes that have not been visited might indicate that the test suite is incomplete or that part the code is not reachable.

We have adapted this approach for LOTOS specifications (steps (6) in Figure 1). Special (\*\_PROBE\_\*) comments are added at specific places in the specification, and then they are translated automatically into hidden Probe gates with unique identifiers. Careful insertion of probes leads to a new specification that is observationally equivalent to the original one, and therefore they do not affect the verdicts of the tests. During testing, labeled transition systems (LTSs) resulting from the composition of each test with the specification can be generated, and occurrences of probes counted. If a probe is not visited by any of the test cases, then the structural coverage of the specification is incomplete. More specifically, this indicates that some code could be unreachable in the specification, or that the test suite is incomplete.

We inserted 55 probes in the specification of the system only (no need to cover the behaviour of the test as this is done through plain testing). Out of these, 5 were not covered by the whole test suite, but for good reasons (see Table 3). Therefore we conclude that the structural coverage of the specification by the validation test suite is adequate, and that no further test cases are required.

Probe Number	Line Number	Reason For Not Being Covered
P_35	1339	Case not specified yet. OutBusy2 is used by one of the remaining unspecified features.
P_37	1353	Case not specified yet. OutPC4 is used by one of the remaining unspecified features.
P_52	1509	Case not specified yet. The air interface is to be used by the Cellular feature.
P_53	1513	Case not specified yet. The air interface is to be used by the Cellular feature.
P_54	1518	Bug in LOLA's <i>TestExpand</i> . The OS is not reinstantiated as the occurrence of the internal probe is not forced by a test case.

#### Table 3 Probes Not Covered by the Test Suite

Note that we also measured the coverage of the test suites for POTS/individual features and for pairs of features. Both test suites covered all probes except the five already mentioned.

# 6 DISCUSSION

### 6.1 Adding New Features

Adding new features has a direct impact on the global UCM, the specification, and the test suite. Here are come comments on the scalability of this approach.

#### Impact on the UCMs

In our experience, the integration of three new features to the first 10 ones, which were already integrated together, did not have a major impact on the global UCM. The root map did not have to change, but a **busy** stub (with a new output path) had to be added in Figure 9. The disconnection UCM was slightly adjusted, and new plugins were created for the **process-call** and **billing** stubs.

The impact is probably proportional to how coupled the features are in a map. The more a map is decoupled and modular (for instance, by using stubs), the less likely major modifications will be necessary. More experiments on this aspect still need to be done in order to have better conclusions.

### Impact on the Specification

Since the specification reflects the global UCM, the conclusions are basically the same as for the impact on the UCMs. In our experiment, we added a few new gates (for the new air interface) and added appropriate ADTs, with their operations, to support new data structures. Some previous types were also expanded to cover the new features. The impact on the structure is not really known because we have not fully implemented these three features yet.

#### Impact on the Test Suite

The addition of a feature has a profound impact on the test suite. Like before, we will distinguish between test suites for individual features and test suites for pairs of features. Each feature has a set of *c* preconditions (enumerated in Table 1), the conditions that may affect the result for one same stimulus. In theory, since each of these is either true or false, the upper bound on the number of combinations is  $u = 2^c$ . That is, the number of test cases for each feature grows exponentially with the number of preconditions, which is no surprise. We need one test case for each of these combinations, unless some of these are unnecessary. UCMs can help determine which are relevant and which are not by following the paths and associated conditions. For instance, when *TeenTime* is not restricted in INTL, whether the PIN is valid or not has no impact, and one of these two combinations can be dropped. Therefore, due to this test selection approach, there is a potential gain at this level. We define this gain as g = u - t, where *t* is the number of test cases actually present in our test suite (Table 4).

Feature	Number of Conditions ( <i>c</i> )	Theoretical Upper Bound ( <i>u</i> = 2 <sup><i>c</i></sup> )	Actual Number of Test Cases ( <i>t</i> )	Gain ( <i>g = u - t</i> )
INTL	2	4	3	1
CND	1	2	2	0
INFB	1	2	2	0
TCS	2	4	3	1

Table 4 Number of Test Cases for Individual Features

For pairs of features, UCMs can help again reducing the number of necessary test cases w.r.t. the upper bound. The number of conditions for a pair (c) is the cardinality of the union of the two sets of preconditions (as some conditions may be shared, which is a major cause of interaction), and the theoretical upper bound is again  $u = 2^c$ . However, we already know from the previous table which cases have to be looked at. Hence, we define p as the product of the two number of actual test cases ( $t_1 * t_2$ ). This leads us to a better upper bound (b) defined as the minimum of u and p. As explained at the end of Section 5.1, we can get a better partitioning of the input domain by removing the cases that are equivalent, from a path point of view, to cases already covered in the test suite for individual features. Another gain can be achieved here (g = b-t), illustrated in Table 5.

Pair of Features	Number of Distinct Conditions ( <i>c</i> )	Theoretical Upper Bound ( <i>u</i> = 2 <sup>c</sup> )	Product of Number of Cases ( <i>p</i> )	Better Upper Bound <i>b = MIN(u, p)</i>	Actual Number of Test Cases ( <i>t</i> )	Gain ( <i>g = b - t</i> )
INTL-CND	3	8	3*2 = 6	6	3	3
INTL-INFB	3	8	3*2 = 6	6	3	3
CND-INFB	1	2	2*2 = 4	2	2	0
INTL-TCS	4	16	3*3 = 9	9	9	0
CND-TCS	2	4	2*3 = 6	4	4	0
INFB-TCS	2	4	2*3 = 6	4	4	0

 Table 5 Number of Test Cases for Pairs of Features

Note that a negative gain implies a non-optimal selection of test cases. This represents a simple way to measure the efficiency of this selection.

In conclusion, the number of pairs of features is  $n^*(n-1)/2$ , and for each the number of test cases grows exponentially with the number of distinct conditions. The impact of the integration of a new feature will be higher if new types of conditions have to be accounted for in the input domain.

### 6.2 Performance

From a tool perspective, testing with LOLA seems a very efficient solution for the validation of prototypes and the detection of feature interactions. The compilation of this 2864-line specification and the execution of all the test cases take about 30 seconds on a low-end PC (Cyrix P150, Win95, 48MB RAM). This means that this technique can be used in an iterative and incremental process where numerous modifications, additions, debugging sessions, and executions of regression test suites need to be supported.

The verification of the structural coverage (with probes), which is usually performed towards the end of a macro-iteration in the design cycle, takes about 7 minutes of processing time on the same platform. For this part, internal actions must not be simplified through on-the-fly equivalence reductions (otherwise, we could not "observe" the probes in the resulting LTSs), and thus more time and resources are required.

Once probes are inserted, some specifications may result in a number of states too large to be handled by LOLA and similar tools. We see at least three practical solutions to this problem

- Use half the probes for a first measure, then use the other half for a second one. The set of probes visited is the union of the probes visited in each experiment. This is the best solution.
- Simplify the test processes by splitting them into many sub-tests (with an equivalent testing power).
- Use heuristics in the execution of the tests. LOLA allows to test a specification according to upper bounds in memory usage, or according to an estimate of the coverage of the possible branches. Although the functional coverage is not complete anymore, experience shows that a complete structural coverage might be achieved anyway.

### 6.3 Improved Call Structure

The abstract underlying structure in our UCMs is insufficient for the specification of all the features. The current behaviour of our switch is tightly coupled to the progression of one unique call session. For call sessions involving more than two parties (e.g., for the support of features such as INBL, INFR, 3WC, INCF and CW), the current call structure needs to be improved. Call sessions need to be instantiated upon request, and the status database needs to be decoupled from the current Switch process (Figure 25(a)) in order to be accessible to these sessions. A new **StatusDatabase** process (where appropriate values from *sdb* and *status* would be stored), with new query and update messages, would solve this problem (Figure 25(b)). ADTs need to be partitioned accordingly, and all of the specification must then reflect this modification. This kind of structure is similar to those used in many LOTOS specifications for telephony systems [20][22][23]. IN-like architectures, and especially the Basic Call Model, could also be considered.



Figure 25 Current and Improved Switch Structures

The UCM structure, derived from the given network structure (Figure 3), did not specify anything about the internals of the switch. To keep our synthesis straightforward, we did not introduce any new structural entities. However, for the sake of extensibility of the specification, this improvement can hardly be avoided. New components are needed at the specification level, and they probably need to be mirrored at the UCM level. However, this is left for future work.

# 6.4 Limitations of Plugins, Bindings, and Composition

We have observed the following limitations of the UCM notation while integrating the scenarios:

- Although the stub/plugin mechanism is useful for abstraction, modularity, and dynamic behaviour, its use in a global map makes the end-to-end scenarios more difficult to visualize at a first glance. Often, the reader has to mentally flatten the global map to get a better understanding of these scenarios.
- The binding of a plugin to a stub is done through an external mechanism (the binding relation), which is not visual.
- The composition of plugins in a stub is described at a lower level of abstraction. Not having this information (whether it is visual or not) at the UCM level makes the selection of plugins very ambiguous. On the other end, it allows for the designer to play with different alternatives and to decide which composition should be used. But the problem remains that once this composition has been decided, it is documented with another notation (LOTOS in our case).

Designers must use the stub/plugin mechanism with care. Otherwise, they might defeat the intent of UCMs which is to provide a good bird's eye view of the system.

# 6.5 Comparison with Other Techniques

We include here a short discussion on three related approaches to design with UCM and to detection of FI.

#### Agent Systems

A path that goes from UCMs to agent prototypes was illustrated by a feature interaction example in [15][16]. This approach also aims to avoid interactions at design time (with UCMs), but the main property of these agents (implemented in CLIPS and Java) remains the opportunistic avoidance of interactions at *run time*. However, the mapping between UCMs and these agents is still fairly immature, and detection/validation techniques on this agent environment are still ad hoc. For these reasons, and also because the FI contest was mainly about detection at design time (not avoidance), we took a somewhat different direction that led to the current exercise. How LOTOS would fit in a design process that involves UCMs and such agents is still the topic of future research.

### GCS and GPRS

Design and validation of LOTOS specifications from UCMs have been performed for two previous projects: a *Group Communication Server* (GCS) in [4][6], and the packet-switched mobile telephony standard *General Packet Radio Services* (GPRS) in [5]. In both cases, the integration of the UCM scenarios was done directly at the LOTOS level. There was no global UCM, and no stub was used. Test cases were generated solely from the UCMs (requirements were in plain English, without anything similar to Chisel diagrams) and applied to the specification in order to validate the integration.

Since the burden of the integration is pushed down to the level of LOTOS, designers not too familiar with this language may have a hard time coping with the construction of the specification. Moreover, other people not involved in the LOTOS part (clients, marketing, management, etc.) would not know anything about how the individual UCMs fit together. For these two reasons, although the GCS and GPRS experiments were successful in the sense that moving to LOTOS directly also resulted in correct specifications and validated test suites, integrating the scenarios at the UCM level seems a better alternative.

In the GCS case study, we derived rejection test cases for each of the individual scenarios (as illustrated in Figure 16). In the current example, doing so will require a better knowledge of what could go wrong in the system (which can be anything right now). If Chisel diagrams had included branches labeled "reject", "interdicted" or "forbidden", directly in the requirements, then generating rejection test cases would have been easy. Again, an OPI-like notation would be of great help in this context.

### Faci's Approach

In [22], Faci presents a detection technique also based on the integration of scenarios and the use of the LOTOS testing theory. This approach makes a distinction between the concepts of composition and integration. *Composition*, noted  $f_1|[]|f_2$ , expresses the synchronization of features on their common actions with POTS and their interleaving on their independent actions. *Integration*, noted  $f_1*f_2$ , expresses the extension of POTS with *n* features (two in the examples), such that each feature is able to execute all of its actions which are allowed in the context of POTS, when the other features are disabled. Features are captured as labeled transition systems (LTSs) instead

of as UCMs. Integration relates very well to our own UCM integration (validated by the test cases for individual features), whereas the composition simply represents the generalized synchronization operator and does not relate to anything in our methodology. The approach states that an interaction exists between *n* features if their *integration* does not *conform* to their *composition*.

Conformance is checked through validation test cases, from the user's point of view, similarly to what we are doing. Test cases are derived manually (using "knowledge and experience") from the composition specification, and then they are applied to the integration specification. When a deadlock occurs between a test case and the integration specification, an interaction is said to be detected. This last specification is generated manually at the LTS level, which is far less scalable and modular than generating specifications from global UCMs. Indeed, all the examples provided in this thesis contained only pairs of features integrated together, for obvious complexity reasons. UCMs are a means to integrate scenarios while avoiding some interactions, and they allow for multiple complex features to be considered (13 in [35]).

Faci's approach leads to multiple feature interactions that we already referred to as trivial and artificial. Indeed, any integration operator (\*) other than the generalized synchronization (|[]|) is very likely to cause deadlock situations. The test suite, although it could be generated almost automatically from the composition, is of low quality as it does not consider the way the features were integrated together. The test suites generated from UCMs are much more representative of the intended system behaviour, and they are more likely to be reusable down the road towards the implementation.

### 7 CONCLUSIONS

This report presented an approach for the avoidance and detection of feature interactions at design time. Features are captured as UCM scenarios, integrated in one global map with stubs and plugins, and then transformed into a LOTOS specification. Test cases are generated from the requirements (Chisel diagrams in our case) and from the UCMs. We use them to validate the integration and to detect unexpected interactions.

UCMs describe features (and systems in general) at an interesting level of abstraction. We showed how, during their integration, some interactions can be avoided by insuring deterministic and complete preconditions and by composing plugins in stubs according to the intent of the features. Many features can be considered in a global UCM, and they can be represented as one or more plugins in one or more stubs. Further design decisions are necessary when synthesizing the specification, although the burden of the integration is mostly taken care of at the UCM level. The canonical tester theory and test selection techniques based on UCMs help us generate reduced sets of test cases for individual features. Test suites for detecting interactions between pairs of features are constructed on top of existing test cases, hence promoting reuse and consistency among tests. The generation of these tests is guided by the integration done at the UCM level, which again reduces the number of necessary cases to cover. Several interactions between a pair of features were detected. They were caused by the composition of plugins in a stub, and this is where we fixed the problem at the LOTOS and UCM levels. The quality of the specification and of the validation test suite is finally assured by measuring the structural coverage through probe insertion. Good tool support for the UCM integration (UCM Navigator) and for the validation and coverage measurement of the LOTOS specification (LOLA) suggests that this approach can be used in an iterative and incremental design process.

### Future Work

The following list enumerates several research issues and work items, some of which were already raised in this document:

- Improvement of the call process within the Switch for the support of features involving more than two users.
- Completion of the specification by integrating the remaining nine features. By observing the impact on the specification and on the number of test cases required for validation, it might be possible to learn new lessons.
- Derivation of rejection test cases from UCMs and/or the Chisel diagrams in order to detect more interactions.
- Comparison with other LOTOS-based techniques applied to the same set of features, by detecting interactions in our specification with their approaches (if the tools allow it) and by applying our test cases to their specifications. We could also observe how "trivial and artificial" interactions detected with their techniques have been avoided by our UCMs.
- Linkage of the OPI model to the UCM notation. The intent of a feature would be better described by indicating which events or paths are obliged, permitted, or forbidden to be in the implementation. This would also allow for an easy way of generating rejection test cases.
- Further study of the visualization of bindings and compositions of plugins.
- Finally, we could look at the best way of integrating this approach in a design process that generates agent prototypes from Use Case Maps.

#### Acknowledgements

This work has been supported by *Communications and Information Technology Ontario* (CITO). I would like to thank Dorin Petriu for providing me with the global UCM used in this case study. Andrew Miga and Chris Witham supported us with the UCM Navigator tool. I am grateful to Luigi Logrippo for useful and constructive comments on a first draft of this report. Finally, many thanks to the agent group at Carleton U. and Mitel, and to the LOTOS group at U. of Ottawa for useful discussions.

### 8 **REFERENCES**

- [1] Aho, A., Gallagher, S., Griffeth, N., Scheel, C., and Swayne, D. (1998) "Sculptor with Chisel: Requirements Engineering for Communications Services". In: K. Kimbler and W. Bouma (eds.), *Fifth International Work-shop on Feature Interactions in Telecommunications Software Systems*, IOS Press, 45-63. http://www-db.research.bell-labs.com/user/nancyg/sculptor.ps
- [2] Amyot, D. (1994) *Formalization of Timethreads Using Lotos*. M.Sc. thesis, Dept. of Computer Science, University of Ottawa, Ottawa, Canada. http://www.csi.uottawa.ca/~damyot/phd/msctheses.pdf
- [3] Amyot, D., Bordeleau, F., Buhr, R. J. A., and Logrippo, L. (1995) "Formal support for design techniques: a Timethreads-LOTOS approach". In FORTE VIII, 8th International Conference on Formal Description Techniques, Montréal, October 1995. Chapman & Hall, 57-72. http://lotos.csi.uottawa.ca/~damyot/phd/forte95/forte95.pdf
- [4] Amyot, D., Logrippo, L., and Buhr, R.J.A. (1997) "Spécification et conception de systèmes communicants : une approche rigoureuse basée sur des scénarios d'usage". In: *CFIP 97, Ingénierie des protocoles*, Liège, Belgique, September 1997. http://www.csi.uottawa.ca/~damyot/cfip97/cfip97.pdf

- [5] Amyot, D., Hart, N., Logrippo, L., and Forhan, P. (1998) "Formal Specification and Validation using a Scenario-Based Approach: The GPRS Group-Call Example". In: *ObjecTime Workshop on Research in OO Real-Time Modeling*, Ottawa, Canada, January 1998. http://www.csi.uottawa.ca/~damyot/wrroom98/wrroom98.pdf
- [6] Amyot, D. (1998) Group Communication Server: A Scenario-Based Design Exercise. Telecommunication Research Institute of Ontario, report #1388, Ottawa, Canada, June 1998. http://www.csi.uottawa.ca/~damyot/gcs/
- [7] Barbuceanu, M., Gray, T., and Mankovski, T. (1998) "How To Make Your Agents Fulfil Their Obligations". In: H.S. Nwana and D.T. Ndumu (Eds), *PAAM'98, Third Conference on Practical Application of Intelligent Agents and Multi-Agents*, London, UK, March 1998, 255-276.
- [8] Boumezbeur, R. and Logrippo, L. (1993) "Specifying telephone systems in LOTOS". IEEE Communications Magazine, 31 no. 8 (August), 38-45. http://lotos.csi.uottawa.ca/ftp/pub/Lotos/Papers/svtsl.ps.Z
- [9] Brinksma, E. (1988) "A theory for the derivation of tests". In: S. Aggarwal and K. Sabnani (Eds), *Protocol Specification, Testing and Verification VIII*, North-Holland, 63-74, June 1988.
- [10] Buhr, R.J.A. and Casselman, R.S. (1995) Use Case Maps for Object-Oriented Systems, Prentice-Hall, USA.
- [11] Buhr, R.J.A. (1997) *High Level Design and Prototyping of Agent Systems*, research project description. http://www.sce.carleton.ca/rads/agents/
- [12] Buhr, R.J.A., Elammari, M., Gray, T., Mankovski, S., and Pinard, D. (1997) "Understanding and Defining the Behaviour of Systems of Agents, with Use Case Maps". Poster session, *PAAM'97, Second Conference on Practical Application of Intelligent Agents and Multi-Agents*, London, UK, April 1997. http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/4paam97.pdf
- [13] Buhr, R.J.A., Elammari, M., Gray, T., and Mankovski, S. (1998) "A High Level Visual Notation for Understanding and Designing Collaborative, Adaptive Behaviour in Multi-agent Systems", *Hawaii International Conference on System Sciences (HICSS'98)*, Hawaii, January 1998. http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/agents-ucms.pdf
- [14] Buhr, R.J.A., Elammari, M., Gray, T., and Mankovski, S. (1998) "Applying Use Case Maps to Multi-agent Systems: A Feature Interaction Example", *Hawaii International Conference on System Sciences (HICSS'98)*, Hawaii, January 1998. http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/hiccs98.pdf
- [15] Buhr, R.J.A., Amyot, D., Elammari, M., Quesnel, D., Gray, T., and Mankovski, S. (1998) "High Level, Multiagent Prototypes from a Scenario-Path Notation: A Feature-Interaction Example". In: H. S. Nwana and D. T. Ndumu (eds.), *PAAM'98, Third Conference on Practical Application of Intelligent Agents and Multi-Agents*, London, UK, March 1998. http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/4paam98.pdf
- [16] Buhr, R.J.A., Amyot, D., Elammari, M., Quesnel, D., Gray, T., and Mankovski, S. (1998) "Feature-Interaction Visualization and Resolution in an Agent Environment". In: K. Kimbler and W. Bouma (eds.), *Fifth International Workshop on Feature Interactions in Telecommunications Software Systems*, IOS Press, 135-149. http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/fiw98.pdf
- [17] Buhr, R.J.A. (1998) "Use Case Maps as Architectural Entities for Complex Systems". To appear in: *Transactions on Software Engineering*, IEEE, 1998. http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/tse98final.pdf
- [18] Cameron, E.J., Griffeth, N., Linand, Y.-J., Nilson, Y.-J., Schnure, W.K. and Velthuijsen, H. (1994) "A Feature Interaction Benchmark for IN and Beyond". In: L. G. Bouma and H. Velthuijsen (eds), *Feature Interactions in Telecommunications Systems*, Amsterdam, The Netherlands, May 1994. IOS Press, 1-23. http://www-db.research.bell-labs.com/user/nancyg/benchmark.ps
- [19] Faci, M., Logrippo, L. and Stépien, B. (1989) "Formal Specification of telephone systems in LOTOS", *Protocol Specification, Verification and Testing*, IX, North-Holland.
- [20] Faci, M., Logrippo, L., and Stépien, B. (1991) "Formal Specification of Telephone Systems in LOTOS: The Constraint-Oriented Approach". *Computer Networks and ISDN Systems*, 21 (1991) 53-67. http://lotos.csi.uottawa.ca/ftp/pub/Lotos/Papers/telephone.CNIS9007.ps.Z

- [21] Faci, M. and Logrippo, L. (1994) "Specifying Features and Analysing their Interactions in a LOTOS Environment". In: L. G. Bouma and H. Velthuijsen (eds), Second International Workshop on Feature Interactions in Telecommunications Software Systems, IOS Press, 136-151. http://lotos.csi.uottawa.ca/ftp/pub/Lotos/Papers/Fits94.CameraReady.ps.gz
- [22] Faci, M. (1995) Detecting Feature Interaction in Telecommunications Systems Designs. Ph.D. thesis, Department of Computer Science, University of Ottawa, November 1995. http://lotos.csi.uottawa.ca/ftp/pub/Lotos/Theses/mf\_phd.ps.gz
- [23] Faci, M., Logrippo, L., and Stépien, B (1997) "Structural Models for Telephone Specifications". In: *Computer Network & ISDN Systems*, 29 (1997) 501-528. http://lotos.csi.uottawa.ca/ftp/pub/Lotos/Papers/isdn95.ps.gz
- [24] Griffeth, N.D. and Velthuijsen, N.D. (1994) "The Negotiating Agents Approach to Runtime Feature Interaction Resolution". In: L. G. Bouma and H. Velthuijsen (eds), Second International Workshop on Feature Interactions in Telecommunications Software Systems, IOS press, 217-235. http://www-db.research.bell-labs.com/user/nancyg/fiw94.ps
- [25] Griffeth, N.D., Tadashi, O., Grégoire, J.-C. and Blumenthal, R. (1998) "First Feature Interaction Detection Contest". In: K. Kimbler and W. Bouma (eds.), *Fifth International Workshop on Feature Interactions in Telecommunications Software Systems*, IOS Press, 327-359. http://www.tts.lth.se:80/FIW98/contest.html
- [26] ISO (1989), Information Processing Systems, Open Systems Interconnection, "LOTOS A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", IS 8807.
- [27] ISO/EIC (1991), Information Technology, Open Systems Interconnection, "Conformance Testing Methodology and Framework (CTMF)", IS 9646, ISO, Geneve. Also: CCITT X.290-X.294.
- [28] ITU (1995), Q.1200 (General) Recommendation Series. Geneva.
- [29] ITU (1996), "Recommendation Z. 120: Message Sequence Chart (MSC)". ITU, Geneva.
- [30] Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. (1993) *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison-Wesley, ACM Press.
- [31] Kamoun, J. (1996) Formal Specification and Feature Interaction Detection in the Intelligent Network. M.Sc. thesis, Dept. of Computer Science, University of Ottawa, Ottawa, Canada. http://lotos.csi.uottawa.ca/ftp/pub/Lotos/Theses/jk\_msc.ps.gz
- [32] Kamoun, J. and Logrippo, L. (1998) "Goal-Oriented Feature Interaction Detection in the Intelligent Network Model". In: K. Kimbler and W. Bouma (eds.), *Fifth International Workshop on Feature Interactions in Telecommunications Software Systems*, IOS Press.
- [33] Kimbler, K. and Søbirk, D. (1994) "Use case driven analysis of feature interactions". In: L. G. Bouma and H. Velthuijsen (eds), *Feature Interactions in Telecommunications Systems*, Amsterdam, The Netherlands, May 1994. IOS Press, 167-177.
- [34] Miga, A. (1998) Application of Use Case Maps to System Design with Tool Support. M.Eng. thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada. http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/am\_thesis.pdf
- [35] Petriu, D. (1998) Feature Interaction Detection and Avoidance Smart Design of Telephony System with Use Case Maps. CITO report, Ottawa, Canada. To appear.
- [36] Pavón, S., Larrabeiti, D., and Rabay, G. (1995) *LOLA—User Manual, version 3.6.* DIT, Universidad Politécnica de Madrid, Spain, LOLA/N5/V10 (February).
- [37] Quemada, J., Pavón, S. and Fernández, A. (1988) "Transforming LOTOS Specifications with LOLA: The Parametrized Expansion". In: K. J. Turner (Ed), *Formal Description Techniques, I*, IFIP/North-Holland, 45-54.
- [38] Stépien, B. and Logrippo, L. (1995) "Feature Interaction Detection using Backward Reasoning with LOTOS". In: S. Vuong (ed.), *Protocol Specification, Testing and Verification XIV*, Vancouver, 71-86. http://lotos.csi.uottawa.ca/ftp/pub/Lotos/Papers/pstv.94.book.ps.Z

- [39] Stépien, B. and Logrippo, L. (1995) "Representing and Verifying Intentions in Telephony Features using Absract Data Types". In: K. E. Cheng and T. Ohta (eds.), *Third International Workshop on Feature Interactions in Telecommunications Software Systems*, IOS Press, 141-155. http://lotos.csi.uottawa.ca/~bernard/intention.ps.Z
- [40] Tuok, R. (1996) Modeling and Derivation of Scenarios for a Mobile Telephony System in LOTOS. M.Sc. thesis, Dept. of Computer Science, University of Ottawa, Ottawa, Canada. http://lotos.csi.uottawa.ca/ftp/pub/Lotos/Theses/rt\_msc.ps.gz
- [41] Turner, K.J. (1998) "Validating Architectural Feature Descriptions using LOTOS". In: K. Kimbler and W. Bouma (eds.), *Fifth International Workshop on Feature Interactions in Telecommunications Software Systems*, IOS Press.

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

Created Default & INTL plugins for PreDialStub

Added types Cadence, PIN, Message, TriggerName

ResponseType, Log, LogRecord, Feature, FList \*)

(\* Switch2User: Start AudibleRinging

(\* Switch2User: Start CallWaitingTone

(\* Switch2User: Stop AudibleRinging

(\* Switch2User: Stop CallWaitingTone

(\* NEW: Initialize switch for testing.

\*)

(\* NEW: Allows to guery OS' Log.

(\* Switch2User: Start Ringing

(\* Switch2User: Stop Ringing

Added processes DisplayStub, PreDialStub

Worked on User and Switch for POTS/INTL

Added two complex test processes for POTS.

\*)

\*)

\* )

\*)

\* )

\* )

\*)

\*)

\* )

\*)

\*ĵ

\*)

\*)

\*)

\*)

\* )

\*)

\* )

\* )

\* )

\*)

\*)

\* )

July 6, 1998 : Added types SPList, Status and SCPDB

SINFo, SDB, AddList \*) July 4, 1998 : Added Boolean, adapted Address, and simplified \*)

July 2, 1998 : Created structure and process skeletons.

(\* User2Switch

(\* User2Switch

(\* User2Switch

(\* User2Switch

(\* Switch2User

(\* Switch2User

(\* Switch2Uger

CreateUser, (\* NEW: For creating user instances.

July 5, 1998 : Added process GlobalClock

June 16, 1998: Modified IS8807 ADT.

OnHook .

Dial,

Flash,

DialTone,

StartAR,

StartCWT,

StopAR,

StopCWT,

Announce,

Display,

Init,

Ouerv

]:noexit

(\*-----\*)

(\* Types FBoolean, Element, and Set contain corrections \*)

(\* to the library from the International Standard 8870. \*) (\* Type Boolean remains the same, but NaturalNumber was \*)

(\* simplified by removing unnecessary arithmetic and \*)

\_ implies \_, \_ iff \_, \_ eq \_, \_ ne \_: Bool, Bool -> Bool

x xor y = x and not (y) or (y and not (x));

Modified IS8807 ADT definitions

StopR,

StartR,

NaturalNumbers.

LineBusyTone, (\* Switch2User

Disconnect, (\* Switch2User

Created process PostDialStub

#### A LOTOS SPECIFICATION

Here is the fully commented LOTOS specification derived from our Use Case Maps. It contains the following elements:

- Modification history: lines 1 to 58.
- · Definition of observable gates/events: lines 59 to 79.
- Basic data structures and operations (ADTs simpler than the International Standard's): lines 80 to 227.
- · Data structures and operations for features: lines 228 to 853.
- · Processes representing the components, the stubs and the plugins: lines 854 to 1533.
- · POTS common behaviour: lines 1534 to 1647.
- POTS test cases: lines 1648 to 1714.
- Test cases for individual features: lines 1715 to 2123.
- Test cases for detecting interactions between pairs of features: lines 2124 to 2864.

2         (* Feature Interactions from UCM         *)           3         (* Version: 0.14c         *)           4         (* Date : September 8, 1998         *)           5         (* Authors: Daniel Amyot (damyofesi.uottawa.ca)         *)           6         (* Dorin Petriu (dorin@sce.carleton.ca)         *)           7         (* SCE Department, Carleton University, Ottawa, Canada         *)
3       (* Version: 0.14c       *)         4       (* Date : September 8, 1998       *)         5       (* Authors: Daniel Amyot (damyot@csi.uottawa.ca)       *)         6       (* Dorin Petriu (dorin@sce.carleton.ca)       *)         7       (* SCE Department, Carleton University, Ottawa, Canada       *)         8       (* History:       *)
4         (* Date : September 8, 1998         *)           5         (* Authors: Daniel Amyot (damyot@csi.uottawa.ca)         *)           6         (* Dorin Petriu (dorin@sce.carleton.ca)         *)           7         (* SCE Department, Carleton University, Ottawa, Canada         *)           8         (* History:         *)
5       (* Authors: Daniel Amyot (damyot@csi.uottawa.ca)       *)         6       (* Dorin Petriu (dorin@sce.carleton.ca)       *)         7       (* SCE Department, Carleton University, Ottawa, Canada       *)         8       (* History:       *)
6 (* Dorin Petriu (dorin@sce.carleton.ca) *) 7 (* SCE Department, Carleton University, Ottawa, Canada *) 8 (* History: *)
7 (* SCE Department, Carleton University, Ottawa, Canada *) 8 (* History:
8 (* History: *)
o ( mbcot).
9 (* Sep. 8, 1998 : Added Probes for structural coverage. *)
10 (* Aug. 19, 1998: Fixed part of INFB-TCS interaction in billing. *)
11 (* Solved the INFB-TCS FI in ProcessCallStub. *)
12 (* Aug. 13, 1998: Recomposed TCS as an alternative to INFB. *)
13 (* No FI between TCS and CND, as expected. *)
14 (* FI found between TCS and INFB in this way! *)
15 (* Aug. 12, 1998: TCS implemented and tested. No FI with INTL. *)
16 (* Aug. 11, 1998: New stub for Busy in Post-Dial. Conforms to the *)
17 (* new map with 13 features. *)
18 (* Added support for AirBegin and AirEnd (Cell). *)
19 (* Started implementing TCS (priority over others) *)
20 (* July 12, 1998: Added INFB and 2 tests. The feature works. *)
21 (* Added Query gate to OS. *)
22 (* All test cases now check the final billing Log. *)
23 (* Checking interactions between INTL and INFB. *)
24 (* None found, as expected. *)
25 (* Checking interactions between CND and INFB. *)
26 (* None found, as expected? *)
27 (* July 10, 1998: Added a FList to users. CND now works. *)
28 (* Restructured the test suite *)
29 (* Checks for interactions between CND and INTL. *)
30 (* None found, as expected. *)
31 (* July 9, 1998 : Sets AudibleRinging and Ringing. *)
32 (* Updated SCP. Now INTL works. *)
33 (* Updated the switch. CND almost works. *)
34 (* July 8, 1998 : Specified part of PostDialStub and *)
35 (* ProcessCallStub to make POTS work. It does now. *)
36 (* July 7, 1998 : Added 1t and ge to type Time *)
37 (* Reimplemented SCPDB and query operations *)
38 (* Reimplemented Status and query operations *)
39 (* Created six generic POTS state processes *)
40 (* Created two test processes for POTS and three *)
41 (* for INTL from the Chisel diagrams. *)
42 (* Defined the mechanism for composing feature *)
43 (* plugins in stubs. *)

12 November, 1998 17:47

LOTOS Specification

p. 40

12 November, 1998 17:47

44 (\*

45 (\*

46 (\*

47 (\*

48 (\*

49 (\*

50 (\*

51

52 (\*

53 (\*

54 (\* 55 (\*

56

57 (\*

58

59 60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79 80

81 (\*

82

83 84

85

86 87

88

89

90 91

92

93 opns

94

95

96

97

98 eqns 99

100

101

102 103

104 105

106

107

108

(\*

(\*

specification FI\_UCM[OffHook,

(\* comparison operators

true, false: -> Bool

\_ and \_, \_ or \_, \_ xor \_,

not (true) = false ;
not (false) = true ;

x and true = x ; x and false = false ;

x or true = true ;

x implies y = y or not (x);

x or false = x ;

not: Bool -> Bool

forall x, v: Bool

ofsort Bool

type Boolean is

Bool

sorts

LOTOS Specification

p. 42

12 November, 1998 17:47

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

LOTOS Specification

109 x iff y = x implies y and (y implies x) ; 110 x eq y = x iff y ;111 x ne y = x xor y ; 112 endtype (\* Boolean \*) 113 115 116 type NaturalNumber is Boolean 117 sorts 118 Nat 119 opns 120 0: -> Nat Succ: Nat -> Nat 121 \_ + \_: Nat, Nat -> Nat 122 \_ eq \_, \_ ne \_: Nat, Nat -> Bool 123 124 eqns 125 forall m, n: Nat ofsort Nat 126 127 m + 0 = m ; 128 m + Succ (n) = Succ (m) + n;ofsort Bool 129 130 0 eq 0 = true ; 131 0 eq Succ (m) = false ; 132 Succ (m) eq 0 = false ; 133 Succ (m) eq Succ (n) = m eq n ; 134 m ne n = not (m eq n) ; 135 endtype (\* NaturalNumber \*) 136 138 139 type FBoolean is 140 formalsorts FBool 141 formalopns true : -> FBool : FBool -> FBool 142 not 143 formaleons forall x : FBool 144 145 ofsort FBool not(not(x)) = x;146 147 endtype (\* FBoolean \*) 148 149 150 Element **is** FBoolean 151 type 152 formalsorts Element 153 formalopns \_ eq \_, \_ ne \_ : Element, Element -> FBool 154 formaleqns forall x, y, z : Element 155 156 ofsort Element 157 x eq y = true => 158 х = у ; 159 160 ofsort FBool 161 x = y => 162 x eq y = true : 163 x eq y =true , y eq z = true => 164 x eg z = true ; 165 166 x ne v = not(x eq v); 167 endtype (\* Element \*) 168 169 170 171 type Set is Element, Boolean, NaturalNumber 172 sorts Set 173 opns {} : -> Set

12 November, 1998 17:47

174 Insert, Remove : Element, Set -> Set 175 \_IsIn\_, \_NotIn\_ : Element, Set -> Bool 176 \_Union\_, \_Ints\_, \_Minus\_ : Set, Set -> Set 177 \_eq\_, \_ne\_, \_Includes\_, \_IsSubsetOf\_ : Set, Set -> Bool 178 Card : Set -> Nat 179 180 eqns forall x, y : Element, 181 s,t : Set ofsort Set 182 183 184 x IsIn Insert(y,s) => 185 Insert(x, Insert(y,s)) = Insert(y,s) Remove(x, {}) = {}
Remove(x, Insert(x,s)) = s 186 187 188 x ne y = true of FBool => 189 Remove(x, Insert(y,s)) = Insert(y, Remove(x,s)); 190 191 {} Union s = 9 192 Insert(x,s) Union t = Insert(x,s Union t) ; 193 194 {} Ints s = { } ; 195 x IsIn t => 196 Insert(x,s) Ints t = Insert(x,s Ints t) ; 197 x NotIn t => 198 Insert(x,s) Ints t = s Ints t ; 199 200 s Minus {} = 9 s Minus Insert(x, t) = Remove(x,s) Minus t ; 201 202 203 ofsort Bool 204 x IsIn {} = false 205 x eq y = true of FBool => 206 207 x IsIn Insert(v,s) = true 208 x ne y = true of FBool => x IsIn Insert(y,s) = x IsIn s 209 210 = not(x IsIn s) x NotIn s 211 212 s Includes {} = true s Includes Insert(x,t) = (x IsIn s) and (s Includes t) 213 214 215 s IsSubsetOf t = t Includes s ; 216 217 s eq t = (s Includes t) and (t Includes s); 218 219 s ne t = not(s eq t) ; 220 221 ofsort Nat 222 223 Card({}) = 0 224 x NotIn s => 225 Card(Insert(x,s)) = Succ(Card(s)) 226 endtype (\* Set \*) 227 (\*=======\*) 228 229 (\* FI UCM ADT definitions \*) (\*-----\*) 230 231 (\* The Time type is mapped onto natural numbers. \*) 232 type Timel is NaturalNumber renamedby 233 234 sortnames 235 Time for Nat 236 opnnames 237 tic **for** succ 238 initTime for 0

```
Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features
```

```
239 endtype (* Timel *)
240
241
    (* Additional comparison operators for time range. *)
242
     type Time is Timel
243
    opns
244
        _ lt _, _ ge _ : Time, Time -> Bool
245 egns
        forall t1, t2 : Time
246
247
         ofsort Bool
          tl lt initTime
                           = false ;
248
          initTime lt tic(tl) = true ;
249
250
          tic(t1) lt tic(t2) = t1 lt t2 ;
                            = not (t1 lt t2);
          tl ge t2
251
    endtype (* Time *)
252
253
    254
255
256 (* The Address type contains the Address sort, *)
    (* which is an enumeration of user identifiers *)
257
258
    (* or numbers that can be dialled.
    type Address is NaturalNumber
259
260
    sorts Address
261 opns
262
         userA, userB, userC, anonymous, undefined, star69 : -> Address
263
         zeroPlus : Address -> Address
264
         map : Address -> Nat
265
        dest : Address -> Address
266
        _ eq _, _ ne _ : Address, Address -> Bool
267 eqns
268
        forall user1, user2 : Address
269
        ofsort Nat
270
          map(userA)
                              = 0;
271
          map(userB)
                              = succ(0);
272
          map(userC)
                              = succ(succ(0));
273
          map(anonymous)
                              = succ(succ(succ(0)));
                              = succ(succ(succ(0))); (* for CND *)
274
          map(undefined)
                              = succ(succ(succ(succ(0)))); (* for RC *)
275
          map(star69)
276
          map(zeroPlus(user1)) = succ(succ(succ(succ(succ(0)))))); (* for CC *)
277
        ofsort Address
278
          dest(zeroPlus(user1)) = user1; (* for CC *)
279
         ofsort Bool
280
          user1 eq user2 = map(user1) eq map(user2);
          user1 ne user2 = not(user1 eq user2);
281
282
    endtype (* Address *)
283
284
    (* List of addresses, implemented as a set.
285
    (* We avoid the problem with ISLA's renaming in actualization *)
286
    type AddList0 is Set
287
    actualizedby Address using
288
    sortnames
289
        Address for Element
290
        Bool for FBool
291 endtype (* AddList0 *)
292
    type AddList is AddList0 renamedby
293
294
    sortnames
        AddList for Set
295
296
    opnnames
        NoAddList for {} (* Empty list of addresses *)
297
    endtype (* AddList *)
298
299
    300
301
302
    (* The Cadence is either Ring or SpecialTone. *)
303 type Cadence is Boolean
```

```
304 sorts Cadence
305
    opns
306
         specialRing, tone : -> Cadence
307
        _ eq _, _ ne _ : Cadence, Cadence -> Bool
308 egns
309
        forall cl. c2 : Cadence
310
        ofsort Bool
         specialRing eq specialRing = true;
311
          specialRing eq tone
                                  = false;
312
313
         tone eq specialRing
                                  = false;
314
          tone eq tone
                                  = true;
315
         cl ne c2
                                  = not(c1 eq c2);
316 endtype (* Cadence *)
317
     318
319
320
    (* The PIN is either validPIN or invalidPIN *)
321 type PIN is Cadence renamedby
322 sortnames PIN for Cadence
323 opnnames
        validPIN for tone
324
325
        invalidPIN for specialRing
326
    endtype (* PIN *)
327
    328
329
330
    (* The Message type is mainly for announcements *)
331 type Message is NaturalNumber
332 sorts Message
333 opns
       AskForPIN, displayMessage,
334
        collectedDigits, ScreenedMessage : -> Message
335
336
        map : Message -> Nat
        _ eq _, _ ne _ : Message, Message -> Bool
337
338 eqns
        forall ml. m2 : Message
339
340
        ofsort Nat
          map(AskForPIN) = 0;
341
          map(displayMessage) = succ(0);
342
          map(collectedDigits) = succ(succ(0));
343
344
          map(ScreenedMessage) = succ(succ(succ(0)));
345
          (* Add new messages when necessary *)
346
        ofsort Bool
347
          ml eq m2 = map(m1) eq map(m2);
348
          ml ne m2 = not(ml eq m2);
349
    endtype (* Message *)
350
    351
352
353
   (* The TriggerName sort is an enumeration of *)
354
    (* the names of IN triggers.
                                             *)
355
    type TriggerName is NaturalNumber
356
    sorts TriggerName
357 opns
        ORIGINATION ATTEMPT, INFO COLLECTED, INFO ANALYZED,
358
         NETWORK_BUSY : -> TriggerName
359
        map : TriggerName -> Nat
360
       _ eq _, _ ne _ : TriggerName, TriggerName -> Bool
361
362 ecms
        forall ml, m2 : TriggerName
363
364
        ofsort Nat
          map(ORIGINATION_ATTEMPT) = 0;
365
366
          map(INFO_COLLECTED)
                               = succ(0);
367
          map(INFO_ANALYZED)
                                 = succ(succ(0));
368
          map(NETWORK_BUSY)
                                = succ(succ(succ(0)));
```

12 November, 1998 17:47

Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

	Use case maps for the Design and the validation of interaction-1 ree receptiony i
369	ofsort Bool
370	ml eq m2 = map(ml) eq map(m2);
371	ml ne m2 = not(ml eq m2);
372	endtype (* TriggerName *)
373	·····
374	(**************************************
375	(* The Demonstration of the second seco
370	(* the Kesponselype sort is an enumeration of *)
370	(" Lie Expression of Nature Number
370	sorts Responsetive is Natural Number
380	
381	ANALYZE ROITE CONTINUE FORMARD CALL SEND TO RESOURCE
382	DISCONNECT : -> ResponseType
383	map : ResponseType -> Nat
384	eq , ne : ResponseType, ResponseType -> Bool
385	eqns
386	forall m1, m2 : ResponseType
387	ofsort Nat
388	map(ANALYZE_ROUTE) = 0;
389	map(CONTINUE) = succ(0);
390	<pre>map(FORWARD_CALL) = succ(succ(0));</pre>
391	<pre>map(SEND_TO_RESOURCE) = succ(succ(0)));</pre>
392	<pre>map(DISCONNECT) = succ(succ(succ(0))));</pre>
393	ofsort Bool
394	ml eq m2 = map(m1) eq map(m2);
395	<pre>ml ne m2 = not(ml eq m2);</pre>
396	endtype (* ResponseType *)
397	
398	(**************************************
399	
400	(* The type of log in the US is Begin, End, AirBegin, or AirEnd *)
401	type Logiype is NaturalNumber
402	Solids Logitype
403	Degin End AirBegin AirEnd> LogTime
405	man : LogType -> Nat
406	eq . ne : LogType . JogType -> Bool
407	egns
408	forall m1, m2 : LogType
409	ofsort Nat
410	<pre>map(Begin) = 0;</pre>
411	<pre>map(End) = succ(0);</pre>
412	<pre>map(AirBegin) = succ(succ(0));</pre>
413	<pre>map(AirEnd) = succ(succ(0)));</pre>
414	ofsort Bool
415	ml eq m2 = map(m1) eq map(m2);
416	ml ne m2 = not(ml eq m2);
417	endtype (* LogType *)
418	
419	(* A record for the Log. *)
420	(* can be i(Begin, X, Y, F)) or i(End, X, Y, Underlined, I) for regular logs *)
421	(* And 1(ArrBegIn, X, underIned, underIned, 1) or ")
422	(* (Artishdy, Anderinder, Inderinder, 1) for certain rogs)
424	sorts LogRecord
425	
426	1 : LogTvpe, Address, Address, Address, Time -> LogRecord
427	eg , ne : LogRecord, LogRecord -> Bool
428	eqns
429	forall X1, X2, Y1, Y2, P1, P2 : Address,
430	T1, T2 : Time,
431	LT1, LT2 : LogType
432	ofsort Bool
433	(LT1 eq LT2) and (X1 eq X2) and (Y1 eq Y2) and (P1 eq P2) and (T1 eq T2) =>

434 l(LT1,X1,Y1,P1,T1) eq l(LT2,X2,Y2,P2,T2) = true; 435 not((LT1 eq LT2) and (X1 eq X2) and (Y1 eq Y2) and (P1 eq P2) and (T1 eq T2)) => 436 l(LT1,X1,Y1,P1,T1) eq l(LT2,X2,Y2,P2,T2) = false; 437 l(LT1,X1,Y1,P1,T1) ne l(LT2,X2,Y2,P2,T2) = not(l(LT1,X1,Y1,P1,T1) eq l(LT2,X2,Y2,P2,T2)); 438 endtype (\* LogRecord \*) 439 440 (\* List of log records (logs), implemented as a set. \*) (\* We avoid the problem with ISLA's renaming in actualization \*) 441 442 type Log0 is Set 443 actualizedby LogRecord using 444 sortnames 445 LogRecord for Element 446 Bool for FBool 447 endtype (\* Logs0 \*) 448 type Log is Log0 renamedby 449 450 sortnames Log **for** Set 451 452 opnnames NoLog for {} (\* Empty list of log records \*) 453 454 endtype (\* Log \*) 455 456 457 458 (\* The Feature sort is an enumeration of the \*) 459 (\* features to which users can subscribe, \*) 460 (\* including POTS. \*) 461 type Feature is NaturalNumber 462 sorts Feature 463 opns POTS, (\* Plain Old Telephone System 464 \* ) CFBL, (\* Call Forward Busy Line \*ĵ 465 CND, (\* Call Name Delivery 466 \*) INFB, (\* IN Freephone Billing 467 \*) INFR, (\* IN Freephone Routing 468 \*) INTL, (\* IN Teen Line 469 \*) 470 TCS, (\* Terminating Call Screening 471 3WC, (\* Three-way Calling \*) INCF, (\* IN Call Forwarding 472 CW, (\* Call Waiting CC, (\* Charge Call 473 \*) 474 475 (\* Phase II features, plus one more. \*) 476 Cell, (\* Cellular \* 1 477 RC, (\* Return Call \*1 478 ACB (\* Automatic Call Back (Dorin's) \*) : -> Feature 479 480 map : Feature -> Nat 481 \_ eq \_, \_ ne \_ : Feature, Feature -> Bool 482 eqns 483 forall m1, m2 : Feature 484 ofsort Nat 485 map(POTS) = 0;486 map(CFBL) = succ(0); 487 map(CND) = succ(succ(0)); map(INFR) = succ(succ(succ(0))); 488 489 map(INFB) = succ(succ(succ(succ(0)))); 490 map(INTL) = succ(succ(succ(succ(0)))); map(TCS) = succ(succ(succ(succ(succ(0))))); 491 map(3WC) = succ(succ(succ(succ(succ(0)))))); 492 map(INCF) = succ(succ(succ(succ(succ(succ(0))))))); 493 map(CW) = succ(succ(succ(succ(succ(succ(succ(o))))))))); 494 495 496 497 498 

12 November, 1998 17:47

LOTOS Specification

12 November, 1998 17:47

Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

564

p. 48

499 ofsort Bool 500 ml eq m2 = map(m1) eq map(m2);501 ml ne m2 = not(ml eq m2); 502 endtype (\* Feature \*) 503 504 (\* List of features, implemented as a set. 505 (\* We avoid the problem with ISLA's renaming in actualization \*) type Fligt() is Set 506 507 actualizedby Feature using 508 sortnames 509 Feature for Element 510 Bool for FBool endtype (\* Logs0 \*) 511 512 type Flist is Flist0 renamedby 513 514 sortnames 515 Flist for Set 516 opnnames NoFList for {} (\* Empty list of features \*) 517 endtype (\* Flist \*) 518 519 520 521 522 (\* A record for the subscriber information. 523 (\* Format: sub(userID, Features, BLForward, LastIncoming, \*) 524 (\* Screened, ChargePin) 525 type SInfo is AddList, FList, PIN 526 sorts SInfo 527 opns (\* User identifier 528 sub : Address \*) (\* List of subscribed features \*) 529 FList (\* BLForward, for CFBL 530 Address, \*) Address, (\* LastIncoming, for CND 531 \*) AddList, (\* Screened list, for TCS 532 \*) 533 PTN (\* Charge PIN, for CC \*) -> SInfo \_ eq \_, \_ ne \_ : SInfo, SInfo -> Bool 534 535 eqns forall s1, s2, b11, b12, li1, li2: Address, 536 fl1, fl2: Flist, 537 538 sll, sl2: AddList, 539 pl, p2: PIN 540 ofsort Bool (sl eq s2) and (fll eq fl2) and (bll eq bl2) and (lil eq li2) 541 542 and (sll eq sl2) and (pl eq p2) => 543 sub(s1, f11, b11, li1, s11, p1) eq sub(s2, f12, b12, li2, s12, p2) = true; 544 not((sl eq s2) and (fll eq fl2) and (bll eq bl2) and (lil eq li2)545 and (sll eq sl2) and (pl eq p2)) => 546 sub(s1, f11, b11, li1, s11, p1) eq sub(s2, f12, b12, li2, s12, p2) = false; 547 sub(s1, f11, b11, li1, s11, p1) ne sub(s2, f12, b12, li2, s12, p2) = 548 not(sub(s1, f11, b11, li1, s11, p1) eq sub(s2, f12, b12, li2, s12, p2)); 549 endtype (\* SInfo \*) 550 551 (\* Database of subscriber records (SInfo), implemented as a set. \*) 552 (\* We avoid the problem with ISLA's renaming in actualization. \*) 553 type SDB0 is Set 554 actualizedby SInfo using 555 sortnames SInfo for Element 556 Bool for FBool 557 endtype (\* SDB0 \*) 558 559 type SDB1 is SDB0 renamedby 560 561 sortnames 562 SDB for Set 563 opnnames

LOTOS Specification

12 November, 1998 17:47

```
NoSDB for {} (* Empty list of subscribers *)
565 endtype (* SDB1 *)
566
567
     (* Query operators *)
568
     type SDB is SDB1
569
     opns
570
         (* Tells whether a subscriber has subscribed a particular feature *)
571
         has : Address Feature SDB -> Bool
         (* Sets/Gets the$LastIncoming caller *)
572
         setLastIncoming : Address, Address, SDB -> SDB
573
574
         getLastIncoming : Address, SDB -> Address
575
         (* Check whether the caller party is on the callee's TCS *)
576
         isOnTCS : Address, Address, SDB -> Bool (* Caller, Callee *)
577
578
     eqns
579
         forall s1, s2, s3, b11, li1, li2: Address,
580
               sll: AddList,
               pl: PIN,
581
582
               f1, f2: Feature,
583
               fl : FList,
584
                sdb
                     : SDB
585
         ofsort Bool
           has(s1, f1, NoSDB) = false;
586
587
           s1 eq s2 =>
588
             has(s1, f1, Insert(sub(s2,f1,b11,li1,s11,p1), sdb)) = f1 IsIn f1;
           s1 ne s2 =>
589
590
             has(s1, f1, Insert(sub(s2,f1,b11,li1,s11,p1), sdb)) = has(s1, f1, sdb);
591
592
           isOnTCS(s1, s2, NoSDB) = false;
593
           s3 eg s2 =>
            isOnTCS(s1, s2, Insert(sub(s3,f1,b11,li1,s11,p1), sdb)) = s1 IsIn s11;
594
595
           s3 ne s2 =>
596
             isOnTCS(s1, s2, Insert(sub(s3,f1,b11,li1,s11,p1), sdb)) = isOnTCS(s1, s2, sdb);
597
598
         ofsort SDB
           setLastIncoming(s1, li1, NoSDB) = NoSDB;
599
600
           s1 eg s2 =>
601
             setLastIncoming(s1, li1, Insert(sub(s2,f1,bl1,li2,sl1,p1), sdb)) =
                                                 Insert(sub(s2,f1,bl1,li1,sl1,p1), sdb);
602
           s1 ne s2 =>
603
             setLastIncoming(s1, li1, Insert(sub(s2,f1,bl1,li2,sl1,p1), sdb)) =
604
               Insert(sub(s2,f1,bl1,li2,sl1,pl), setLastIncoming(s1, li1,sdb));
         ofsort Address
605
606
           getLastIncoming(s1, NoSDB) = undefined;
607
           s1 eg s2 =>
608
             getLastIncoming(s1, Insert(sub(s2,f1,bl1,li1,s11,p1), sdb)) = li1;
609
           s1 ne s2 =>
             getLastIncoming(sl, Insert(sub(s2,fl,bl1,li1,sl1,pl), sdb)) = getLastIncoming(sl, sdb);
610
611
     endtype (* SDB *)
612
     613
614
615 (* The SCPit sort is an enumeration of the *)
     (* SCP types of information in the database. *)
616
     type SCPit is NaturalNumber
617
618
    sorts SCPit
619 opns
         Redirect, TeenPIN, TeenTime, ForwardedTo : -> SCPit
620
         map : SCPit -> Nat
621
         _ eq _, _ ne _ : SCPit, SCPit -> Bool
622
623 eqns
         forall s1, s2 : SCPit
624
625
         ofsort Nat
626
           map(Redirect) = 0;
627
           map(TeenPIN)
                           = succ(0);
12 November, 1998 17:47
                                               LOTOS Specification
```

```
Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features
```

```
628
            map(TeenTime) = succ(succ(0));
629
            map(ForwardedTo) = succ(succ(0)));
630
          ofsort Bool
631
           s1 eq s2 = map(s1) eq map(s2);
632
           s1 ne s2 = not(s1 eg s2);
633 endtype (* SCPit *)
634
635 (* Information records about the feature parameters in the SCP *)
     (* These heterogeneous records share the same format to simplify *)
636
637
     (* the equations.
     type SCPinfo is SCPit, Address, Time, PIN
638
639
     sorts SCPinfo
640
     opns
         (* INFR: Redirect A B T1 T2 C
                                                                                      *)
641
642
                  -> scp(Redirect, A, B, T1, T2, C, validPIN)
                                                                                      *)
          (* INTL: TeenPIN A PIN
643
644
                  -> scp(TeenPIN, A, undefined, initTime, initTime, undefined, PIN)
                                                                                      *)
          (* INTL: TeenTime A T1 T2
645
                  -> scp(TeenTime, A, undefined, T1, T2, undefined, validPIN)
                                                                                      *)
646
          (* INCF: ForwardedTo B C
647
648
                  -> scp(ForwardedTo, undefined, B, initTime, initTime, C, validPIN) *)
649
         scp : SCPit, Address, Address, Time, Time, Address, PIN -> SCPinfo
         _ eq _, _ ne _ : SCPinfo, SCPinfo -> Bool
650
651
     eans
652
         forall s1, s2
                                      : SCPit,
653
                al, a2, b1, b2, c1, c2: Address,
654
                t11, t12, t21, t22 : Time,
655
                pin1, pin2
                                     : DIN
656
         ofsort bool
657
         (sl eq s2) and (al eq a2) and (bl eq b2) and (cl eq c2) and (tll eq tl2) and (t2l eq t22) and
                                                   (pinl eq pin2) =>
           scp(s1, a1, b1, t11, t21, c1, pin1) eq scp(s2, a2, b2, t12, t22, c2, pin2) = true;
658
         not((sl eq s2) and (al eq a2) and (bl eq b2) and (cl eq c2) and (tll eq t12) and (t2l eq t22)
659
                                                   and (pin1 eg pin2)) =>
660
           scp(s1, a1, b1, t11, t21, c1, pin1) eq scp(s2, a2, b2, t12, t22, c2, pin2) = false;
         scp(sl, al, bl, tll, t21, cl, pinl) ne scp(s2, a2, b2, tl2, t22, c2, pin2) =
not(scp(sl, al, bl, tll, t21, cl, pinl) eq scp(s2, a2, b2, tl2, t22, c2, pin2))
661
662
     endtype (* SCPinfo *)
663
664
665
     (* Database of feature parameters (SCPinfo) in the SCP, implemented as a set. *)
666
     (* We avoid the problem with ISLA's renaming in actualization. *)
667
     type SCPDB0 is Set
     actualizedby SCPinfo using
668
669
     sortnames
670
         SCPinfo for Element
671
         Bool for FBool
     endtype (* SCPDB0 *)
672
673
674
     type SCPDB1 is SCPDB0 renamedby
675
     sortnames
676
         SCEDE for Set
     opnnames
677
         NoSCPDB for {} (* Empty list of feature parameters. *)
678
     endtype (* SCPDB1 *)
679
680
    (* Ouerv operators *)
681
     type SCPDB is SCPDB1
682
683
    opns
         (* Tells whether this is an INTL restricted time or not *)
684
         IsInTeenTime : Address, Time, SCPDB -> Bool
685
686
         IsValidTeenPIN : Address, PIN, SCPDB -> Bool
687
     eqns
688
         forall scpit
                             : SCPit,
689
                al, a2, b, c : Address,
690
                t, t1, t2 : Time,
```

```
691
                p, p1, p2 : PIN,
692
                scpdb
                             : SCPDB
693
         ofsort Bool
694
         (* IsInTeenTime *)
695
           IsInTeenTime(al, t, NoSCPDB) = false;
696
           (scpit eq TeenTime) and (al eq a2) and (t ge t1) and (t lt t2) =>
697
             IsInTeenTime(al, t, Insert(scp(scpit, a2, b, t1, t2, c, p), scpdb)) = true;
698
           not((scpit eg TeenTime) and (al eg a2) and (t ge t1) and (t lt t2)) =>
             IsInTeenTime(al, t, Insert(scp(scpit, a2, b, t1, t2, c, p), scpdb)) =
699
700
                 IsInTeenTime(al. t. scpdb);
701
702
         (* IsValidTeenPIN *)
           IsValidTeenPIN(al, pl, NoSCPDB) = false;
703
704
           (scpit eq TeenPIN) and (al eq a2) and (pl eq p2) =>
705
             IsValidTeenPIN(al, pl, Insert(scp(scpit, a2, b, t1, t2, c, p2), scpdb)) = true;
706
           not((scpit eq TeenPIN) and (al eq a2) and (pl eq p2)) =>
707
             IsValidTeenPIN(al, pl, Insert(scp(scpit, a2, b, t1, t2, c, p2), scpdb)) =
708
                 IsValidTeenPIN(al, pl, scpdb);
709
     endtype (* SCPDB *)
710
     711
712
713
     (* The StatItem sort is an enumeration of the *)
714
     (* status items in the switch in the database. *)
715
     type StatItem is NaturalNumber
716
     sorts StatItem
717
     opns
718
         Busy, Ringing, AudibleRinging, ThreeWay, CallWaiting : -> StatItem
719
         map : StatItem -> Nat
         _ eq _, _ ne _ : StatItem, StatItem -> Bool
720
721 egns
         forall s1, s2 : StatItem
722
723
         ofsort Nat
724
           map(Busv)
                              = 0;
725
           map(Ringing)
                              = \operatorname{succ}(0);
           map(AudibleRinging) = succ(succ(0));
726
          map(ThreeWay) = succ(succ(succ(0)));
map(CallWaiting) = succ(succ(succ(succ(0))));
727
728
         ofsort Bool
729
730
           s1 eq s2 = map(s1) eq map(s2);
731
           s1 ne s2 = not(s1 eq s2);
732
     endtype (* StatItem *)
733
734
     (* Status records collected in the switch during calls *)
735
     type Stat is Address, StatItem
736
     sorts Stat
737
     opns
738
        (* POTS: Busy A
                                    -> stat(Busy, A, undefined)
                                                                        * 1
739
        (* POTS: Ringing A B
                                    -> stat(Rigning, A, B)
                                                                        *)
740
         (* POTS: AudibleRinging A B -> stat(AudibleRinging, A, B)
                                                                        *)
741
         (* 3WC : ThreeWay X
                                    -> stat(ThreeWay, X, undefined)
                                                                        *)
742
         (* CW : CallWaiting X
                                    -> stat(CallWaiting, X, undefined) *)
         stat : StatItem, Address, Address -> Stat
743
744
         _ eq _, _ ne _ : Stat, Stat -> Bool
745 egns
746
         forall al, a2, b1, b2: Address.
747
               sil, si2: StatItem
         ofsort Bool
748
           (a) eq a2) and (b) eq b2) and (sil eq si2) =>
749
             stat(sil, al, bl) eq stat(si2, a2, b2) = true;
750
           not((al eq a2) and (bl eq b2) and (sil eq si2)) =>
751
            stat(sil, al, bl) eq stat(si2, a2, b2) = false;
752
753
           stat(sil, al, bl) ne stat(si2, a2, b2) =
754
            not(stat(sil, al, bl) eq stat(si2, a2, b2));
755
     endtype (* Stat *)
```

LOTOS Specification

p. 50

12 November, 1998 17:47

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

756 757 (\* Database of status records in the switch, implemented as a set. \*) (\* We avoid the problem with ISLA's renaming in actualization. \*) 758 type Status0 is Set 759 760 actualizedby Stat using 761 sortnames Stat for Element 762 763 Bool for FBool 764 endtype (\* Status0 \*) 765 type Status1 is Status0 renamedby 766 767 sortnames 768 Status for Set 769 opnnames 770 NoStatus for {} (\* Empty list of status. \*) 771 endtype (\* Status1 \*) 772 773 (\* Query operators \*) 774 type Status is Statusl 775 opns 776 (\* Tells whether a subscriber is Idle or Busy \*) 777 isIdle, isBusy : Address, Status -> Bool 778 eqns 779 forall al, a2, b1, b2 : Address, 780 sil, si2 : StatItem, 781 : Status G 782 ofsort Bool 783 (\* isIdle \*) 784 isIdle(al, NoStatus) = true; 785 (al eg a2) and (si2 eg Busy) => isIdle(al, Insert(stat(si2, a2, b2), s)) = false; 786 not((al eg a2) and (si2 eg Busy)) => 787 isIdle(al, Insert(stat(si2, a2, b2), s)) = isIdle(al, s); 788 789 (\* isBusv \*) 790 isBusy(al, s) = not(isIdle(al, s)); 791 endtype (\* Status \*) 792 (\*========\*) 793 Stub Path ADT definitions 794 (\* 795 (\*======\*) 796 797 (\* Entry and exit points of each stub in the maps \*) type StubPath is NaturalNumber 798 799 sorts StubPath 800 opns 801 inPreD1, outPreD1, outPreD2, (\* pre-dial stub \*) 802 inPostD1, outPostD1, outPostD2, outPostD3, outPostD4, 803 outPostD5, (\* post-dial stub inBill1, outBill2. 804 (\* billing stub \*) 805 inPC1, outPC1, outPC2, outPC3, (\* process-call stub \*) 806 outPC4, 807 inDispl, outDispl, (\* displav stub \*) inBusyl, outBusyl, outBusy2 808 (\* busy stub \*) : -> StubPath 809 map : StubPath -> Nat \_ eq \_, \_ ne \_ : StubPath, StubPath -> Bool 810 811 egns forall spl, sp2 : StubPath 812 813 ofsort Nat \*) = 0; 814 map(inPreD1) (\* From OffHook map(outPreD1) (\* To Dial \*) = succ(map(inPreD1)); 815 map(outPreD2) (\* To Reject \*) = succ(map(outPreD1)); 816 map(inPostD1) (\* From Dial \*) = succ(map(outPreD2)); 817 818 map(outPostD1) (\* To Term-Connected \*) = succ(map(inPostD1)); map(outPostD2) (\* To Orig-Connected \*) = succ(map(outPostD1)); 819 820 map(outPostD3) (\* To Billing \*) = succ(map(outPostD2));

```
821
           map(outPostD4) (* To Reject
                                            *) = succ(map(outPostD3));
822
           map(outPostD5) (* To Busy
                                             *) = succ(map(outPostD4));
823
           map(inBill1) (* From Post-Dial
                                            *) = succ(man(outPostD5));
824
          map(outBill2) (* To Result-OS
                                            *) = succ(map(inBill1));
          map(inPC1)
825
                        (* From Call
                                             *) = succ(map(outBill2));
826
          map(outPC1)
                        (* To Ring (Term)
                                            *) = succ(map(inPC1));
          map(outPC2)
827
                        (* To Busv
                                            *) = succ(map(outPC1));
                                             *) = succ(map(out PC2));
          map(outPC3)
                        (* To Reject
828
                        (* To stub itself *) = succ(map(outPC3));
829
          map(outPC4)
          map(inDispl) (* From PC stub
                                             *) = succ(map(outPC4));
830
831
          map(outDispl) (* To OffHook
                                             *) = succ(map(inDispl));
832
          map(inBusy1) (* From Process-Call *) = succ(map(outDisp1));
          map(outBusy1) (* To Busy
                                            *) = succ(map(inBusv1));
833
          map(outBusy2) (* To Call_X
                                             *) = succ(map(outBusy1));
834
835
         ofsort Bool
          spl eq sp2 = map(sp1) eq map(sp2);
836
          spl ne sp2 = not(spl eq sp2);
837
838 endtype (* StubPath *)
839
840 type SPList0 is Set
     actualizedby StubPath using
841
842
     sortnames
        StubPath for Element
843
844
        Bool for FBool
845
    endtype (* SPList0 *)
846
847
     type SPList is SPList0 renamedby
848
    sortnames
849
        Spligt for Set
850
    oppnames
        NoSPList for {} (* Empty list of path identifiers. *)
851
852
     endtype (* SPList *)
853
854
     (*========*)
855
     (* Behaviour Description
856
     (*-----*)
857
858
     behaviour
859
860
       (* Gates not visible to the users are set to be internal. *)
861
       (* Interfaces (e.g. Switch2User) are splitted into several*)
862
       (* gates, one per type of message.
       hide Trigger, (* Switch2SCP *)
863
864
           Resource, (* Switch2SCP *)
865
           Response, (* SCP2Switch *)
866
           LogBegin, (* 20S
                                  * 1
867
           LogEnd, (* 20S
                                  *1
868
           AirBegin, (* 20S
                                  *)
869
           AirEnd, (* 20S
                                  *)
870
           Time
                    (* NEW: Used by the Switch to get the time *)
871
       in
872
873
       (* Get the Initial state from the environment *)
       Init 2InitSDB:SDB 2InitStatus:Status 2InitSCODB:SCODB 2currentTime:Time:
874
875
876
         (* We create as many users as necessary, *)
         UserFactory [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
877
                    StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
878
                    Disconnect, Display, CreateUser]
879
         [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR.
880
881
          StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
882
          Disconnect, Display]
883
         (
884
885
            GlobalClock [Time](currentTime)
```

12 November, 1998 17:47

LOTOS Specification

p. 52

12 November, 1998 17:47

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

886	[[Time]]
887	Switch [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
888	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
889	Disconnect, Display, Trigger, Resource, Response, LogBegin,
890	LogEnd, AirBegin, AirEnd, Time (InitSDB, InitStatus)
891	[Trigger Resource Response]
002	[[III]] [Trigger, Resource, Responder LegDegin, LegEnd AirDegin, AirEnd]/InitCODDD)
092	SCP [IIIgger, Resource, Response, Logaegin, Logand, Airbegin, Airbid](IIIISCPDB)
893	
894	[LogBegin, LogEnd, AirBegin, AirEnd]
895	OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](NoLog)
896	)
897	)
898	
899	where
900	
901	/*************
001	
902	(" Process UserFactory, To create and initialise necessary users . ")
903	(**************************************
904	
905	process UserFactory [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
906	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
907	Disconnect, Display, CreateUser]: noexit :=
908	CreateUser ?userId:Address ?userFeatures:FList;
909	
910	(* Create the user *)
911	Neer (OffHook OnHook Dial Flash DialTone Startle Starte
012	Charteful Charteful Charteful Charteful Charteful Charteful Charteful
012	Disconst, Diopar, Stopar, Stopar, Linebusytone, Amounce,
913	Disconnect, Displays (userid, userreatures)
914	
915	(* Prepare to accept new creation request *)
916	UserFactory [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
917	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
918	Disconnect, Display, CreateUser]
919	)
920	endproc (* UserFactory *)
921	
922	
023	/*************
024	(* Process Rear: To be instantiated by all users with a userId *)
0.25	( FICESS User, To be instantiated by all users with a useria. )
925	()
926	
927	process User [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
928	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
929	Disconnect, Display] (userId: Address, uf:FList): <b>noexit</b> :=
930	
931	(* POTS - Origination (Root map) *)
932	OffHook !userId; (*_PROBE_*)
933	User [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
934	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce.
935	Disconnect Displayl (userId uf)
000	Disconnece, Display, (abelia, ar)
930	
937	
938	DialTone !userId;
939	
940	Dial !userId ?userTo:Address;
941	User [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
942	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
943	Disconnect, Display] (userId, uf)
944	
945	OnHook luserId; (* PROBE *) stop
946	
047	,
049	
248	LJ
549	
200	LineBusyTone (useriu)

LOTOS Specification

p. 54

12 November, 1998 17:47

12 November, 1998 17:47

951 OnHook !userId; (\*\_PROBE\_\*) stop 952 953 [] 954 955 (\* POTS - Origination (post-dial default map) \*) StartAR !userId ?userTo:Address; 956 957 958 OnHook !userId; 959 StopAR !userId !userTo; (\*\_PROBE\_\*) stop 960 F 1 961 StopAR !userId !userTo; (\* CONNECTED state! Use Disconnect map. \*) 962 963 964 OnHook !userId; (\*\_PROBE\_\*) stop 965 [] 966 Disconnect !userId !userTo; 967 OnHook !userId; (\*\_PROBE\_\*) stop 968 ) 969 ) 970 971 [] 972 973 (\* POTS - Termination (post-dial default map) \*) 974 975 976 977 StartR !userId ?userFrom:Address; exit(userId, userFrom, uf, any SPList) 978 DisplayStub[Display](userId, uf, Insert(inDispl, NoSPList)) 979 980 ) 981 >> 982 accept userId:Address, userFrom:Address, uf:FList, outPaths:SPList in (\* outDispl is the only possible outPath... \*) 983 984 [outDispl IsIn outPaths] -> 985 986 OffHook !userId; 987 StopR !userId !userFrom; 988 (\* CONNECTED state! Use Disconnect map) \*) 989 990 Disconnect !userId !UserFrom; 991 OnHook !userId; (\*\_PROBE\_\*) stop 992 [] 993 OnHook !userId; (\*\_PROBE\_\*) stop 994 ) 995 [] 996 (\* userFrom has gon on-hook. \*) StopR !userId !userFrom; (\*\_PROBE\_\*) stop 997 998 1 999 ) 1000 [] 1001 1002 1003 (\* INTL - Origination (pre-dial INTL map) \*) Announce !userId !AskForPIN; 1004 1005 Dial !userId ?p:PIN; (\*\_PROBE\_\*) 1006 1007 User [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1008 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1009 Disconnect, Display] (userId, uf) [] 1010 OnHook !userId; (\*\_PROBE\_\*) stop 1011 1012 1013 1014 [] 1015

LOTOS Specification

p. 55

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

1016	Announce  userId  InvalIdPIN;	1000	(tor disconnection purposes). *)
1017	UnHook !useria, (*_PROBE_*) stop	1080	(* Paths 4, 5, and {1,2,3} are alternative paths. *)
1018		1081	[outPostD1 Isin outPaths] ->
1019	11	1082	(t user From may have disconnected *)
1020	(* TCS reject *)	1084	(" user from may have disconnected")
1021	( TED TEJECE / Amounta Lucaria / ScreenedMassace:	1085	Time 2t-time:
1022	Onlock lucerid: (* DPDP *) etco	1085	IorDecin LucerErom LucerEo LucerEau It:
1023	SIMOK ABELLA (KOB ) SCOP	1087	Independent inservice inservery it?
1024	(* NOT TO ECOPET: Flash may be interpreted as Onlock (filesk *)	1089	(* Orig Connected and Orig disconnects (DOTC 7) *)
1025	(" NOT TO FORGET. Flash may be interpreted as Onnook-Orthook ")	1088	(- org-connected, and org disconnects (POIS /) -)
1020	where	1000	
1029	WIGLE	1090	OnBook LugerFrom:
1020	/*********	1092	
1020	(* Stub Process DisplayStub: in map Post-Dial One input nath and *)	1092	let status Status - Remove(stat(Rusy userFrom undefined) status) in
1031	(*	1094	(
1032	( // // // // // // // // // // // // //	1095	Disconnect LuserTrom:
1032	process DisplayStub[Display]/userId/Address uf Flist inDaths SDDist) .	1096	evit(userFrom userTo sch status)
1034	evit/Advecs DigrayStur(Display)(useria/Advecs DigrayStur(Display)) -	1097	li
1035	(* In this stub (NND is optional (we do not have access to SCDDD) *)	1098	Time 2t time:
1036	( In this study, the is optimized if we do not have access to Stridy, $($ ) (CNN horth will $= x/x$ (2014) the a probe here (here $12-1061D74DR$ instead *)	1090	LogPad LucerProm LucerTo It:
1037	(* DOTS default plugin *)	1100	avit (user from user to sdb status)
1039	( FOIS default program / Theart(outDign) NoSDIg())	1100	ente (userrion, userro, sub, status)
1030	()	1101	/
1039		1102	Orlight user re.
1040	(UND ISIN UL	1104	
1042	Display Lucar To 2000 Promit Advance: (* DPORF *)	1104	lat status Status - Remove/stat/Busy userTo undefined) status) in
1042	evid userial and Marses of Insert(outDiss) NoCOList))	1105	(* DODE *)
1045	entrustin, any Andress, ur, insert(outbisg), Nosenist))	1106	
1045		1100	
1045	endproc (* liker *)	1108	
1040	enquice ( user )	1109	
1048		1110	n'
1049	/**************************************	1111	(* Term-Connected and Term disconnects (POTS 10) *)
1050	(* Process Switch: *)	1112	[outDestD3_ISIN_outDateds
1050	(* Process Switch, */	1112	[OULPSLDS ISIN OULPSLNS] ->
1052	nrogage Switch (Offlook Onlook Dial Flach DialTone StartAP StartP	1114	OnWook LugerTo:
1052	Startful Control Cond Charles International Annunce	1115	
1054	Discourse Discusser Friday Printer Resource Resource Lordagin	1115	let status Status - Remove(stat(Rusy userTo undefined) status) in
1055	Lagra AirBean AirBad Timal (ab. CD status (tatus)	1117	/
1056	: noatt :=	1118	Disconnect LucerFrom LucerTo:
1057		1119	evit(userFrom userTo sdb status)
1058	OffHook SuserFrom: Eddress [isIdle(userFrom status)]:	1120	
1059	DreDialStub[OnHook Trigger Response Resource Announce Dial Time]	1121	Time 2t:time:
1060	(Incert(introl) NoSpinst), userFrom sch	1122	LogEnd luserFrom luserTo It:
1061	(* Set userFirm Bigg *)	1123	evit (user From user To sdb status)
1062	Theert(stat(Busy userFrom undefined) status))	1124	
1063	>>	1125	> accept userFrom:Address, userTo:Address, sdb:SDB, status:Status in
1064	accept userFrom:Address, sdb:SDB, status:Status, outPaths:SPList in	1126	OnHook luserFrom;
1065	(* outPreD1: 0k *)	1127	
1066	(outPrebl IsTo outPaths) ->	1128	let status:Status = Remove/stat(Rusy userFrom undefined) status)
1067		1120	in (* DROBE *)
1068	DialTone LuserFrom:	1129	stop
1069		1130	
1070		1131	
1071	Dial LucerFrom 2ucerTo:Eddress:	1132	
1072	PostDialStub(OffHook Onlock Dial Flagh DialTone StartAP	1133	
1073	Start Start Start Start Store Store Store	1134	
1074	LineBusyTone Bnounce Disconnect Display	1135	[TeIdle(userFrom status)] ->
1075	Trigger Decourse Desponse Loperajn LogErd	1126	(isine(uselfioii, status)) ->
1076	Ting/Tingert(inDerti) Norther Joyney How Marker and Atorica	1127	(* UserFrom bes gone on-book *) (* PPOPE *)
1077	Time;(insert(infostbi, Moshist), dseriom, dserio, Sdb, Status)	1120	( USEFFOR has gone on-hook/ ("_FRODL_"/
1079		1120	scop
10/8	accept userriom.Address, userrio.Address, userray.Address, sdb.SDB, status.status, outPat	1140	
1070	is office interview of the second sec	1141	, []
10/2	Courput path i comes before 2,5, which can work as alternatives	1141	t j

p. 56

12 November, 1998 17:47

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

1142 [outPostD4 IsIn outPaths] -> 1143 1144 (\* Reject path. \*) 1145 Announce !userFrom !ScreenedMessage 1146 OnHook !userFrom; 1147 1148 let status:Status = Remove(stat(Busy, userFrom, undefined), status) in (\* PROBE \*) 1149 stop 1150 ) 1151 1152 [] [outPostD5 IsIn outPaths] -> 1154 (\* Busy. (POTS 15) \*) 1155 1156 LineBusyTone !userFrom; 1157 OnHook !userFrom; 1158 let status:Status = Remove(stat(Busy, userFrom, undefined), status) in 1159 (\*\_PROBE\_\*) 1160 stop 1161 1162 1163 1164 [] 1165 OnHook !userFrom; (\* From POTS 7-12? \*) 1166 1167 (\* Set userFrom Idle \*) 1168 let status:Status = Remove(stat(Busy, userFrom, undefined), status) in (\*\_PROBE\_\*) 1169 stop 1170 1171 ) 1172 1173 1174 (\* outPreD2: reject \*) 1175 [outPreD2 IsIn outPaths] -> 1176 1177 OnHook !userFrom; (\* From INTL 12 \*) 1178 1179 (\* Set userFrom Idle \*) 1180 let status:Status = Remove(stat(Busy, userFrom, undefined), status) in (\*\_PROBE\_\*) 1181 stop 1182 1183 1184 1185 where 1186 1187 1188 (\* Stub Process PreDialStub: 1189 1190 process PreDialStub[OnHook, Trigger, Response, Resource, Announce, Dial, Time] 1191 (inPaths: SPList, userFrom: Address, sdb: SDB, 1192 status: Status):exit (Address, SDB, Status, SPList) := 1193 (\* In this stub, INTL is mutually exclusive with all other features. \*) 1194 (\* INTL plugin \*) 1195 1196 [has(userFrom, INTL, sdb)] -> (\* NEW EVENTS: we believe that the INTL information should be located in the SCP. \*) 1197 (\* Is the time in the subscriber's TeenTime interval? \*) 1198 Time ?time:Time; (\* Get the current time \*) 1199 Resource !INTL !userFrom !time; 1200 1201 Response !INTL !userFrom ?inTeenTime:Bool; 1202

exit (userFrom, sdb, status, Insert(outPreD1, NoSPList)) (\* Restricted time for INTL \*) [inTeenTime] -> Trigger |ORIGINATION ATTEMPT |userFrom |userFrom |undefined |time ; Response |SEND TO RESOURCE !userFrom ?m:message; Announce !userFrom !m; OnHook !UserFrom; (\* PROBE \*) stop (\* INTL 13 \*) [] Dial !userFrom ?pin:PIN; Resource !userFrom !pin; Response (CONTINUE luserFrom luserFrom lundefined; (\* PROBE \*) exit (userFrom, sdb, status, Insert(outPreD1, NoSPList)) [] Response !SEND\_TO\_RESOURCE !userFrom !invalidPIN; Resource !userFrom !undefined; Announce !userFrom !invalidPIN; Response !DISCONNECT !userFrom !undefined; (\*\_PROBE\_\*) exit (userFrom, sdb, status, Insert(outPreD2, NoSPList)) ) [] (\* Default plugin \*) [not(has(userFrom, INTL, sdb))] -> (\* PROBE \*) exit (userFrom, sdb, status, Insert(outPreD1, NoSPList)) endproc (\* PreDialStub \*) (\* Stub Process PostDialStub; process PostDialStub[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display Trigger, Resource, Response, LogBegin, LogEnd, Time] (inPaths: SPList, userFrom: Address, userTo:Address, sdb: SDB, status: Status):exit (Address, Address, Address, SDB, Status, SPList) := (\* Use the processCallStub first \*) ProcessCallStub[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display Trigger, Resource, Response, LogBegin, LogEnd, Time] (Insert(inPC1, NoSPList), userFrom, userTo, sdb, status) accept userFrom:Address, userTo:Address, userPay:Address, sdb:SDB, status:Status, outPaths:SPList in (\* All choices are mutually exclusive here. \*) (\* outPC1: Tdle \*) [outPC1 IsIn outPaths] -> (\* Set userTo Busy \*) let status:Status = Insert(stat(Busy, userTo, undefined), status) in StartAR JuserFrom JuserTo; (\* PROBE \*) exit (userFrom, userTo, userPay, any SDB, Insert(stat(AudibleRinging, userFrom, userTo), Insert(stat(Ringing, userTo, userFrom), status))) StartR !userTo !userFrom; (\*\_PROBE\_\*)

12 November, 1998 17:47

(\* Unrestricted time for INTL \*)

[not(inTeenTime)] -> (\*\_PROBE\_\*)

1203

1204

LOTOS Specification

p. 58

12 November, 1998 17:47

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

LOTOS Specification

exit (userFrom, userTo, userPay, any SDB, Insert(stat(AudibleRinging, userFrom, userTo),

```
Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features
```

Insert(stat(Ringing, userTo, userFrom), status))) (\* For SetDisplayStub and CND plugin for ProcessCallStub \*) [has(userTo, CND, sdb)] -> (\* Update LastIncoming and Display it. \*) Display !userTo !userFrom; (\*\_PROBE\_\*) exit (userFrom, userTo, userPay, setLastIncoming(userTo, userFrom, sdb), any Status) [] [not(has(userTo, CND, sdb))] -> (\* Do nothing special \*) (\*\_PROBE\_\*) exit (userFrom, userTo, userPay, sdb, any Status) ) ) >> accept userFrom:Address, userTo:Address, userPay:Address, sdb:SDB, status:Status in OffHook !userTo; (\* POTS 5 \*) ( StopAR !userFrom !userTo; (\*\_PROBE\_\*) exit ( userFrom, userTo, userPay, sdb, Remove(stat(AudibleRinging, userFrom, userTo), Remove(stat(Ringing, userTo, userFrom), status)), Insert(outPostD1, Insert(outPostD2, Insert(outPostD3, NoSPList))) ) StopR !userTo !userFrom; (\*\_PROBE\_\*) exit ( userFrom, userTo, userPay, sdb, Remove(stat(AudibleRinging, userFrom, userTo), Remove(stat(Ringing, userTo, userFrom), status)), Insert(outPostD1, Insert(outPostD2, Insert(outPostD3, NoSPList))) ) ) [] (\* Disconnection possible here... \*) OnHook !userFrom; (\* POTS 5 \*) ( (\* Set userFrom Idle \*) let status:Status = Remove(stat(Busy, userFrom, undefined), status) in StopAR !userFrom !userTo; (\* Set userTo idle after the synchronization \*) (\*\_PROBE\_\*) exit (userFrom, userTo, userPay, sdb, Remove(stat(Busy, userTo, undefined), Remove(stat(AudibleRinging, userFrom, userTo), Remove(stat(Ringing, userTo, userFrom), status))), Insert(outPostD1, NoSPList)) StopR !userTo !userFrom; (\*\_PROBE\_\*) exit (userFrom, userTo, userPay, sdb, Remove(stat(Busy, userTo, undefined), Remove(stat(AudibleRinging, userFrom, userTo), Remove(stat(Ringing, userTo, userFrom), status))), Insert(outPostD1, NoSPList)) ) ) [] (\* outPC2: Busv \*) [outPC2 IsIn outPaths] -> (\* Invoke BusyStub \*) BusyStub[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Trigger, Resource, Response, LogBegin, LogEnd, Time] (Insert(inBusyl, NoSPList), userFrom, userTo, userPay, sdb, status) >>

1333	<pre>accept userFrom:Address, userTo:Address, userPay:Address, sdb:SDB, status:Status, outPaths:SPList in</pre>
1334	[outBusyl IsIn outPaths] ->
1335	(* CC: invalid PIN *) (* PROBE *)
1336	exit (userFrom, userTo, userPay, sdb, status, Insert(outPostD5, NoSPList))
1337	
1338	[outBuey2 IsTn outPaths] ->
1330	(at DF DONE *1 (* DPOPE *1)
1340	
1340	scop
1341	
1342	
1343	(* OULPC3: Reject *)
1344	[outpc3 IsIn outpaths] ->
1345	
1346	(* TCS Reject *) (*_PROBE_*)
1347	<b>exit</b> (userFrom, userTo, userPay, sdb, status, Insert(outPostD4, NoSPList))
1348	)
1349	[]
1350	(* outPC4: Back to stub *)
1351	[outPC4 IsIn outPaths] ->
1352	(
1353	(*_PROBE_*) <b>stop</b> (* TO BE DONE *)
1354	)
1355	
1356	where
1357	
1358	(**************************************
1359	(* Stub Process ProcessCallStub: *)
1360	(**************************************
1361	process ProcessCallStub [OffHook, OnHook, Dial, Flash, DialTone,
1362	StartAR, StartR, StartCWT, StopAR, StopR, StopCWT,
1363	LineBusyTone, Announce, Disconnect, Display,
1364	Trigger, Resource, Response, LogBegin, LogEnd, Time]
1365	(inPaths: SPList, userFrom: Address, userTo:Address,
1366	sdb: SDB, status: Status)
1367	: exit (Address, Address, Address, SDB, Status, SPList) :=
1368	
1369	(* CND will be taken care of at outPCL, after all these plug-ins. *)
1370	
1371	(* TCS (reject path) has priority over the other features. *)
1372	[has(userTo, TCS, sdb) and isOnTCS(userFrom, userTo, SDB)] -> (* PROBE *)
1373	(* Caller on the list. Reject call. *)
1374	exit (userFrom, userTo, userFrom, sdb, status, Insert(outPC3, NoSPList))
1375	
1376	(* Remaining features *)
1377	[not(bas(userTo, TCS, sdb) and isOnTCS(userFrom, userTo, SDB)]] ->
1378	
1379	(* TNFB *)
1380	(1) $(1)$
1381	PluginINFRONT (Office A Dial Flash DialTone Startar
1202	Figurine Bioteck, State Charles, State State K,
1302	Line Diversion Announce Disconnect Dischlar
1303	ThireBusyTone, Announce, Disconnect, Display,
1205	frigger, kesource, kesponse, Logsegin, Logsna, fimej
1385	(inpaths, userFrom, userIo, sdb, status)
1380	
1387	(* DEFAULT *)
1388	<pre>[not(nas(userTo, INFB, SdD))] -&gt; (*_PROBE_*)</pre>
T388	PiuginDerauit[OffHook, OnHook, Dial, Flash, DialTone, StartAR,
1390	StartR, StartCWT, StopAR, StopR, StopCWT,
1391	LineBusyTone, Announce, Disconnect, Display,
1392	Trigger, Resource, Response, LogBegin, LogEnd, Time]
1393	(inPaths, userFrom, userTo, sdb, status)
1394	)
1395	where
1396	

12 November, 1998 17:47

LOTOS Specification

p. 60

12 November, 1998 17:47

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

1397	process PluginINFB[OffHook, OnHook, Dial, Flash, DialTone,
1398	StartAR, StartR, StartCWT, StopAR, StopR, StopCWT,
1399	LineBusyTone, Announce, Disconnect, Display,
1400	Trigger, Resource, Response, LogBegin, LogEnd, Timel
1401	(inPaths: SPList, userFrom: Address, userTo:Address,
1402	sdb: SDB, status: Status)
1403	: exit (Address, Address, Address, SDB, Status, SPList) :=
1404	
1405	(* INFB plugin for ProcessCallStub *)
1406	Time 21:Time;
1407	Trigger LINFO ANALYZED LUSERTO LUSERTOM LUSERTO LL;
1408	Response IANALYZE ROUTE JuserTo JuserTo JuserTo JuserTo;
1409	
1410	$[\text{Istdle}(\text{userTo}, \text{status})] \rightarrow (* PROBE *)$
1411	(* Called party (userTo) pays. *)
1412	exit (userFrom, userTo, userTo, sdb, status, Insert(outPC1, NoSPList))
1413	
1414	[TeBusy(userTo_status)] -> /* PROBE *)
1415	evit (userTo, status); / (
1416	Care (derrow, derro, derro, sub, status, insert(outrez, hosphist))
1417	, and read (* Plugin INFR *)
1418	
1419	process PlugipDefault (OffHook OnHook Dial Flash DialTone
1420	Start R Start R Start WT Ston R Ston R Ston R Ston R
1421	LineBusyTone Announce Disconnect Display
1422	Triager Resource Resource LogRed LogEnd Timel
1423	(inpaths: Spligt userFrom: Address userToiAddress
1424	adb. CDB_status. Status.
1425	: avit (Address Address SDR Status SDList) :=
1426	
1427	(* Default plugin for ProcessCallStub *)
1429	[Tetdle(userTo_ctatus)] => (* PP/PF *)
1429	evit (userFrom userFrom sch status Insert(outPC1 NoSPList))
1420	(1)
1421	[J]
1432	[IBUSY(USERFOR) USERFOR SCHEET/N SCHEET/NET/OUTPC2 NoSDIST))
1432	exit (definition, definit, definition, sub, status, filsert(outpuz, Nosphist))
1433	endplot (" Filginberault ")
1435	endprog (* ProcessCellStub *)
1436	endplot ( Flotesstallstub )
1437	· ************************************
1439	(* Stub Process PusyStub)
1439	( ) 5 cm = 1 0 c c c s 3 y c c m , ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) (
1440	process BusyStub (OffHook OnHook Dial Flash DialTone
1441	Startip Start Start Start Start Start
1442	LineBusyTone Announce Disconnect Display
1443	Trigger, Resource, Response, LogRegin, LogEnd, Timel
1444	(inPaths: SPList, userFrom: Address, userTo:Address,
1445	userPav:Address, sdb: SDB, status; Status)
1446	: exit (Address, Address, Address, SDB, Status, SPList) :=
1447	(,,, _,, _,, _
1448	(* POTS default plugin, *)
1449	exit (userFrom, userTo, userPay, sdb, status, Insert(outBusy1, NoSPList))
1450	(* No probe here. Obviously covered for now. *)
1451	(* TO BE DONE: other pluging, *)
1452	endproc (* BusyStub *)
1453	
1454	endproc (* PostDialStub *)
1455	
1456	endproc (* Switch *)
1457	
1458	
1459	(**************************************
1460	(* Process SCP: *)
1461	(**************************************

1462	process SCP [Trigger, Resource, Response, LogBegin, LogEnd, AirBegin, AirEnd]
1463	(scpbd:SCPDB) : noexit :=
1464	
1465	(* From INTL: check if time is within TeenTime limits, *)
1466	Resource !INTL ?user:Address ?time:Time;
1467	Response LINTL, luser LISINTeenTime(user, time, scopd); (* PROBE *)
1468	SCP [Trigger, Resource, Response, LogRegin, LogEnd, AirBegin, AirEnd](scpbd)
1469	[]
1470	(* From INTL: Origination attempt to connect lask for PIN *)
1471	( FIGH INTE OFFICIENT ALCONDICT States States States Internet Ask for the states of the states (see a second
1470	inger (origination_Antempt fuser.Address fuserz.Address funderined ft.inde [user eq userz];
1472	Response (SEMD_10_RESOURCE (USE) (ASKFOIFIN; ("_RKOBE_")
1473	SCP [Irigger, Resource, Response, LogBegin, LogEnd, AirBegin, AirBegin, AirBegin, AirBegin, AirBegin, ScpDd)
1474	
1475	(* From INIL: Check PIN *)
1470	Resource fuser.Address fpin.Fin,
1470	
14/8	[ISValidieenPik(User, pin, schol]] ->
1479	Response !CONTINUE !user !user !undefined; (*_PROBE_*)
1480	SCP [Trigger, Resource, Response, LogBegin, LogEnd, AirBegin, AirEnd](scpbd)
1481	[]
1482	[not(IsValidTeenPIN(user, pin, scpbd))] ->
1483	Response !SEND_TO_RESOURCE !user !invalidPIN;
1484	Resource ?user:Address !undefined;
1485	Response !DISCONNECT !user !undefined; (*_PROBE_*)
1486	SCP [Trigger, Resource, Response, LogBegin, LogEnd, AirBegin, AirEnd](scpbd)
1487	)
1488	[]
1489	(* From INFB: IN Analyze *)
1490	Trigger !INFO_ANALYZED ?userTo:Address ?userFrom:Address ?userTo2:Address ?t:Time [userTo eq userTo2]
1491	Response !ANALYZE_ROUTE !userTo !userFrom !userTo !userTo; (*_ <i>PROBE_</i> *)
1492	SCP [Trigger, Resource, Response, LogBegin, LogEnd, AirBegin, AirEnd](scpbd)
1493	endproc (* SCP *)
1494	
1495	
1400	
1495	{*****
1495 1496 1497	(*************************************
1495 1496 1497 1498	(* Process OS: *) (***********************************
1495 1496 1497 1498 1499	(*************************************
1495 1496 1497 1498 1499 1500	<pre>(************************************</pre>
1496 1497 1498 1499 1500 1501	<pre>(**Process OS: *) (**Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin?Prom:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBeqin, LogEnd, AirBegin, AirEnd, Ouery]</pre>
1496 1497 1498 1499 1500 1501 1502	<pre>(************************************</pre>
1495 1496 1497 1498 1499 1500 1501 1502 1503	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?From:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(Begin, From, To, Paying, t),log)) []</pre>
1495 1496 1497 1498 1499 1500 1501 1502 1503 1504	<pre>(**Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?From:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(Begin, From, To, Paying, t,).log)) [] LogEnd ?From:Address ?To:Address ?t:Time; (* PROBE *)</pre>
1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin, ?From:Address ?To:Address ?To:Yaying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Begin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query]</pre>
1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506	<pre>(**Process OS: *) (* Process OS: *) (**Process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?Prom:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(Begin, From, To, Paying, t),log)) [] LogEnd ?Prom:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(End, From, To, Undefined, t),log)) </pre>
1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507	<pre>(**Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?From:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] [] LogEnd ?From:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(End, From, To, undefined, t),log)) [] </pre>
1495 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?From:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Hegin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] [] [] [] [] [] [] [] [] [] [] [] []</pre>
1495 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508	<pre>(**Process OS: *) (* Process OS: *) (**Process OS: *) (**Process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?From:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Bed, From, To, Paying, t),log)) [] LogEnd ?Prom:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] [* For Phase II features *) bitPedin ?Errom:Address ?t:Time; (* PROBE *)</pre>
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?From:Address ?To:Address ?Paying:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Begin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] [] (* For Phase II features *) AirBegin ?From:Address ?t:Time: (*_PROBE_*) OS[LogBerin LogEnd AirBegin AirEnd, Query]</pre>
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510	<pre>(**Process OS: *) (* Process OS: *) (**Process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?Prom:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(Bed, From, To, vaying, t,),log)) [] (* For Phase II features ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(End, From, To, undefined, t,),log)) [] (* For Phase II features *) AirBegin ?Prom:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(End, From, To, undefined, t,),log)) [] (* For Phase II features ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(Direxs) ?t:Time; Undefined, Undefined, t, log)) (* To the total content is the total content</pre>
1496 1497 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1510	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin, ?From:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirAnd, Query] (Insert(1(Begin, From, To, Paying, t),10g)) [] LogEnd ?From:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirAnd, Query] (Insert(1(End, From, To, undefined, t),10g)) [] (* For Phase II features *) AirBegin ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirAnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),10g)) [] </pre>
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1511	<pre>(**Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?From:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Bed, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirBegin ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, To, undefined, t),log)) [] (* For Phase II features *) AirBegin ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] </pre>
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1511	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin, From:Address ?Po:Address ?Poying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Begin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirBegin ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OV[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1AirBegin, From, Undefined, Undefined, t),log)) [] </pre>
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?Prom:Address ?Po:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, Paying, t),log)) [] LogEnd ?Prom:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirBegin ?Prom:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, AirBegin, AirEnd, Query] (Insert(1(End, AirBegin, AirEnd, Query] (Insert(1(End, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?Prom:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (SlegBegin, LogEn</pre>
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1511 1511 1513 1514 1514	<pre>(**Process OS: *) (* Process OS: *) (**Process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?From:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(End, From, To, Paying, t),log)) [] (* For Phase II features *) AirBegin ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirBegin, Prom:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(AirEngin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(AirEngin, From, undefined, Query] (Insert(l(AirEnd, From, undefined, Query]) (Insert(l(AirEnd, From, Undefined, Undefined, t),log)) (I) </pre>
1496 1497 1498 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1515	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin, Prom:Address ?To:Address ?Paying:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(Begin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(End, From, To, undefined, t),log)) [] [] (* For Phase II features *) AirBegin ?From:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(AirBegin, Prom, undefined, Query] (Insert(l(AirBegin, From, To, undefined, U),log)) [] AirEnd ?From:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(AirBegin, From, Undefined, U),log)) [] AirEnd ?From:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(l(AirBegin, From, Undefined, U),log)) [] </pre>
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1516 1517	<pre>(**Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?From:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Bedin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirBegin ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirEnd, From, undefined, undefined, t),log)) [] (* NEW functionality which allow a test case to check the Log. *) Comment Version (************************************</pre>
1496 1497 1498 1500 1501 1502 1503 1504 1505 1506 1507 1511 1512 1511 1512 1511 1514 1515 1516 1517 1517	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin, ?From:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Begin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirBegin ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] (* For Mass of the test case to check the Log. *) Query llog; (*_PROBE_*) </pre>
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519	<pre>(**Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?Prom:Address ?Po:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Bedin, From, To, Paying, t),log)) [] LogEnd ?Prom:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirBegin ?Prom:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?Prom:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirEnd, From, undefined, undefined, t),log)) [] (* NEW functionality which allow a test case to check the Log. *) Query log: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query](og)</pre>
1496 1497 1498 1500 1501 1502 1503 1504 1505 1506 1507 1508 1507 1510 1511 1512 1513 1514 1513 1514 1515 1516 1517 1519 1520	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin, YFrom:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Begin, From, To, Paying, t,).log)) [] LogEnd ?From:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t).log)) [] (* For Phase II features *) AirBegin ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t).log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t).log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirEnd, From, undefined, undefined, t).log)) [] (* NEW functionality which allow a test case to check the Log. *) Query log; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query](log) endproc (* OS *)</pre>
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1510 1511 1512 1513 1514 1515 1515 1515 1515 1515 1515	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?From:Address ?To:Address ?Paying:Address ?L:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Begin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirEngin ?From:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirEnd, From, undefined, undefined, t),log)) [] (* NEW functionality which allow a test case to check the Log. *) Query flog: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query](uery[log] endproc (* OS *)</pre>
1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1511 1511 1511 1511 1514 1515 1516 1517 1519 1520	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query[log:Log] : noexit := LogBegin /From:Address ?Po:Mdress ?Po:Mdress ?Po:Merres ??:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Beglin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirEnd, From, undefined, undefined, t),log)) [] (* NEW functionality which allow a test case to check the Log. *) Query log; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query](log) endproc (* OS *)</pre>
1496 1497 1498 1498 1498 1501 1501 1503 1505 1505 1506 1507 1508 1508 1508 1508 1510 1511 1513 1514 1515 1516 1515 1516 1515 1516 1515 1516 1515 1516 1515 1516 1515 1516 1515 1516 1515 1516 1517 1512 1520 1522 1522 1523 1522 1523 1524 1522 1523 1524 1524 1524 1524 1555 1555 1555 1555 1555 1555 1555 1557	<pre>(* Process OS: *) (* Process OS: *) (* Process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin, ?From:Address ?To:Address ?To:Yadress ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Eegin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirBegin ?From:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] (* NEW functionality which allow a test case to check the Log. *) Query log: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query](log) endproc (* OS *) </pre>
1496 1497 1498 1499 1500 1501 1503 1504 1505 1506 1507 1506 1507 1508 1509 1511 1511 1511 1511 1514 1515 1516 1517 1518 1519 1521 1522 1523 1524 1523 1524	<pre>(**Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin ?Prom:Address ?Po:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Red, From, To, Paying, t),log)) [] LogEnd ?Prom:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirBegin ?Prom:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?Prom:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirEnd, From, undefined, undefined, t),log)) [] (* NEW functionality which allow a test case to check the Log. *) Query 1Log: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query](log) endproc (* OS *) </pre>
1496 1497 1498 1498 1498 1500 1501 1503 1504 1505 1506 1507 1510 1511 1512 1513 1514 1515 1516 1517 1512 1512 1525 1525 1525 1525 1526 1527 1522 1522 1522 1522 1522 1522 1522 1522 1522 1522 1522 1522 1522 1525 1555	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : noexit := LogBegin, ?Prom:Address ?To:Address ?Paying:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Begin, From, To, Paying, t),log)) [] LogEnd, ?Prom:Address ?To:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, To, undefined, t),log)) [] AirEnd ?Prom:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?Prom:Address ?t:Time; (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirEnd, From, undefined, undefined, t),log)) [] (* NFW functionality which allow a test case to check the Log. *) Query llog: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query](log) endproc (* 0S *) [***********************************</pre>
1496 1497 1498 1498 1499 1500 1501 1503 1504 1505 1506 1507 1508 1508 1508 1509 1511 1513 1514 1515 1516 1517 1518 1518 1519 1520 1522 1522 1522 1525 1526	<pre>(* Process OS: *) (* Process OS: *) process OS [LogBegin, LogEnd, AirBegin, AirEnd, Query](log:Log) : newxit := LogBegin ?From:Address ?To:Address ?Paying:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(Begin, From, To, Paying, t),log)) [] LogEnd ?From:Address ?To:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(End, From, To, undefined, t),log)) [] (* For Phase II features *) AirEngin ?From:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirBegin, From, undefined, undefined, t),log)) [] AirEnd ?From:Address ?t:Time: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query] (Insert(1(AirEnd, From, undefined, undefined, t),log)) [] (* NEW functionality which allow a test case to check the Log. *) Query liog: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query](og: (*_PROBE_*) OS[LogBegin, LogEnd, AirBegin, AirEnd, Query]() ] (* Process Globalclock: computes the relative time incrementally and*) (* Process Globalclock: computes the relative time incrementally and*) [* Process Globalclock: computes the relative time incrementally and*) [* Process Globalclock: computes the relative time incrementally and*) [* Process Globalclock: computes the relative time incrementally and*) [* Process Globalclock: computes the relative time incrementally and*) [* Process Globalclock: computes the relative time incrementally and*) [* Process Globalclock: computes the relative time incrementally and*) [* Process Globalclock: computes the relative time incrementally and*) [* Process Globalclock: computes the relative time incrementally and*) [* Process Globalclock: computes the relative time incrementally and*] [* Process Globalclock: computes the relative time incrementally and*] [* Process Globalclock: computes the relative time incrementally and*] [* Process Globalclock: computes the relative time incrementally and*] [* Process Globalclock: computes the relative time incrementally and*] [* Process Globalclock: c</pre>

LOTOS Specification

12 November, 1998 17:47

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

1527 1528 process GlobalClock [Time](t:Time) : noexit := 1529 Time !T; GlobalClock[Time](tic(t)) (\* No probe here. Obviously covered... \*) 1530 endproc (\* GlobalClock \*) 1531 1532 1533 1534 (\*-----\*) 1535 (\* \*) 1536 (\* POTS PROCESSES \*) 1537 (\* \* 1 1538 (\*-----\*) 1539 1540 (\* These six processes represent common test sequences (repreenting \*) 1541 (\* a canonical tester) among many features (POTS states 1, 2, 4, 5, \*) (\* 13 and 15). 3WC and CW are not covered entirely by these. 1542 1543 (\* They all exit so that we can check the Log afterwards in test cases\*) 1544 (\* using these processes. 1545 1546 process POTS\_1 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1547 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1548 Disconnect, Display, Success] (userFrom:Address, userTo:Address) 1549 : exit(Nat) := 1550 OffHook !userFrom; 1551 POTS\_2 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1552 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1553 Disconnect, Display, Success] (userFrom, userTo) 1554 endproc (\* POTS 1 \*) 1555 1556 1557 process POTS\_2 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1558 Disconnect, Display, Success] (userFrom:Address, userTo:Address) 1559 1560 : exit(Nat) := 1561 DialTone !userFrom; (\* State 2 \*) 1562 i; OnHook !userFrom; exit(succ(succ(succ(0))))) (\* State 17 \*) 1563 1564 i; Dial !userFrom !userTo; (\* State 3 \*) 1565 1566 1567 POTS\_4 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1568 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1569 Disconnect, Display, Success] (userFrom, userTo) 1570 1571 POTS\_15 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1572 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1573 Disconnect, Display, Success] (userFrom, userTo) 1574 1575 1576 endproc (\* POTS\_2 \*) 1577 1578 1579 process POTS 4 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1580 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce. Disconnect, Display, Success] (userFrom:Address, userTo:Address) 1581 1582 : exit(Nat) := 1583 1584 StartAR !userFrom !userTo; exit(userFrom, userTo) 1585 StartR !userTo !userFrom; exit(userFrom, userTo) 1586 1587 >> accept userFrom:Address, userTo:Address in 1588 1589 1590 i; POTS\_5 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1591 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,

LOTOS Specification

p. 64

12 November, 1998 17:47

12 November, 1998 17:47

1592 Disconnect, Display, Success] (userFrom, userTo) 1593 1594 i; POTS\_13[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1595 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1596 Disconnect, Display, Success] (userFrom, userTo) 1597 1598 endproc (\* POTS 4 \*) 1599 1600 1601 process POTS\_5 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1602 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1603 Disconnect, Display, Success] (userFrom:Address, userTo:Address) 1604 : exit(Nat) := OffHook luserTo; 1605 1606 (\* State 6 \*) 1607 1608 StopAR !userFrom !userTo; exit(userFrom, userTo) 1609 1610 StopR !userTo !userFrom; exit(userFrom, userTo) 1611 1612 >> accept userFrom:Address, userTo:Address in 1613 1614 i; OnHook !userFrom; (\* State 7 \*) 1615 Disconnect !userTo !userFrom; (\* State 8 \*) 1616 onHook !userTo; exit(0) (\* State 9 \*) 1617 1618 i; OnHook !userTo; (\* State 10 \*) 1619 Disconnect !userFrom !userTo; (\* State 11 \*) 1620 OnHook !userFrom; exit(succ(0)) (\* State 12 \*) 1621 1622 endproc (\* POTS\_5 \*) 1623 1624 1625 process POTS 13/OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1626 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1627 Disconnect, Display, Success] (userFrom:Address, userTo:Address) 1628 : exit(Nat) := OnHook !userFrom; 1629 1630 1631 (\* State 14 \*) 1632 StopAR !userFrom !userTo; exit(succ(succ(0))) 1633 1634 StopR !userTo !userFrom; exit(succ(succ(0))) 1635 1636 endproc (\* POTS\_13 \*) 1637 1638 1639 process POTS\_15[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1640 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1641 Disconnect, Display, Success] (userFrom:Address, userTo:Address) 1642 : exit(Nat) := 1643 LineBusyTone !userFrom; OnHook !userFrom; exit(succ(succ(0)))) (\* State 16 \*) 1644 1645 endproc (\* POTS 15 \*) 1646 1647 1648 (\*-----\*) 1649 (\* \*) 1650 (\* TEST PROCESSES \*) 1651 (\* \*) 1653 1654 (\*\*\*\*\*\*\*\*\*) 1655 (\*\* POTS \*\*) 1656 (\*\*\*\*\*\*\*\*\*)

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

1657 1658 (\* TEST CASES \*) 1659 process tPOTS1 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1660 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1661 Disconnect, Display, Init, CreateUser, Query, Success] : noexit := 1662 (\* Cases where userB is not busy. \*) Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), 1663 Insert(sub(userB, NoFList, undefined, undefined, NoAddList, validPIN), NoSDB)) 1664 1665 !NoStatus INOSCEDB 1666 1667 !InitTime; 1668 CreateUser !userA !NoFList; CreateUser luserB !NoFList; 1669 POTS\_1 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1670 1671 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1672 Disconnect, Display, Success] (userA, userB) 1673 1674 (\* Check the Log \*) 1675 >> accept exitCode:Nat in 1676 1677 (\* One connection \*) 1678 [(exitCode eq 0) or (exitCode eq succ(0))] -> 1679 Query !Insert(1(End, userA, userB, undefined, tic(InitTime)), 1680 Insert(l(Begin, userA, userB, userA, InitTime), NoLog)); 1681 Success; stop 1682 [] 1683 (\* No connection \*) 1684 [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> 1685 Ouery !NoLog; 1686 Success; stop 1687 ) 1688 endproc (\* tPOTS1 \*) 1689 1690 process tPOTS2 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1691 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1692 Disconnect, Display, Init, CreateUser, Query, Success] : noexit := (\* Cases where userB is busy. \*) 1693 1694 Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), Insert(sub(userB, NoFList, undefined, undefined, NoAddList, validPIN), NoSDB)) 1695 !Insert(stat(Busy, userB, undefined), NoStatus) 1696 1697 !NoSCPDB 1698 !InitTime; 1699 CreateUser !userA !NoFList; 1700 CreateUser !userB !NoFList; 1701 POTS\_1 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1702 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success] (userA, userB) 1703 1704 1705 (\* Check the Log \*) 1706 >> accept exitCode:Nat in 1707 ( 1708 (\* No connection only. \*) 1709 [(exitCode eq succ(succ(succ(0)))) or (exitCode eq succ(succ(succ(succ(0)))))] -> 1710 Ouery !NoLog; 1711 Success; stop 1712 ) 1713 endproc (\* tPOTS2 \*) 1714 1715 (\*\*\*\*\*\*\*\*\*) 1716 (\*\* INTL \*\*) 1717 (\*\*\*\*\*\*\*\*\*\*) 1718 1719 (\* COMMON BEHAVIOUR \*) 1720 process cINTL1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1721 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,

1722	Disconnect, Display, Success] : <b>exit</b> (Nat) :=
1723	(* Cases where TeenTime is restricted and A provides the valid PIN. *)
1724	OffHook !userA;
1725	Announce  userA  AskForPIN;
1726	
1727	i; Dial !userA !validPIN;
1728	POTS 2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
1729	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce.
1730	Disconnect Display Success (user Juser B)
1731	[]
1732	i: OnWook luger): exit(succ(succ(succ(succ(succ(0))))))
1722	
1724	
1725	endproc (* civilii *)
1736	PROGASE SINTISIOFFUSAL ANUSAL Dial Flach DialTone StartAD StartD
1737	Charter of the stand s
1720	Diagonat Diagona Cugagadi ani (Nat) in
1720	(* Georgian in anti-ted and a description of the solid DTM (*)
1739	(* Cases where TeenTime is restricted and A does not provide the Valid Pin. *)
1740	UTHOOK (USERA)
1741	Announce lusera laskforpin;
1742	
1743	1; Dial Jusera Jinvallolin;
1744	Announce !userA !invalidPIN;
1745	OnHook !userA; exit (succ(succ(succ(succ(succ(0)))))))
1746	[]
1747	i; OnHook !userA; exit(succ(succ(succ(succ(succ(0))))))
1748	)
1749	endproc (* cINTL2 *)
1750	
1751	(* TEST PROCESSES *)
1752	process tINTL1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
1753	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
1754	Disconnect, Display, Success, CreateUser, Query, Init] : <b>noexit</b> :=
1755	(* Cases where TeenTime is not restricted. *)
1756	Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN),
1757	Insert(sub(userB, NoFList, undefined, undefined, NoAddList, validPIN), NoSDB))
1758	INoStatus
1759	<pre>!Insert(scp(TeenTime, userA, undefined, tic(tic(initTime)), tic(tic(tic(initTime))), undefined,</pre>
	validPIN), NoSCPDB)
1760	initime;
1761	Createuser !usera !Insert(INTL, NoFList);
1762	Createuser luserb INOFLIST;
1763	POTS_I[OTHOOK, OHHOOK, DIAI, Flash, DialTone, StartAR, StartR,
1764	Startcwr, Stopar, Stopar, Stopcwr, LineBusyTone, Announce,
1765	Disconnect, Display, Success (userA, userB)
1766	
1767	(* Check the Log *)
1768	>> accept exitCode:Nat in
1769	
1770	(* One connection *)
1771	[(exitCode eq 0) or (exitCode eq succ(0))] ->
1772	<pre>Query !Insert(l(End, userA, userB, undefined, tic(tic(InitTime))),</pre>
1773	<pre>Insert(l(Begin, userA, userB, userA, tic(InitTime)), NoLog));</pre>
1774	Success; stop
1775	[]
1776	(* No connection *)
1777	<pre>[not( (exitCode eq 0) or (exitCode eq succ(0)) )] -&gt;</pre>
1778	Query !NoLog;
1779	Success; stop
1780	)
1781	<pre>endproc (* tINTL1 *)</pre>
1782	
1783	process tINTL2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
1784	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
1785	Disconnect, Display, Success, CreateUser, Query, Init] : <b>noexit</b> :=

12 November, 1998 17:47

```
Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features
```

(\* Should Display the originator's number. \*)

StartAR !userA !userB; exit(userA, userB)

StartR !userB !userA; exit(userA, userB)

Display !userB !userA; exit(userA, userB)

>> accept userFrom:Address, userTo:Address in

1873 process tCND1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,

(\* Should Display the originator's number. \*)

Disconnect, Display, Success]

[(exitCode eq 0) or (exitCode eq succ(0))] ->

[not( (exitCode eq 0) or (exitCode eq succ(0)) )] ->

1905 process tCND2 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,

!Insert(stat(Busy, userB, undefined), NoStatus)

CreateUser !userB !Insert(CND, NoFList);

i; POTS\_5 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,

i: POTS\_13[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,

Disconnect, Display, Success] (userFrom, userTo)

Disconnect, Display, Success] (userFrom, userTo)

StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,

Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN),

cCND1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,

StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,

Query !Insert(1(End, userA, userB, undefined, tic(InitTime)),

Insert(l(Begin, userA, userB, userA, InitTime), NoLog));

StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,

Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN).

Disconnect, Display, Init, CreateUser, Ouery, Success] : noexit :=

Disconnect, Display, Success, CreateUser, Query, Init] : noexit :=

Insert(sub(userB, Insert(CND, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB))

StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,

DialTone !userA;

Dial !userA !userB;

1786 (\* Cases where TeenTime is restricted and A provides the valid PIN. \*) 1787 Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 1788 Insert(sub(userB, NoFList, undefined, undefined, NoAddList, validPIN), NoSDB)) 1789 !NoStatus 1790 !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(initTime)), undefined, validPIN), 1791 Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB)) 1792 ITnitTime: CreateUser !userA !Insert(INTL, NoFList); 1793 CreateUser !userB !NoFList; 1794 1795 cINTL1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1796 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1797 Disconnect, Display, Success] 1798 (\* Check the Log \*) 1799 1800 >> accept exitCode:Nat in 1801 1802 (\* One connection \*) 1803 [(exitCode eq 0) or (exitCode eq succ(0))] -> 1804 Query !Insert(1(End, userA, userB, undefined, tic(tic(InitTime))), 1805 Insert(l(Begin, userA, userB, userA, tic(InitTime)), NoLog)); 1806 Success; stop 1807 [] 1808 (\* No connection \*) 1809 [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> 1810 Query !NoLog; 1811 Success; stop 1812 1813 endproc (\* tINTL2 \*) 1814 1815 process tINTL3 OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1816 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success, CreateUser, Ouery, Init] : noexit := 1817 (\* Cases where TeenTime is restricted and A does not provide the valid PIN. \*) 1818 1819 Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 1820 Insert(sub(userB, NoFList, undefined, undefined, NoAddList. validPIN). NoSDB)) 1821 INoStatus !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(initTime)), undefined, validPIN), 1822 Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB)) 1823 1824 !InitTime; 1825 CreateUser !userA !Insert(INTL, NoFList); 1826 CreateUser !userB !NoFList; 1827 cINTL2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1828 1829 Disconnect, Display, Success] 1830 1831 (\* Check the Log \*) 1832 >> accept exitCode:Nat in 1833 1834 (\* No connection \*) 1835 [(exitCode eq succ(succ(succ(succ(0)))))) or (exitCode eq succ(succ(succ(succ(0))))))] -> 1836 Ouery !NoLog; 1837 Success; stop 1838 ) 1839 endproc (\* tINTL3 \*) 1840 1841 (\*\*\*\*\*\*\*\*) 1842 (\*\* CND \*\*) 1843 (\*\*\*\*\*\*\*\*) 1844 1845 (\* COMMON BEHAVIOUR \*) 1846 process cCND1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1847 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success] : exit(Nat) := 1848 1849 (\* Starts at POTS state 2. \*)

12 November,	1998 17:47
--------------	------------

LOTOS Specification

p. 68

12 November, 1998 17:47

INOSCEDB

!InitTime;

CreateUser !userA !NoFList;

1903 endproc (\* tCND1 \*)

1850

1851

1852

1853

1854

1855

1856

1857

1858

1859

1860

1861

1862

1863

1864

1865

1866

1867

1868

1869

1871

1872

1874

1875

1876

1877

1878

1879

1880

1881

1882

1883

1884

1885

1886

1887

1888

1889

1890

1891

1892

1893

1894

1895

1896

1897

1898

1899

1900

1901

1902

1904

1906

1907

1908

1909

1910

1911

1912

1913

1914

1870 endproc (\* cCND1 \*)

(\* TEST PROCESSES \*)

INoStatus

!InitTime;

CreateUser !userA !NoFList;

INOSCEDE

OffHook luserA;

(\* Check the Log \*)

>> accept exitCode:Nat in

(\* One connection \*)

Success; stop

(\* No connection \*)

Success; stop

(\* Cases where userB is busy. \*)

Ouery !NoLog;

LOTOS Specification

Insert(sub(userB, Insert(INFB, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB))

```
Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features
```

1915 CreateUser !userB !Insert(INFB, NoFList); 1916 POTS\_1 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1917 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1918 Disconnect, Display, Success] (userA, userB) 1919 1920 (\* Check the Log \*) 1921 >> accept exitCode:Nat in 1922 (\* No connection only, \*) 1923 1924 [(exitCode eq succ(succ(succ(0)))) or (exitCode eq succ(succ(succ(0)))))] -> 1925 Query !NoLog; 1926 Success; stop 1927 ) 1928 endproc (\* tCND2 \*) 1929 1930 (\*\*\*\*\*\*\*\*\*\*) 1931 (\*\* INFB \*\*) 1932 (\*\*\*\*\*\*\*\*\*) 1933 1934 (\* NO SPECIAL COMMON BEHAVIOUR \*) 1935 (\* TEST PROCESSES \*) 1936 process tINFB1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1937 1938 Disconnect, Display, Success, CreateUser, Query, Init] : noexit := 1939 (\* Cases where B is not Busy. Affect the billing. \*) 1940 Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), 1941 Insert(sub(userB, Insert(INFB, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB)) 1942 INoStatus 1943 INOSCEDE 1944 ITnitTime: CreateUser luser& !NoFList: 1945 CreateUser !userB !Insert(INFB, NoFList); 1946 POTS 1/OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1947 1948 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1949 Disconnect, Display, Success](userA, userB) 1950 (\* Check the Log. UserB should be charged. \*) 1951 >> accept exitCode:Nat in 1952 1953 1954 (\* One connection \*) 1955 [(exitCode eq 0) or (exitCode eq succ(0))] -> 1956 Query !Insert(1(End, userA, userB, undefined, tic(tic(InitTime))), Insert(l(Begin, userA, userB, userB, tic(InitTime)), NoLog)); 1957 1958 Success; stop 1959 ſ1 1960 (\* No connection \*) 1961 [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> 1962 Query !NoLog; 1963 Success; stop 1964 1965 endproc (\* tINFB1 \*) 1966 1967 process tINFB2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1968 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success, CreateUser, Query, Init] : noexit := 1969 (\* Cases where B is Busy. Do not affect billing. \*) 1970 1971 Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), Insert(sub(userB, Insert(INFB, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB)) 1972 !Insert(stat(Busy, userB, undefined), NoStatus) 1973 1974 INOSCEDB 1975 !InitTime; CreateUser !userA !NoFList; 1976 1977 CreateUser !userB !Insert(INFB, NoFList); 1978 POTS\_1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 1979 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,

1980 Disconnect, Display, Success](userA, userB) 1981 1982 (\* Check the Log \*) 1983 >> accept exitCode:Nat in 1984 1985 (\* No connection only. \*) 1986 [(exitCode eq succ(succ(succ(0))))) or (exitCode eq succ(succ(succ(succ(0)))))] -> 1987 Query INcLog: 1988 Success; stop 1989 1990 endproc (\* tINFB2 \*) 1991 1992 (\*\*\*\*\*\*\*\*) (\*\* TCS \*\*) 1993 1994 (\*\*\*\*\*\*\*\*\*) 1995 1996 (\* COMMON BEHAVIOUR \*) 1997 process cTCS1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 1998 1999 Disconnect, Display, Success] : exit(Nat) := 2000 OffHook !userA; 2001 DialTone !userA; (\* State 2 \*) 2002 2003 i; OnHook !userA; exit(succ(succ(succ(0))))) (\* State 17 \*) 2004 [] 2005 i; Dial !userA !userB; (\* State 3 \*) 2006 POTS\_4[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2007 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2008 Disconnect, Display, Success](userA, userB) 2009 2010 endproc (\* cTCS1 \*) 2011 2012 process cTCS2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2013 2014 Disconnect, Display, Success] : exit(Nat) := 2015 OffHook luserA; DialTone !userA; (\* State 2 \*) 2016 2017 i; OnHook !userA; exit(succ(succ(succ(succ(0))))) (\* State 17 \*) 2018 2019 [] 2020 i; Dial !userA !userB; (\* State 3 \*) 2021 POTS\_15[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2022 Disconnect, Display, Success](userA, userB) 2023 2024 2025 endproc (\* cTCS2 \*) 2026 2027 process cTCS3[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2028 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2029 Disconnect, Display, Success] : exit(Nat) := 2030 OffHook luserA: 2031 DialTone !userA; (\* State 2 \*) 2032 i; OnHook !userA; exit(succ(succ(succ(0))))) (\* State 17 \*) 2033 2034 i; Dial !userA !userB; (\* State 3 \*) 2035 2036 Announce !userA !ScreenedMessage; 2037 OnHook !userA; exit(succ(succ(succ(0))))) (\* TCS State 4, same as POTS State 17 \*) 2038 2039 2040 endproc (\* cTCS3 \*) 2041 2042 (\* TEST PROCESSES \*) 2043 process tTCS1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2044 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,

LOTOS Specification

p. 70

12 November, 1998 17:47

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

2045 2046 2047 2048	Disconnect, Display, Success, CreateUser, Query, Init] : noexit := (* Cases where B is not Busy and A is not screened. *) Init !Insert(sub(userA, NOFList, undefined, undefined, NOAddList, validPIN), Insert(sub(userB, Insert(TCS, NOFList), undefined, undefined, Insert(userC, NOAddList), validPIN),
2040	No@tatug
2049	INOCCODD
2050	
2051	Createllar Lucera INAFList:
2052	(resteller lucar) Incert/TCS NoTist):
2054	CTCS1[0ffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
2055	StartCWT, StopAR, StopC, StopCWT, LineBusyTone, Announce,
2056	Disconnect, Display, Success]
2057	
2058	(* Check the Log. UserA should be charged. *)
2059	>> accept exitCode:Nat in
2060	(
2061	(* One connection *)
2062	[(exitCode eq 0) or (exitCode eq succ(0))] ->
2063	Query !Insert(l(End, userA, userB, undefined, tic(InitTime)),
2064	<pre>Insert(l(Begin, userA, userB, userA, InitTime), NoLog));</pre>
2065	Success; stop
2066	
2067	(* No connection *)
2068	[not( (exitCode eq 0) or (exitCode eq succ(0)) )] ->
2069	Query !NoLog;
2070	Success; stop
2071	
2072	endproc (* filsi *)
2074	process tTCS2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
2075	StartCWT, StopAr, StopF, StopCWT, LineBusyTone, Announce.
2076	Disconnect, Display, Success, CreateUser, Query, Init] : noexit :=
2077	(* Cases where B is Busy and A is not screened. *)
2078	Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN),
2079	<pre>Insert(sub(userB, Insert(TCS, NoFList), undefined, undefined, Insert(userC, NoAddList), validPIN), NoSDB))</pre>
2080	!Insert(stat(Busy, userB, undefined), NoStatus)
2081	!NoSCPDB
2082	!InitTime;
2083	CreateUser !userA !NoFList;
2084	CreateUser !userB !Insert(TCS, NoFList);
2085	CTUSZIOTHOOK, UNHOOK, DIAI, Flash, Dialtone, Startak, Startk,
2086	Starttwi, Stopak, Stopk, Stoptwi, LineBusylone, Announce,
2087	biscomeet, bispiay, success)
2089	(* Check the Log. *)
2090	>> accept exitCode:Nat in
2091	(
2092	(* No connection only. *)
2093	[(exitCode eq succ(succ(0)))) or (exitCode eq succ(succ(succ(succ(0)))))] ->
2094	Query !NoLog;
2095	Success; stop
2096	)
2097	endproc (* tTCS2 *)
2098	
2099	process tTCS3[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
2100	StartCWT, StopAR, StopCWT, LineBusyTone, Announce,
2101	Disconnect, Display, Success, CreateUser, Query, Init] : noexit :=
2102	(* Cases where A is screened. *)
2103 2104	<pre>init !insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), Insert(sub(userB, Insert(TCS, NoFList), undefined, undefined, Insert(userA, NoAddList), validPIN), NoSDB))</pre>
2105	!NoStatus
2106	INOSCPDB
12 No	weigher 1908 17:47 LOTOS Specification p. 72
14 110	p. 72

2107 !InitTime; 2108 CreateUser !userA !NoFList; 2109 CreateUser !userB !Insert(TCS, NoFList); 2110 cTCS3[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2111 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2112 Disconnect, Display, Success] 2113 (\* Check the Log. \*) 2114 2115 >> accept exitCode:Nat in 2116 ( 2117 (\* No connection only. \*) ((exitCode eq succ(succ(succ(0))))) or (exitCode eq succ(succ(succ(succ(0))))))] -> 2118 Query !NoLog; 2119 2120 Success; stop 2121 2122 endproc (\* tTCS3 \*) 2123 2124 (\*=======\*) 2125 (\* \*) 2126 (\* FI TEST PROCESSES \*) 2127 (\* \*) 2128 (\*=======\*) 2129 2130 (\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*) 2131 (\* INTL - CND \*) 2132 (\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*) 2133 2134 process fiINTL\_CND[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2135 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2136 Disconnect, Display, Success, CreateUser, Query, Init] : noexit := 2137 (\* Should Display the originator's number. \*) 2138 2139 ( 2140 (\* Cases where TeenTime is not restricted. \*) 2141 Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), Insert(sub(userB, Insert(CND, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB)) 2142 2143 INoStatus 2144 !Insert(scp(TeenTime, userA, undefined, tic(tic(initTime)), tic(tic(tic(initTime))), undefined, validPIN), NoSCPDB) 2145 !InitTime; 2146 CreateUser !userA !Insert(INTL, NoFList); 2147 CreateUser !userB !Insert(CND, NoFList); 2148 OffHook !userA; 2149 cCND1 [OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2150 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2151 Disconnect, Display, Success] 2152 2153 (\* Check the Log. \*) >> accept exitCode:Nat in 2154 2155 2156 (\* One connection \*) 2157 [(exitCode eq 0) or (exitCode eq succ(0))] -> Query !Insert(l(End, userA, userB, undefined, tic(tic(InitTime))), 2158 2159 Insert(l(Begin, userA, userB, userA, tic(InitTime)), NoLog)); 2160 Success; stop 2161 [] (\* No connection \*) 2162 2163 [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> 2164 Query !NoLog; 2165 Success; stop 2166 2167 2168 [] 2169 2170 (\* tINTL2 \*) 12 November, 1998 17:47 LOTOS Specification

p. 73

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

2171 (\* Cases where TeenTime is restricted and A provides the valid PIN. \*) 2172 Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 2173 Insert(sub(userB, Insert(CND, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB)) 2174 !NoStatus 2175 !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(initTime)), undefined, validPIN), 2176 Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB)) 2177 ITnitTime: CreateUser !userA !Insert(INTL, NoFList); 2178 CreateUser !userB !Insert(CND, NoFList); 2179 2180 OffHook luserA; Announce !userA !AskForPIN; 2181 2182 Dial JuserA IvalidPIN; cCND1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR. 2183 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2184 Disconnect, Display, Success] 2185 2186 2187 (\* Check the Log. \*) >> accept exitCode:Nat in 2188 2189 2190 (\* One connection \*) 2191 [(exitCode eq 0) or (exitCode eq succ(0))] -> 2192 Query !Insert(1(End, userA, userB, undefined, tic(tic(InitTime))), Insert(l(Begin, userA, userB, userA, tic(InitTime)), NoLog)); 2193 2194 Success; stop 2195 [] 2196 (\* No connection \*) 2197 [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> 2198 Query !NoLog; 2199 Success; stop 2200 ) 2201 2202 [] 2203 ( 2204 (\* tINTL3 \*) 2205 (\* Cases where TeenTime is restricted and A does not provide the valid PIN. \*) Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 2206 Insert(sub(userB, Insert(CND, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB)) 2207 2208 !NoStatus !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(initTime)), undefined, validPIN), 2209 2210 Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB)) 2211 !InitTime; 2212 CreateUser !userA !Insert(INTL, NoFList); 2213 CreateUser !userB !Insert(CND, NoFList); cINTL2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2214 2215 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2216 Disconnect, Display, Success] 2217 2218 (\* Check the Log \*) 2219 >> accept exitCode:Nat in 2220 2221 (\* No connection \*) 2222 [(exitCode eq succ(succ(succ(succ(0)))))) or (exitCode eq succ(succ(succ(succ(succ(0))))))] -> 2223 Ouerv !NoLog; 2224 Success; stop 2225 ) 2226 ) 2227 endproc (\* fiINTL\_CND \*) 2228 2229 (\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (\* INTL - INFB \*) 2230 2232 2233 process fiINTL\_INFB[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2234 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,

2235 Disconnect, Display, Success, CreateUser, Query, Init] : noexit := 2236 (\* Should Display the originator's number. \*) 2237 2238 2239 (\* Cases where TeenTime is not restricted \*) 2240 Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 2241 Insert(sub(userB, Insert(INFB, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB)) 2242 INoStatus 2243 !Insert(scp(TeenTime, userA, undefined, tic(tic(tic(initTime))), 2244 tic(tic(tic(initTime)))), undefined, validPIN), NoSCPDB) ITnitTime; 2245 2246 CreateUser !userA !Insert(INTL, NoFList); CreateUser !userB !Insert(INFB, NoFList); 2247 POTS\_1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2248 2249 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success](userA, userB) 2250 2251 (\* Check the Log. UserB should be charged. \*) 2252 2253 >> accept exitCode:Nat in 2254 2255 (\* One connection \*) 2256 [(exitCode eq 0) or (exitCode eq succ(0))] -> 2257 Query !Insert(l(End, userA, userB, undefined, tic(tic(tic(InitTime)))), 2258 Insert(l(Begin, userA, userB, userB, tic(tic(InitTime))), NoLog)); 2259 Success; stop 2260 2261 (\* No connection \*) 2262 [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> 2263 Ouery !NoLog; 2264 Success: stop 2265 2266 ) 2267 [] 2268 ( 2269 (\* tINTL2 \*) 2270 (\* Cases where TeenTime is restricted and A provides the valid PIN. \*) Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 2271 Insert(sub(userB, Insert(INFB, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB)) 2272 2273 !NoStatus 2274 !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(tic(initTime))), undefined, validPIN), 2275 Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB)) 2276 !InitTime; CreateUser !userA !Insert(INTL, NoFList); 2277 CreateUser !userB !Insert(INFB, NoFList); 2278 2279 cINTL1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2280 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2281 Disconnect, Display, Success] 2282 2283 (\* Check the Log. UserB should be charged. \*) 2284 >> accept exitCode:Nat in 2285 ( 2286 (\* One connection \*) 2287 [(exitCode eq 0) or (exitCode eq succ(0))] -> Ouery !Insert(l(End, userA, userB, undefined, tic(tic(tic(TnitTime)))). 2288 Insert(l(Begin, userA, userB, userB, tic(tic(InitTime))), NoLog)); 2289 2290 Success; stop 2291 [] (\* No connection \*) 2292 2293 [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> Query !NoLog; 2294 2295 Success; stop 2296 2297 2298 [] 2299

12 November, 1998 17:47

LOTOS Specification

p. 74

12 November, 1998 17:47

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

2300 (\* tINTL3 \*) 2301 (\* Cases where TeenTime is restricted and A does not provide the valid PIN. \*) Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 2302 2303 Insert(sub(userB, Insert(INFB, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB)) 2304 INoStatus 2305 !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(tic(initTime))), undefined, validPIN), 2306 Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB)) 2307 ITnitTime: CreateUser !userA !Insert(INTL, NoFList); 2308 CreateUser !userB !Insert(INFB, NoFList); 2309 cINTL2[OffHook, OnHook, Dial, Flash, DialTone, StartAR. StartR. 2310 2311 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success] 2312 2313 (\* Check the Log \*) 2314 2315 >> accept exitCode:Nat in 2316 ( 2317 (\* No connection \*) [(exitCode eq succ(succ(succ(succ(0)))))) or (exitCode eq 2318 succ(succ(succ(succ(0))))))] -> Query !NoLog; 2319 2320 Success; stop 2321 ) 2322 ) 2323 endproc (\* fiINTL\_INFB \*) 2324 2325 (\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2326 (\*\* CND - TNEB \*\* 2328 2329 process fiCND INFB/Offhook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2330 Disconnect, Display, Success, CreateUser, Query, Init] : noexit := 2331 2332 ( 2333 (\* Should Display the originator's number. \*) (\* Cases where B is not Busy. Affect the billing. \*) 2334 Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), 2335 Insert(sub(userB, Insert(CND, Insert(INFB, NoFList)), undefined, undefined, NoAddList, validPIN), 2336 NoSDB)) 2337 !NoStatus 2338 !NoSCPDB 2339 !InitTime; 2340 CreateUser !userA !NoFList; CreateUser !userB !Insert(CND, Insert(INFB, NoFList)); 2341 2342 OffHook !userA; 2343 cCND1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2344 2345 Disconnect, Display, Success] 2346 2347 (\* Check the Log. UserB should be charged. \*) 2348 >> accept exitCode:Nat in 2349 2350 (\* One connection \*) [(exitCode eq 0) or (exitCode eq succ(0))] -> 2351 Ouery !Insert(1(End, userA, userB, undefined, tic(tic(InitTime))), 2352 Insert(l(Begin, userA, userB, userB, tic(InitTime)), NoLog)); 2353 2354 Success; stop 2355 [] 2356 (\* No connection \*) [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> 2357 2358 Query !NoLog; 2359 Success; stop 2360 2361 2362 []

2363 2364 (\* Cases where B is Busy. Does not affect billing. \*) 2365 Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), 2366 Insert(sub(userB, Insert(CND, Insert(INFB, NoFList)), undefined, undefined, NoAddList, validPIN), NOSDB)) 2367 !Insert(stat(Busy, userB, undefined), NoStatus) 2368 INOSCEDE ITnitTime: 2369 2370 CreateUser !userA !NoFList; 2371 CreateUser !userB !Insert(CND, Insert(INFB, NoFList)); 2372 POTS 1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2373 StartCWT. StopAR, StopR, StopCWT, LineBusyTone, Announce, 2374 Disconnect, Display, Success](userA, userB) 2375 2376 (\* Check the Log \*) 2377 >> accept exitCode:Nat in 2378 ( 2379 (\* No connection only. \*) 2380 [(exitCode eq succ(succ(succ(0)))) or (exitCode eq succ(succ(succ(succ(0)))))] -> 2381 Query !NoLog; 2382 Success; stop 2383 2384 2385 2386 endproc (\* fiCND\_INFB \*) 2387 2388 (\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2389 (\* INTL - TCS \*) 2390 (\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2391 2392 process fiINTL TCS/OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2393 2394 Disconnect, Display, Success, CreateUser, Ouery, Init] : noexit := 2395 2396 (\* Cases where A's TeenTime is not restricted. A is not on B's TCS list, and B is idle. \*) 2397 2398 (\* tTCS1 and tINTL1 \*) Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 2399 Insert(sub(userB, Insert(TCS, NoFList), undefined, undefined, Insert(userC, NoAddList), validPIN), 2400 NOSDB)) 2401 INoStatus 2402 !Insert(scp(TeenTime, userA, undefined, tic(tic(initTime)), tic(tic(tic(initTime))), undefined, validPIN), NoSCPDB) 2403 !InitTime; 2404 CreateUser !userA !Insert(INTL, NoFList); CreateUser !userB !Insert(TCS, NoFList); 2405 2406 cTCS1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2407 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2408 Disconnect, Display, Success] 2409 2410 (\* Check the Log. \*) 2411 >> accept exitCode:Nat in 2412 2413 (\* One connection \*) [(exitCode eq 0) or (exitCode eq succ(0))] -> 2414 Query !Insert(1(End, userA, userB, undefined, tic(tic(InitTime))), 2415 2416 Insert(1(Begin, userA, userB, userA, tic(InitTime)), NoLog)); 2417 Success; stop F 1 2418 2419 (\* No connection \*) [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> 2420 Query !NoLog; 2421 2422 Success; stop 2423 2424 )

12 November, 1998 17:47

LOTOS Specification

p. 76

12 November, 1998 17:47

```
Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features
```

2484

p. 78

12 November, 1998 17:47

CreateUser !userB !Insert(TCS, NoFList);

2425 [] 2426 2427 (\* Cases where A's TeenTime is not restricted, A is not on B's TCS list, and B is Busy. \*) 2428 (\* tTCS2 and tINTL1 \*) 2429 Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 2430 Insert(sub(userB, Insert(TCS, NoFList), undefined, undefined, Insert(userC, NoAddList), validPIN), NOSDB)) 2431 !Insert(stat(Busy, userB, undefined), NoStatus) !Insert(scp(TeenTime, userA, undefined, tic(tic(initTime)), tic(tic(tic(initTime))), undefined, 2432 validPIN), NoSCPDB) 2433 !InitTime; 2434 CreateUser !userA !Insert(INTL, NoFList); CreateUser |userB |Insert(TCS, NoFList); 2435 cTCS2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2436 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2437 2438 Disconnect, Display, Success] 2439 (\* Check the Log. \*) 2440 >> accept exitCode:Nat in 2441 2442 (\* No connection only. \*) 2443 2444 [(exitCode eq succ(succ(succ(0)))) or (exitCode eq succ(succ(succ(0)))))] -> 2445 Ouery !NoLog; 2446 Success; stop 2447 2448 ) 2449 [] 2450 2451 (\* Cases where A's TeenTime is not restricted, and A is on B's TCS list. \*) 2452 (\* tTCS3 and tINTL1 \*) Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 2453 Insert(sub(userB, Insert(TCS, NoFList), undefined, undefined, Insert(userA, NoAddList), validPIN), 2454 NoSDB)) 2455 !Insert(stat(Busy, userB, undefined), NoStatus) 2456 !Insert(scp(TeenTime, userA, undefined, tic(tic(initTime)), tic(tic(tic(initTime))), undefined, validPIN), NoSCPDB) 2457 !InitTime; CreateUser !userA !Insert(INTL, NoFList); 2458 2459 CreateUser !userB !Insert(TCS, NoFList); 2460 cTCS3[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2461 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2462 Disconnect, Display, Success] 2463 (\* Check the Log. \*) 2464 2465 >> accept exitCode:Nat in 2466 2467 (\* No connection only. \*) 2468 [(exitCode eq succ(succ(succ(0)))) or (exitCode eq succ(succ(succ(0)))))] -> Query !NoLog; 2469 2470 Success; stop 2471 2472 2473 [] 2474 2475 (\* Cases where A's TeenTime is restricted, A has the valid PIN, A is not on B's TCS list, and B is idle. \* 2476 (\* tTCS1 and tINTL2 \*) Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN), 2477 Insert(sub(userB, Insert(TCS, NoFList), undefined, undefined, Insert(userC, NoAddList), validPIN), 2478 NoSDB)) 2479 !NoStatus !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(initTime)), undefined, validPIN), 2480 2481 Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB)) !InitTime; 2482 CreateUser !userA !Insert(INTL, NoFList); 2483

LOTOS Specification

12 November, 1998 17:47

```
2485
        cINTL1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
2486
              StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
2487
              Disconnect, Display, Success]
2488
2489
          (* Check the Log. *)
2490
         >> accept exitCode:Nat in
2491
            (* One connection *)
2492
2493
            [(exitCode eq 0) or (exitCode eq succ(0))] ->
             Query !Insert(l(End, userA, userB, undefined, tic(tic(InitTime))),
2494
2495
                    Insert(l(Begin, userA, userB, userA, tic(InitTime)), NoLog));
2496
               Success; stop
            ٢1
2497
2498
            (* No connection *)
            [not( (exitCode eq 0) or (exitCode eq succ(0)) )] ->
2499
2500
             Query !NoLog;
2501
               Success; stop
2502
2503
2504 []
2505
     (* Cases where A's TeenTime is restricted, A has the valid PIN, A is not on B's TCS list, and B is busy. *
2506
                                                    )
     (* tTCS2 and tINTL2 *)
2507
       Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN),
2508
2509
             Insert(sub(userB, Insert(TCS, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB))
2510
             !Insert(stat(Busy, userB, undefined), NoStatus)
2511
            !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(initTime)), undefined, validPIN),
2512
             Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB))
2513
            IInitTime:
       CreateUser !userA !Insert(INTL, NoFList);
2514
       CreateUser !userB !Insert(TCS, NoFList);
2515
2516
       cINTL1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
2517
              StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
2518
              Disconnect, Display, Success]
2519
         (* Check the Log. *)
2520
2521
         >> accept exitCode:Nat in
2522
         (
2523
            (* No connection only. *)
2524
            [not( (exitCode eq 0) or (exitCode eq succ(0)) )] ->
             Query !NoLog;
2525
2526
               Success; stop
2527
2528
2529 []
2530
2531 (* Cases where A's TeenTime is restricted, A has the valid PIN, A is on B's TCS list. *)
2532
     (* tTCS3 and tINTL2 *)
2533
       Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, validPIN),
2534
             Insert(sub(userB, Insert(TCS, NoFList), undefined, undefined, Insert(userA, NoAddList), validPIN),
                                                    NOSDB))
2535
             INoStatus
             !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(initTime)), undefined, validPIN),
2536
2537
             Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB))
2538
            !InitTime;
       CreateUser !userA !Insert(INTL, NoFList);
2539
       CreateUser !userB !Insert(TCS, NoFList);
2540
         OffHook luserA;
2541
            Announce !userA !AskForPIN;
2542
2543
2544
             i; Dial !userA !validPIN;
2545
               DialTone !userA;
2546
                 Dial !userA !userB
                                                                                                             p. 79
```

```
Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features
```

2547 Announce !userA !ScreenedMessage; 2548 OnHook !userA; 2549 exit(succ(succ(succ(0))))) (\* TCS State 4, same as POTS State 17 \*) 2550 2551 i; OnHook !userA; exit(succ(succ(0)))) 2552 2553 (\* Check the Log. \*) 2554 >> accept exitCode:Nat in 2555 2556 2557 (\* No connection only. \*) 2558 [(exitCode eq succ(succ(succ(0)))) or (exitCode eq succ(succ(succ(succ(0)))))] -> 2559 Query !NoLog; 2560 Success; stop 2561 ) 2562 2563 [] 2564 ( 2565 (\* Cases where A's TeenTime is restricted, A has an invalid PIN, A is not on B's TCS list, and B is idle\*) 2566 (\* tTCS1 and tINTL3 \*) Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, invalidPIN), 2567 2568 Insert(sub(userB, Insert(TCS, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB)) 2569 !NoStatus 2570 !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(initTime)), undefined, validPIN), 2571 Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB)) 2572 !InitTime; 2573 CreateUser !userA !Insert(INTL, NoFList); 2574 CreateUser !userB !Insert(TCS, NoFList); 2575 cINTL2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2576 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2577 Disconnect, Display, Success] 2578 2579 (\* Check the Log \*) >> accept exitCode:Nat in 2580 2581 (\* No connection \*) 2582 [(exitCode eq succ(succ(succ(succ(0)))))) or (exitCode eq 2583 succ(succ(succ(succ(0))))))] -> 2584 Query !NoLog; 2585 Success; stop 2586 2587 2588 [] 2589 2590 (\* Cases where A's TeenTime is restricted, A has an invalid PIN, A is not on B's TCS list, and B is busy\*) 2591 (\* tTCS1 and tINTL3 \*) 2592 Init !Insert(sub(userA, Insert(INTL, NoFList), undefined, undefined, NoAddList, invalidPIN), 2593 Insert(sub(userB, Insert(TCS, NoFList), undefined, undefined, NoAddList, validPIN), NoSDB)) 2594 !Insert(stat(Busy, userB, undefined), NoStatus) 2595 !Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(initTime)), undefined, validPIN), 2596 Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB)) 2597 ITnitTime: CreateUser !userA !Insert(INTL, NoFList); 2598 CreateUser !userB !Insert(TCS, NoFList); 2599 cINTL2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2600 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2601 2602 Disconnect, Display, Success] 2603 (\* Check the Log \*) 2604 2605 >> accept exitCode:Nat in 2606 2607 (\* No connection \*) 2608 [(exitCode eq succ(succ(succ(succ(0)))))) or (exitCode eq succ(succ(succ(succ(succ(0))))))] ->

610 611 612 613 614	Query !NoLog;
611 612 613 614	Success; stop
612 613 614	)
613 614	)
614	
615	(* Cases where A's TeenTime is restricted. A has an invalid PTN. A is on B's TCS list *)
616	(* tTCS1 and tINTL3 *)
617	This Incertable, Theory Incert(INTL Notics) undefined undefined NotAddist invalidIN
C10	The insert sub users, insert (Mil, write), underined, underined, would be (invalide in), underine (invalide in), under internet (invalide in), under in), under internet (invalide in), under internet (invalide in), un
918	insert(sub(userB, insert(TCS, NoFList), underined, underined, insert(userA, NoAdaList), validPIN)
	NoSDB))
619	!NoStatus
620	!Insert(scp(TeenTime, userA, undefined, initTime, tic(tic(initTime)), undefined, validPIN),
621	Insert(scp(TeenPIN, userA, undefined, initTime, initTime, undefined, validPIN), NoSCPDB))
622	!InitTime;
623	CreateUser !userA !Insert(INTL, NoFList);
624	CreateUser !userB !Insert(TCS, NoFList);
625	CINTL2[0ffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,
626	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,
627	Disconnect Display Success
629	Dibeometer, Dibpita, Pacebb,
620	(* Check the Log *)
620	( check the boy / check in
630	// accept exittode.wat in
031	
632	(* No connection *)
633	[(exitCode eq succ(succ(succ(succ(0)))))) or (exitCode eq
	<pre>succ(succ(succ(succ(succ(0))))))] -&gt;</pre>
634	Query !NoLog;
635	Success; stop
636	)
637	)
638	endproc (* fiINTL_TCS *)
639	
640	(************)
641	(* CND - TCS *)
642	(***********
643	
644	PROPAGE FIGNE TOCIOFFUNCE ONLOCK Dial Plach DialTona Startap Starta
645	State (WT Stand State St
616	Diagonnat Dignity States (Asstations Owner Tail) - newsters
640	Disconnect, Display, Success, Createoser, Query, Init] . Hoexit
647	
648	
649	(* Cases where A is not on B's TCS list, and B is idle. Should Display the originator's number. *)
650	(* tTCS1 and tCND1 *)
651	Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN),
	Insert(sub(userB, Insert(TCS, Insert(CND, NoFList)), undefined, undefined, NoAddList, validPIN),
652	NoSDB))
652	INoStatus
652 653	
652 653 654	INOSCPDB
652 653 654 655	!NoSCPDB !InitTime;
652 653 654 655 656	<pre>!NoSCPDB     !InitTime; CreateUser !user; !NoFList;</pre>
652 653 654 655 656 657	!NoSCPDB !InitTime; CreateUser !userA !NoFList; CreateUser !userB !Insert(TCS, Insert(CND, NoFList));
652 653 654 655 656 657 658	<pre>INOSCPDB IInitTime; CreateUser luserA !NoFList; CreateUser luserB !Insert(TCS, Insert(CND, NoFList)); OffMoke !userFA:</pre>
652 653 654 655 655 657 658 659	<pre>!NoSCPDB !InitTime; CreateUser !userA !NoFList; CreateUser !userB !Insert(TCS, Insert(CND, NoFList)); OffHook !userA; coNDI(offHook OnHook Dial Elash DialTone StartAP StartP</pre>
652 653 654 655 656 657 658 659 660	<pre>!NoSCPDB !InitTime; CreateUser luserA !NoFList; CreateUser luserB !Insert(TCS, Insert(CND, NoFList)); OffHook !userA:</pre>
652 653 654 655 655 655 658 659 660 661	<pre>!NoSCPDB !InitTime; CreateUser luserA !NoFList; CreateUser luserB !Insert(TCS, Insert(CND, NoFList)); OffHook !userA; cCND1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect_Display_Success]</pre>
652 653 654 655 656 657 658 659 660 661	<pre>INOSCEDB IInitTime; CreateUser luserA !NoFList; CreateUser luserB IInsert(TCS, Insert(CND, NoFList)); OffHook !userA; cCNDI(OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCMT, StopAR, StopP, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success]</pre>
652 653 654 655 656 657 658 659 660 661 662	<pre>!NoSCPDB !InitTime; CreateUser luserA !NoFList; CreateUser luserB !Insert(TCS, Insert(CND, NoFList)); OffHook !userA; cCND1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopE, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success] (# Check the Les #)</pre>
652 653 654 655 656 657 658 659 660 661 662 663	<pre>!NoSCPDB !InitTime; CreateUser !userA !NoFList; CreateUser !userB !Insert(TCS, Insert(CND, NoFList)); OffHook !userA; cCNDI(OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success] (* Check the Log *)</pre>
652 653 654 655 656 657 658 659 660 661 662 663 663 664	<pre>!NoSCPDB !InitTime; CreateUser !userB !NoFList; CreateUser !userB !Insert(TCS, Insert(CND, NoFList)); OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, cCND1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCMT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success] (* Check the Log *) &gt;&gt; accept exitCode:Nat in</pre>
652 653 654 655 656 657 658 659 660 661 662 663 664 665	<pre>!NoSCPDB !InitTime; CreateUser luserA !NoFList; CreateUser luserB !Insert(TCS, Insert(CND, NoFList)); OffHook, luserA; cCND1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success] (* Check the Log *) &gt;&gt; accept exitCode:Nat in (</pre>
652 653 654 655 655 655 658 659 660 661 662 663 664 665 665 666	<pre>!NoSCPDB !InitTime; CreateUser luserA !NoFList; CreateUser luserB !Insert(TCS, Insert(CND, NoFList)); OffHook !userA; cCNDl[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,         StartCWT, StopRR, StopCWT, LineBusyTone, Announce,         Disconnect, Display, Success] (* Check the Log *) &gt;&gt; accept exitCode:Nat in (         ( * One connection *)</pre>
652 653 654 655 656 657 658 659 665 665 6661 6662 6663 6664 665 6665 6665 6667	<pre>!NoSCPDB ITnitTime; CreateUser luserA !NoFList; CreateUser luserB !Insert(TCS, Insert(CND, NoFList)); OffHook, UnHook, Dial, Flash, DialTone, StartAR, StartR, StartCMT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success] (* Check the Log *) &gt;&gt; accept exitCode:Nat in ( (* One connection *) [[exitCode eq 0] or (exitCode eq succ(0))] -&gt;</pre>
652 653 654 655 656 657 665 660 661 662 663 664 665 666 666 666 667 668	<pre>!NoSCPDB !InitTime; CreateUser !userA !NoFList; CreateUser !userB !Insert(TCS, Insert(CND, NoFList)); OffHook !userA; cCND1{OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,     StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,     Disconnect, Display, Success] (* Check the Log *) &gt;&gt; accept exitCode:Nat in (     (* One connection *) [(exitCode eq 0) or (exitCode eq succ(0))] -&gt;     Query !Insert(l(End, userA, userB, undefined, tic(InitTime)),</pre>
652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669	<pre>INOSCPDB IInitTime; CreateUser luserA !NoFList; CreateUser luserB IInsert(TCS, Insert(CND, NoFList)); OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, StartCMT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success] (* Check the Log *) &gt;&gt; accept exitCode:Nat in ( (* One connection *) [(exitCode eq 0) or (exitCode eq succ(0))] -&gt; Query IInsert(1(EEd, userA, userB, undefined, tic(InitTime)), Insert(1(Begin, userA, userB, userA, InitTime), NoLog));</pre>
652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670	<pre>INOSCPDB IInitTime; CreateUser luserA !NoFList; CreateUser luserB IInsert(TCS, Insert(CND, NoFList)); OffHook, UserA; cCND1(OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,     StartCHT, StopA, StopP, StopCWT, LineBusyTone, Announce,     Disconnect, Display, Success] (* Check the Log *) &gt; accept exitCode:Nat in (     (* One connection *)     [(exitCode eq 0) or (exitCode eq succ(0))] -&gt;     Query !Insert(1(EEqin, userA, userB, undefined, tic(InitTime)),</pre>

12 November, 1998 17:47

12 November, 1998 17:47

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

2733

p. 82

!NoSCPDB

2671 [] 2672 (\* No connection \*) 2673 [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> 2674 Ouery !NoLog; 2675 Success; stop 2676 2677 ) 2678 [] 2679 ( (\* Cases where A is not on B's TCS list, and B is busy. \*) 2680 2681 (\* tTCS2 and tCND2 \*) 2682 Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), Insert(sub(userB, Insert(TCS, Insert(CND, NoFList)), undefined, undefined, 2683 Insert(userC, NoAddList), validPIN), NoSDB)) 2684 !Insert(stat(Busy, userB, undefined), NoStatus) !NoSCPDB 2685 2686 !InitTime; CreateUser !userA !NoFList; 2687 CreateUser !userB !Insert(TCS, Insert(CND, NoFList)); 2688 2689 cTCS2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2690 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2691 Disconnect, Display, Success] 2692 2693 (\* Check the Log. \*) 2694 >> accept exitCode:Nat in 2695 2696 (\* No connection only. \*) 2697 [(exitCode eq succ(succ(succ(0)))) or (exitCode eq succ(succ(succ(succ(0)))))] -> 2698 Ouery !NoLog; 2699 Success: stop 2700 2701 ) 2702 [] 2703 ( 2704 (\* Cases where A is on B's TCS list, and B is busy. Do not display. \*) 2705 (\* tTCS3 and tCND2 \*) Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN). 2706 2707 Insert(sub(userB, Insert(TCS, Insert(CND, NoFList)), undefined, undefined, Insert(userA, NoAddList), validPIN), NoSDB)) 2708 !Insert(stat(Busy, userB, undefined), NoStatus) 2709 !NoSCPDB 2710 !InitTime; 2711 CreateUser !userA !NoFList; CreateUser !userB !Insert(TCS, Insert(CND, NoFList)); 2712 2713 cTCS3[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2714 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, Disconnect, Display, Success] 2715 2716 2717 (\* Check the Log. \*) 2718 >> accept exitCode:Nat in 2719 2720 (\* No connection only, \*) 2721 [(exitCode eq succ(succ(succ(0)))) or (exitCode eq succ(succ(succ(0)))))] -> 2722 Ouerv !NoLog; 2723 Success; stop 2724 ) 2725 ) 2726 [] 2727 ( 2728 (\* Cases where A is on B's TCS list, and B is idle. Do not display. \*) 2729 (\* tTCS3 and tCND1 \*) Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), 2730 2731 Insert(sub(userB, Insert(TCS, Insert(CND, NoFList)), undefined, undefined, Insert(userA, NoAddList), validPIN), NoSDB)) 2732 INoStatus

LOTOS Specification

12 November, 1998 17:47

2734 !InitTime; 2735 CreateUser !userA !NoFList; 2736 CreateUser !userB !Insert(TCS, Insert(CND, NoFList)); 2737 cTCS3[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2738 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2739 Disconnect, Display, Success] 2740 2741 (\* Check the Log. \*) 2742 >> accept exitCode:Nat in 2743 2744 (\* No connection only. \*) [(exitCode eq succ(succ(succ(0)))) or (exitCode eq succ(succ(succ(succ(0)))))] -> 2745 2746 Query !NoLog; 2747 Success; stop 2748 2749 ) 2750 endproc (\* fiCND\_TCS \*) 2751 2752 2753 (\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*) 2754 (\* INFB - TCS \*) (\*\*\*\*\*\*\*\*\*\*\*\*\* 2755 2756 2757 process fiINFB\_TCS[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2758 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce 2759 Disconnect, Display, Success, CreateUser, Query, Init] : noexit := 2760 2761 2762 (\* Cases where A is not on B's TCS list, and B is idle. Affect the billing. \*) 2763 (\* tTCS1 and tINEB1 \*) Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), 2764 Insert(sub(userB, Insert(TCS, Insert(INFB, NoFList)), undefined, undefined, 2765 Insert(userC, NoAddList), validPIN), NoSDB)) 2766 INoStatus 2767 INOSCEDB 2768 !InitTime; 2769 CreateUser !userA !NoFList; 2770 CreateUser !userB !Insert(TCS, Insert(INFB, NoFList)); 2771 cTCS1[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR, 2772 StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce, 2773 Disconnect, Display, Success] 2774 2775 (\* Check the Log. UserB should be charged. \*) 2776 >> accept exitCode:Nat in 2777 2778 (\* One connection \*) 2779 [(exitCode eq 0) or (exitCode eq succ(0))] -> 2780 Query !Insert(l(End, userA, userB, undefined, tic(tic(InitTime))), 2781 Insert(l(Begin, userA, userB, userB, tic(InitTime)), NoLog)); 2782 Success; stop 2783 [] 2784 (\* No connection \*) [not( (exitCode eq 0) or (exitCode eq succ(0)) )] -> 2785 Query !NoLog; 2786 2787 Success; stop 2788 2789 2790 [] 2791 ( 2792 (\* Cases where A is not on B's TCS list, and B is busy. Do not affect the billing. \*) (\* tTCS2 and tINFB2 \*) 2793 2794 Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN), 2795 Insert(sub(userB, Insert(TCS, Insert(INFB, NoFList)), undefined, undefined, Insert(userC, NoAddList), validPIN), NoSDB)) 12 November, 1998 17:47 LOTOS Specification

p. 83

#### Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

2796	<pre>!Insert(stat(Busy, userB, undefined), NoStatus)</pre>	2859		Success; stop
2797	INOSCPDB	2860	)	
2798	!InitTime;	2861	)	
2799	CreateUser !userA !NoFList;	2862	endproc	(* fiINFB_TCS *)
2800	CreateUser !userB !Insert(TCS, Insert(INFB, NoFList));	2863		
2801	cTCS2[OffHook, OnHook, Dial, Flash, DialTone, StartAR, StartR,	2864	endspec	(* FI_UCM *)
2802	StartCWT, StopAR, StopR, StopCWT, LineBusyTone, Announce,			
2803	Disconnect, Display, Success]			
2804				
2805	(* Check the Log. *)			
2806	>> accept exitCode:Nat in			
2807				
2808	(* NO CONDECTION ONLY *)			
2809	Overy Notor:			
2811	Success ston			
2812	)			
2813				
2814				
2815				
2816	(* Cases where A is on B's TCS list, and B is busy. Do not affect the billing. *)			
2817	(* tTCS3 and tINFB2 *)			
2818	Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN),			
2819	<pre>Insert(sub(userB, Insert(TCS, Insert(INFB, NoFList)), undefined, undefined,</pre>			
	<pre>Insert(userA, NoAddList), validPIN), NoSDB))</pre>			
2820	!Insert(stat(Busy, userB, undefined), NoStatus)			
2821	!NoSCPDB			
2822	!InitTime;			
2823	CreateUser !userA !NoFList;			
2824	CreateUser !userB !Insert(TCS, Insert(INFB, NoFList));			
2825	CTCS3(UTHOOK, ONHOOK, DIAI, FIASH, DIAITONE, STATTAR, STATTAR, STATTAR,			
2826	Startcwr, StopAr, StopK, StopCwr, LineBusyTone, Announce,			
2827	Disconnect, Display, Success)			
2020	(* Check the Loc *)			
2829	(" Check the boy. ")			
2831	i accept extreme in the			
2832	(* No connection only, *)			
2833	[(exitCode eg succ(succ(succ(0)))) or (exitCode eg succ(succ(succ(succ(0)))))] ->			
2834	Query !NoLog;			
2835	Success; stop			
2836	)			
2837	)			
2838	Ω			
2839	(			
2840	(* Cases where A is on B's TCS list, and B is idle. Do not affect the billing. *)			
2841	(* tTCS3 and tINFB1 *)			
2842	Init !Insert(sub(userA, NoFList, undefined, undefined, NoAddList, validPIN),			
2843	Insert(sub(userB, Insert(TCS, Insert(INFB, NoFList)), undefined, undefined,			
	Insert(userA, NoAddList), validPIN), NoSDB))			
2844	!NoStatus			
2845	INOSCHDB			
2846	Initime;			
2847	CreateUser (UserA (NorList,			
2040	erceleter insert insert (its, insert(its, worldst)),			
2850	Start/WT Stople Stople Stople Stople Stople Start/WT Stople Start/WT Stople Sto			
2851	Disconect, Display, Success)			
2852				
2853	(* Check the Log. *)			
2854	>> accept exitCode:Nat in			
2855				
2856	(* No connection only. *)			
2857	[(exitCode eq succ(succ(0)))) or (exitCode eq succ(succ(succ(0)))))] ->			
2858	Query !NoLog;			

12 November, 1998 17:47

LOTOS Specification

12 November, 1998 17:47

#### B ERRONEOUS SPECIFICATION FOR STUB PROCESS-CALL

Here is the part of the incorrect LOTOS specification that was replaced by lines 1361 to 1395 in the correct specification of appendix A.

```
1361
             process ProcessCallStub [OffHook, OnHook, Dial, Flash, DialTone,
1362
                        StartAR, StartR, StartCWT, StopAR, StopR, StopCWT,
1363
                        LineBusyTone, Announce, Disconnect, Display,
1364
                        Trigger, Resource, Response, LogBegin, LogEnd, Time]
1365
                            (inPaths: SPList, userFrom: Address, userTo:Address,
1366
                             sdb: SDB, status: Status)
1367
                        : exit (Address, Address, Address, SDB, Status, SPList) :=
1368
1369
               (* CND will be taken care of at outPC1, after all these plug-ins. *)
1370
                (* TCS *)
1371
1372
               [has(userTo, TCS, sdb)] ->
1373
                 PluginTCS[OffHook, OnHook, Dial, Flash, DialTone, StartAR,
1374
                            StartR, StartCWT, StopAR, StopR, StopCWT,
1375
                            LineBusyTone, Announce, Disconnect, Display,
1376
                            Trigger, Resource, Response, LogBegin, LogEnd, Time]
1377
                              (inPaths, userFrom, userTo, sdb, status)
1378
               []
1379
                (* INFB *)
1380
               [has(userTo, INFB, sdb)] ->
1381
                 PluginINFB[OffHook, OnHook, Dial, Flash, DialTone, StartAR,
1382
                             StartR, StartCWT, StopAR, StopR, StopCWT,
1383
                            LineBusyTone, Announce, Disconnect, Display,
1384
                             Trigger, Resource, Response, LogBegin, LogEnd, Time]
1385
                               (inPaths, userFrom, userTo, sdb, status)
1386
1387
               (* Default *)
1388
               [not(has(userTo, INFB, sdb)) and not(has(userTo, TCS, sdb))] ->
                 PluginDefault[OffHook, OnHook, Dial, Flash, DialTone, StartAR,
1389
1390
                                StartR, StartCWT, StopAR, StopR, StopCWT,
1391
                                LineBusyTone, Announce, Disconnect, Display,
1392
                                Trigger, Resource, Response, LogBegin, LogEnd, Time]
1393
                                  (inPaths, userFrom, userTo, sdb, status)
1394
             where
1395
1396
               process PluginTCS [OffHook, OnHook, Dial, Flash, DialTone,
1397
                        StartAR, StartR, StartCWT, StopAR, StopR, StopCWT,
1398
                        LineBusyTone, Announce, Disconnect, Display,
1399
                        Trigger, Resource, Response, LogBegin, LogEnd, Time]
1400
                            (inPaths: SPList, userFrom: Address, userTo:Address,
1401
                             sdb: SDB, status: Status)
1402
                        : exit (Address, Address, Address, SDB, Status, SPList) :=
1403
1404
                  (* TCS plugin for ProcessCallStub *)
1405
                  [isOnTCS(userFrom, userTo, SDB)] ->
                    (* Caller on the list. Reject call. *)
1406
1407
                    exit (userFrom, userTo, userFrom, sdb, status, Insert(outPC3, NoSPList))
1408
1409
                 [not(isOnTCS(userFrom, userTo, SDB))]->
1410
                   (* Caller NOT on the list. Continue. *)
1411
1412
                      [IsIdle(userTo, status)] ->
1413
                        exit (userFrom, userTo, userFrom, sdb, status, Insert(outPC1, NoSPList))
1414
                      [1
1415
                      [IsBusy(userTo, status)] ->
1416
                        exit (userFrom, userTo, userFrom, sdb, status, Insert(outPC2, NoSPList))
1417
               endproc (* PluginTCS *)
1418
```