Feature Interaction Detection: a LOTOS-based approach

By Q. Fu, P. Harnois, L. Logrippo, J. Sincennes
Telecommunications Software Engineering Research Group
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ont. Canada K1N 6N5

Abstract

A methodology to detect feature interactions in the design of telephone systems is presented. First, the system with its features is specified in LOTOS. With the use of tools, the specification is then expanded into a state transition model, representing either the entire global state space or significant portions of it. Properties describe proper system and feature behavior; violations to these properties are symptoms of feature interaction. Observers, watchdogs, and trace analysis tools are constructed to check for deviations from these properties. These processes are used to search the state transitions, both during and after the construction of the state transition model. Traces presenting violation of properties are printed as Message Sequence Charts.

1 Introduction

On the occasion of the Fifth International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'98), a Feature Interaction (FI) detection contest was held [22]. The contest offered an opportunity to compare the efficiency and the adaptability of different methods and tools in the detection of feature interactions. Inviting research teams of all schools of thought to compete, under controlled conditions, in accomplishing a specific and predetermined task, set a basis to permit the assessment of the advantages and capabilities of the various methodologies. The University of Ottawa group participated in the contest. The team consisted of Luigi Logrippo (project leader), Qifang (Jennifer) Fu (graduate student), Pascal Harnois (graduate student) and Jacques Sincennes (systems programmer/analyst). This paper presents the approach of the team.

The descriptions of the behaviors of the system and of its features were submitted to the contestants in terms of Chisel diagrams [2]. Brief informal descriptions of the intended behavioral properties of each feature were also supplied. We refer to a companion paper in this special issue for basic information about the contest and the features.

The approach outlined in this paper rests on the use of the Formal Description Technique (FDT) [35] LOTOS [6][25][27]. LOTOS is used to specify the basic system and the features, along with the feature properties (in terms of expected and unexpected events and values).

A straightforward conversion of the Chisel diagrams into LOTOS was possible. In turn, specifying the properties necessitated more ingenuity: desired system

characteristics needed to be inferred from the Chisel diagrams and informal descriptions.

Several tools were used to assist in manipulating LOTOS specifications: syntactic and static semantics analyzers, step-by-step and random execution, symbolic and valued expansion, comparison and search of behaviors. To present the results, a tool for printing LOTOS traces in the form of Message Sequence Charts (MSCs) was used. Some of these tools were written especially for the contest, others were adapted.

2 Features and Feature Interactions

Although several formal definitions of features and feature interactions have been proposed in the literature, at least one of which is from the authors' group [16], it appears that none of them covers all the phenomena that in practice have been associated with these terms. Since the contest required identifying as many feature interactions as possible, it was decided to use pragmatic and semi-formal definitions, rather than formal ones. Of course, exact definitions for our approach exist in the sense that they are encoded in the formal specification language and in the detection tools that were used. However such definitions are complex and cover many cases, so they cannot be expressed in few lines or few formulae.

2.1 Features

In this approach, a system is specified in terms of its behavior, consisting of sets of temporally ordered external events, the *traces*. A feature is a self-contained subset of the behavior of the system. Adding a feature consists of extending the system's behavior. Intuitively in the design of telephone systems, this is conceived of as adding functionality. Such is the case for features like Call Forward and Caller Name Display. In other instances, a feature is somewhat destructive and reduces the set of behaviors of the system. An example of the latter case would be a feature like Incoming Call Screening which, under certain circumstances, prevents a call from being terminated.

Adding a feature with respect to a given system consists of modifying the system such that it incorporates the modifications while retaining its previous functionality, to the extent set by the new feature.

2.2 Feature Interactions

The definition of feature interaction used in this paper is requirements-oriented. The contest specifications state the purpose of the system and of each feature. A feature interaction occurs when the purpose of the system or of a feature is not satisfied. This can occur directly, in the sense that a property that is a consequence of the purpose is false at a point where it should be true: for example, a call received by a Call Forward Busy Line (CFBL) subscriber is not forwarded. Or it can occur indirectly, in the sense that some basic requirement for the correct functioning of the system is violated. A common example of this second case is the situation where conflicting signals, such as Ring and Busy,

are received by a subscriber or where improper billing occurs, such that a wrong user is billed or the amount billed is incorrect.

More formally, the definition of feature interaction used in this paper is loosely based on a definition proposed by Combes and Pickin [11] and Bouma and Zuidweg [7]. In these papers, the desired system operation is formalized as properties of features. It is said that there is feature interaction if there is a violation of these properties when a new feature is introduced. This definition is closely related to the one proposed in other logic-based approaches, where a feature interaction is seen as an inconsistency in the properties of the system [4][5]. Combes and Pickin's definition is extended here to take into consideration the case where a system property is violated while the properties of the individual features are all present. System properties are hence added to the set of properties to be checked. It easy to see that the properties of the features may be satisfied individually, but system properties are violated if the two features issue conflicting signals (this happens in many situations in the contest's features).

Let S be the specification of a basic telephony system and F_1, F_2, ... F_n be the specifications of features. The integration of the features in the system is denoted as $S \oplus F_1 \oplus F_2 \oplus \ldots \oplus F_n$. Let SP be the logical formula expressing the properties of S and FP_1, FP_2, ... FP_n be formulae expressing the properties of F_1, F_2, ... F_n, with $N \models P$ denoting that a system specification N satisfies formula P (i.e. N is a model of P). Then there is an interaction between features F_1, F_2, ... F_n if $\forall i$, $1 \le i \le n$, $S \oplus F_i \models SP \wedge FP_i$, but $\neg(S \oplus F_1 \oplus F_2 \oplus \ldots \oplus F_i \models SP \wedge FP_1 \wedge FP_2 \wedge \ldots \wedge FP_n)$.

The $\oplus$ operator is an integration operator; the integration being done in different ways in different system architecture, $\oplus$ remains an informal operator [16]. However for each given system architectures, the integration can be formalized.

It should be noted that although our definition of feature interaction is logic-based, our detection method is model-based, i.e. interactions are detected by constructing traces that reveal that the stated feature properties are false in the model.

3 Tools and Methods Used to Find Feature Interactions

3.1 LOTOS

The Language Of Temporal Ordering Specification (LOTOS) was chosen as the Formal Description Technique (FDT) to be used for the specification of the telephone system and its features.

At its origin, LOTOS was designed to be used for the specification of Open Systems Interconnection protocols and services. Subsequently, it has been used to specify many other types of reactive and distributed systems and their protocols with satisfactory results. A very brief description of the language follows, in order to make this paper self-contained.

The main ideas of LOTOS are those of Milner's Calculus of Communicating Systems (CCS) process algebra [28]. However the parallel composition operator

with the multi-way synchronization is inspired by Hoare's Communicating Sequential Processes (CSP) [23]. Further, syntactic and semantic sugar was added to improve the ease of use of LOTOS.

LOTOS represents system behavior by using *actions* and *behavior expressions*. Actions represent basic behaviors of the system, for example, *offhook* or *ring*. There is also an internal (invisible) action written as **i**. Three basic behavior expressions are **stop** (also called *deadlock* or *unsuccessful termination*), **exit** (successful termination) and process instantiation P[G](V), where P is the name of a LOTOS process, G is a set of gate parameters, and V is a set of value parameters. Given behavior expressions B, B1, B2, etc. and actions a, a1, a2, etc., LOTOS operators can be used to construct more complex behavior expressions as follows.

a**;** B: the *action prefix* operator **;** means that the system offers an action a followed by behavior B.

B1 **[]** B2: the *choice* operator **[]** means that the next action offer can be obtained either from B1 or from B2. The other behavior expression is discarded.

B1 **||** B2: the *full synchronization* parallel operator **||** means that a common next action from behavior expressions B1 and B2 has to be found in order for the system to proceed. If such actions exist, they are offered (synchronization) and then the next common action is obtained and so on. Otherwise we have a deadlock, i.e. **stop**.

B1 **|||** B2: the *interleaving* operator **|||** means that at any point, independent actions from behavior expression B1 or B2 are offered.

B1 **|[a1, a2, ... an]|** B2: the *selective synchronization operator* **|[]|** is a generalization of the full synchronization and interleave operators. It means that on actions a1, ... an, B1 and B2 must synchronize; on other actions they interleave.

B1 **[>** B2: the *disable* operator **[>** means that any time during the execution of B1, B2 can take over, thus terminating B1.

B1 **>>** B2: the *enable* operator **>>** means that, provided that B1 has completed successfully (**exit** behavior), B2 can start.

hide a **in** B: this internalizes (transforms to **i**) all offerings of actions a in B

So far, we have discussed *basic* LOTOS, where the notions of action and gate coincide. LOTOS includes also a basic Abstract Data Type formalism, called ACT ONE [15], which is used to represent data abstractions. Data can be associated with actions in two basic ways: **!**, meaning value offer, and **?**, meaning value query. These can be combined with gates to form complex actions, for example

```
switch !subscriber ?destination:destination_sort
```

denotes an action where on gate switch, the current value of the variable subscriber is offered, and a value for destination is queried simultaneously. Offers and queries are called *experiments*. This example shows the abstraction

power of LOTOS, by which it is possible to describe as simple actions system operations that require decomposition in an implementation. An advantage of this abstraction is that the state space is reduced.

A basic concept in process algebraic languages is *expansion*. Any LOTOS behavior expression can be rewritten as an equivalent expression containing only choice, action prefix, and **stop** (although this expression could be infinite). An expanded LOTOS specification represents directly the *labeled transition system* (LTS) [28] of the system in consideration (an LTS is like a Finite-State Machine). Each alternative path in an expanded specification, or each branch in an LTS, represents explicitly a possible sequence of actions in the system. Sequences of visible actions will be called *traces*.

LOTOS has two main assets in the area of specifying telephony systems and their features: it is capable of representing clearly system structure, and it has a good set of validation tools. The tools will be described in following sections. Concerning the representation of telephony system structure, LOTOS operators allow us to represent clearly agents that coexist independently (interleave), communicate (general parallelism), follow each other's actions (enable), or can interrupt each other (disable). Internal system actions can be hidden. Examples of this for specifying the contest's features are presented later in the paper. Several *specification styles* are possible in LOTOS [37]. Some styles are appropriate for representing abstractly system requirements, while others are appropriate for representing implementation structures.

Our University of Ottawa group has a long experience in the use of LOTOS for specifying system features and detecting feature interactions [8][9][16][17][17][19][26][31][32]. Other LOTOS-based approaches to the feature interaction problem have been proposed in the literature: [34] in particular is close to ours, and [35] should be cited from the recent literature. However, for this contest we decided to rethink our approach. New specification structures were developed, appropriate for allowing efficient use of the verification tools that were selected.

3.2 LOTOS Tools

The Canadian-European Eucalyptus toolkit was used to detect the interactions. This toolkit consists of several components. The ELUDO component (*Environnement LOTOS de l'Université d'Ottawa*) [21] was used for the development of the specification; CADP (the *CAESAR-Aldébaran Development Package* of CNRS/IMAG and INRIA in Grenoble [20]) was used to obtain and explore finite LTS representations of the behavior of the system. The LOTOS Laboratory tool (LOLA) [29][30] was used for validating the specification against test cases. All this is discussed further below.

3.3 SDL Tool

The traces converted into the Message Sequence Chart/PR (MSC-PR) notation are processed with the MSC editor of the Telelogic Tau SDT tool [33] to produce conventional MSCs [24]. These are the printed forms by which the interactions are displayed.

3.4 Other Tools

A C preprocessor was used to assemble the specifications for testing. By means of conditional constructs, the specifications can be tailored to the needs of the two architectural models and detection methods (see below), with selective use of the corresponding LOTOS operators.

LOTOS execution traces leading to interactions are transformed into the MSC-PR notation with the assistance of a PERL program.

4 Description of the Approach

The approach can be summarized in the following steps:

- Specification of the System and Features in LOTOS
- Validation of the Specification
- Formalization of Requirements
- Testing
- Report of Interactions

The feature descriptions (in Chisel) were manually analyzed and the constraints that they impose on the telephone system were used to formalize the requirements.

Two different methods were used, which differ from two points of view: the architectural model and the detection method. Only one set of LOTOS specification components was used, but they were combined in different ways in the two cases. These points will be discussed below.

Once an interaction is detected, the Exhibitor tool (part of CADP) is used to provide a trace leading to it.

4.1 Specification of the System and Features in LOTOS

The Chisel notation [2], which was included with the specification of the system and of its features that was provided to the contestants, is a variation of the Finite State Machine formalism. Numbered nodes contain information pertaining to the events that occur, while edges from one node to the next indicate the possible sequences of execution. Transitions sometimes have conditions that need to be valid for a branch to be executed.

Conversion of Chisel diagrams into LOTOS behaviors could be automated. First, a LOTOS process is created for each node of the diagram. The behavior of these processes consists of the events that are executed for the given node. Following these events, an exclusive choice of the instantiation of the processes corresponding to the next possible nodes is added. Conditions restricting a transition in the Chisel diagram become guards of the instantiations of the corresponding processes.

Unfortunately such a straightforward transformation would result in a considerable number of processes, many of which would have only one reference in the specification. A much smaller number of processes can be obtained by a more sophisticated conversion. The enhancement consists in grouping under a single process a number of behaviors, starting from a referred

node, together with all subsequent nodes that are never referred to directly by other diagrams. Furthermore, the processes representing the features are specified in a self-contained manner such that they may be selectively incorporated in pairs into the basic telephone system for testing.

It should be emphasized that the Chisel diagrams were translated faithfully. For example, action names were not modified to include indication of the feature with which they are associated. Such an indication facilitates the reading of the MSCs, but also facilitates the detection of interactions in a way that perhaps could not be used in realistic settings and might change the behavior of the system.

The structure of the LOTOS specification at the highest level is composed of 7 main processes: Switch, Plain Old Telephone System (POTS), Service Control Point (SCP), Operating System (OS), Users, Status Manager (StatMgr) and Clock.

These processes selectively communicate with each other via specific gates, by the general parallel operator. For example, communicating events occur between the Switch and the SCP via gate sw_scp; and between the Switch and the clock via gate sw_clk. The physical requirements are shown in Figure 1 graphically and in Figure 2 in the form of excerpts of the LOTOS specification.

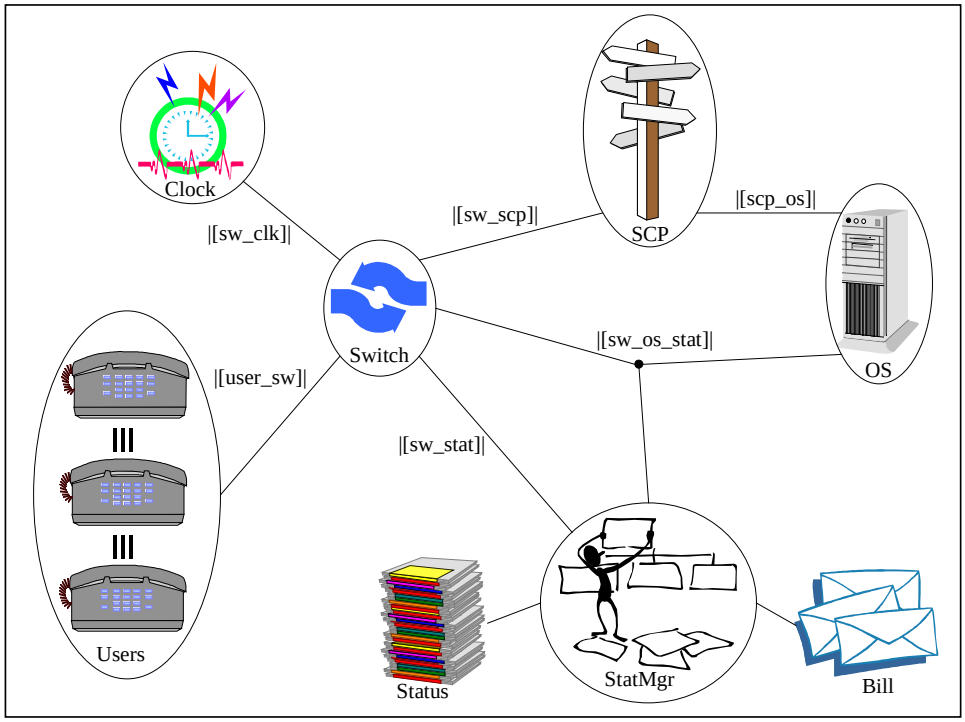

Figure 1: Telephone System

4.1.1 Combining the Features

Since the contest rules required checking interactions by pairs of features, and in addition our two methods (described later) combined features in different ways, we could not work with a single complete specification. A flexible organization was necessary. The system and the features were specified as

independent behaviors. The specifications that were tested were assembled for each pair of features to be tested. These specifications contained the common basic functionality of the telephone system combined with two features. The LOTOS operator used for the assembly depends on the type of combination desired, according to each method, as described below.

Although precise specifications of each of the features were provided to the contestants, there was no precise indication of how the features would interact. Several possibilities existed, and it was decided to experiment with two of them, as follows.

Method 1: Exclusive Alternatives

In the architectural model of Method 1, the features are composed as exclusive alternatives with the LOTOS *choice* ([]) operator. The triggering of a feature may then disable another one. The interactions found by this method are mostly of this basic type: the behavior of a feature cannot be realized because of another feature disabling it. This usually happens after a few events. Hence, traces produced by this method can be fairly short and usually find only one interaction.

Method 2: Interleaving

In the architectural model of Method 2, the features are interleaved (|||), i.e. features execute independently of each other. This generates conflicting signals, which constitute violation of system properties and are detected as such by the watchdog process (see below). At the end, traces and billing records are examined. The traces produced by this method are longer than those produced by the preceding method and often find several interactions.

As mentioned, both methods used the same LOTOS processes, although they were combined in different ways. It should be noted that the possibilities chosen represent two extremes, between which other possibilities exist, by use of the selective synchronization operator |[...]|. However we did not experiment with such additional possibilities.

4.1.2 Main Specification Components

Switch

The switch is the central component of the system. It instantiates POTS processes through interleaving (LOTOS operator |||), in the sense that events of one phone do not synchronize directly with events of the other ones. The number of processes instantiated depends on the needs of the tests to be performed. In order to speed up testing procedures, the number of states of the system is kept to a minimum by instantiating as few POTS processes as necessary.

Plain Old Telephone System (POTS)

This is the specification of the basic telephone system that constitutes the architectural framework to which the features are incorporated. In the

specification structure, it is inside Switch. POTS handles the basic phone networking functionality that enables one user to call another one.

Each POTS instance is assigned to a single user. In general, three POTS instances are used for testing, yielding a telephone system of three users. Note that it is possible to add phones that do not have a corresponding POTS instance by instantiating the system with those phones already in the desired state (e.g. Busy). As many such phones can be added as necessary.

Service Control Point (SCP)

Phones that subscribe to features of the Intelligent Network (IN) have a part of the control taken over by the SCP. For instance, the SCP is in charge of the validation of Personal Identification Numbers (PINs).

```
process System[user_sw,sw_scp,sw_os_stat,sw_stat,scp_os,sw_clk]: noexit :=
    (  (    Users[user_sw]
            |[user_sw]|
            Switch[user_sw,sw_scp,sw_os_stat,sw_stat,sw_clk]
        )
        |[sw_scp,sw_os_stat]|
        (   SCP[sw_scp,scp_os,sw_stat]
            |[scp_os]|
            OS[scp_os,sw_os_stat]
    )  )
    |[sw_stat,sw_os_stat]|
        StatusManager[sw_stat,sw_os_stat](
                (* Initial Set of Phones *)
                    ins(InitialA,ins(InitialB,ins(InitialC,{}))),
                (* Initial Billing Records *)
                    ins(UB(A,{}),ins(UB(B,{}),ins(UB(C,{}),{})))
            )
endproc (* System *)

process Users[user_sw]: noexit :=
        User[user_sw](A)
    ||| User[user_sw](B)
    ||| User[user_sw](C)
endproc (* Users *)

process Switch[user_sw,sw_scp,sw_os_stat,sw_stat,sw_clk]: noexit :=
        (   POTS_1[user_sw,sw_scp,sw_os_stat,sw_stat,sw_clk](A)
        ||| POTS_1[user_sw,sw_scp,sw_os_stat,sw_stat,sw_clk](B)
        ||| POTS_1[user_sw,sw_scp,sw_os_stat,sw_stat,sw_clk](C)
        )
    |[sw_clk]|
        Clock[sw_clk](T1 (* Initial Time *))
endproc (* Switch *)
```

Figure 2: Some Components of the LOTOS Specification

Operating System (OS)

The OS is responsible for billing operations.

Users

The Users are very simple processes: they accept all events that the switch offers. They are marginally useful to the model, and the contest specification provided no information concerning their characteristics.

Status Manager

The Status Manager is a process with dual functionality. First of all, it keeps the information related to the profile of the subscribers. Details like the features subscribed, and the parameters of the features, are contained therein. The Status Manager provides this information on request when it becomes needed, for instance to know if a feature should be triggered, or to make available parameters of features such as the number to which Call Forwarding has been set. Further, this process also serves to keep track of the state of the phones, namely Idle, Busy, Ringing or other. When the state of a phone changes, the Status Manager updates its database accordingly. Events that are used for this purpose are hidden, since they have no meaning outside the system.

In some cases, it is desirable that a phone be in state Busy for the whole duration of a test. This can be accomplished by instantiating a POTS process for that phone, then adding a preamble constraining the phone to become Busy. But an alternative is to instantiate the Status Manager accordingly, i.e. letting it know from the beginning that one of the phones is already busy. This technique greatly reduces the state space of the global system, for verification and validation purposes.

Clock

The Clock process provides global time for the system. However this is not a hard real-time system; only relative times are specified by means of abstract data types and operations on them. Operations that require a timestamp, such as log billing and time-dependent decisions, query the clock to obtain the timestamp. The mathematical operations possible on them are increment and comparison.

4.2 Validation of the Specification

In this phase, the specification was validated to ensure that the various components of the system were properly specified individually. In order to be able to check later that combined features have the desired properties, it is first of all necessary to check that each feature by itself has its own desired properties. Referring to the definition of feature properties in Section 2.2, this means that it is necessary to show that $S \models SP$ and $S \oplus Fi \models SP \wedge FP_i$ for $1 \leq i \leq n$.

An initial validation was performed by executing the specification in step-by-step mode with the tools. This permitted rapid discovery and correction of simple specification errors, such as missing or extra arguments and mismatched data sorts. This validation was done with the ELUDO and LOLA tools. The next step executed test suites derived from the Chisel diagrams. Those sequences may simply be synchronized with the main behavior of the

system. The complete execution of the system when restricted in this manner ensures that the model conforms to the expectations. Any error would cause the sequence to terminate before it could be entirely executed. The specification then would have to be corrected. This process is further explained in the next two sections.

4.2.1 Deriving Test Cases

The test sequences can be derived from the Chisel diagrams by visiting all paths. Starting each time at the root node, all the nodes visited by each path leading to a leaf node are taken into consideration for a given test. The test sequences become the succession of events of all the nodes, in the same order as they were visited, as depicted in Figure 3. Deriving a test sequence for each path leading from the root node to each and every leaf node of the Chisel diagrams provides coverage of all events of the system.

In practice, test sequences are derived individually for each Chisel diagram of the system and of each feature, then they are combined and concatenated according to the next nodes, which vary depending on the features involved.

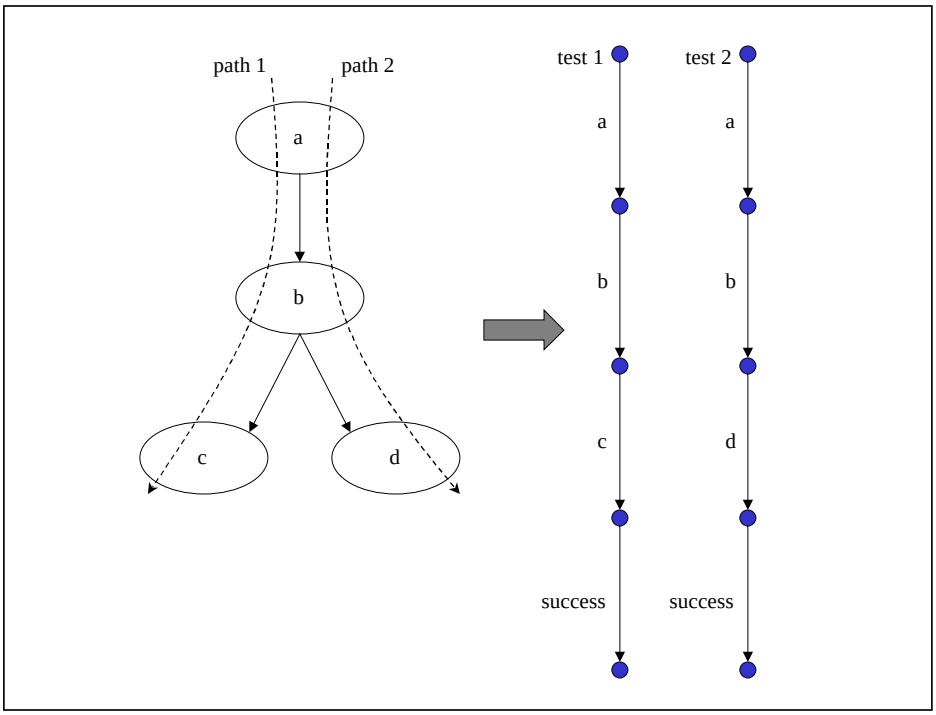

Figure 3: Test Derivation from a Chisel Diagram

4.2.2 Executing Test Cases

An efficient means of running test cases is to add a *success* event at the end of each test case, incorporate the LOTOS behavior in a process called *test*, and use **TestExpand** in **LOLA** with the targeted event *success*.

The outcome of TestExpand will be either [29]:

- MUST PASS: the test case can never fail

- MAY PASS: the test case might fail under some circumstances
- FAIL: the test case fails all the time

4.3 Formalization of Requirements

The most important step in detecting feature interactions consists in defining which properties should be verified for each feature, based on the requirements of each feature, and formalizing them in LOTOS. Note that this is a different step from the one above, because before we were simply checking for correspondence between the LOTOS and Chisel specifications of individual features, while here we are checking that the properties of the features are true after their pair-wise integration.

4.3.1 Scenario Selection and Interaction Description

Scenarios are not used in Method 1. The features are analyzed and the requirements they determine for the system are translated into observers.

In Method 2, the scenario selection process is essentially manual. The process followed to generate test scenarios provides confidence that the set of tests created is fairly complete with respect to the requirements. A single scenario is a formal representation of a requirement. It is manually extracted, specified in terms of a LOTOS behavior and inserted in the specification under test as a LOTOS process.

4.3.2 Method 1: Observers

The violation of feature properties is detected in this method by constructing *observer* processes that monitor the properties of the features under consideration. Observers have been studied in the testing and validation literature [1][12][13], although with different meanings. An observer does not affect the actions of the system, except for the production of an *error* action when a property violation is detected. In other words, observers synchronize with the observed process on all its actions. For actions that are not of interest, the observer does nothing; it simply stays in its current state. For other actions, the observer changes its state, to remember that the action has occurred. Occurrence of certain actions in certain observer states signals to the observer that a property has been violated and results in an error message. The observer then stops.

It is worth noting that the multi-way synchronization mechanism of LOTOS facilitates the implementation of observers. Actions in LOTOS specifications usually result from the synchronization of a number of processes. Observers are additional processes synchronizing on the same events, and several observers can be active at a given time, all synchronizing on the same actions, to monitor different properties or different components of a system.

For example, suppose that it is desired to detect feature interactions related to two features: Calling Number Delivery (CND) and Call Forward on Busy Line (CFBL). Focusing on violation of requirements for the second feature, the requirements are (from the contest specifications):

'CFBL enables a subscriber to have incoming calls that encounter a busy condition to be redirected.'

Furthermore, if the phone to which the call is to be redirected is busy, the calling subscriber will get a busy signal.

In order to check these requirements, an appropriate observer process must be constructed. A system architecture with three phones, A, B, and C, is used. B subscribes to the CFBL feature and specifies C as the phone to which calls should be directed when busy. The observer process ignores all events until it detects the condition: *Dial A B* $\wedge$ *Busy B* $\wedge$ *Idle C.* At this point, it changes state and starts watching for one of two events: *StartRinging D A* $\wedge$ *D*$\neq$*C* $\vee$ *StartRinging C A*. If the first event is encountered, an error is flagged. If the second is encountered, the observer returns to a state where all remaining actions are ignored.

The other case, where C is Busy, can be tested by instantiating the system with C in Busy state and having the observer ignore all events until it detects the condition *Dial A B* $\wedge$ *Busy B* $\wedge$ *Busy C*, then flag an error on *StartRinging D A*.

Observer processes were constructed for many such requirements, but they are combined in an observer for each feature. When features are combined, there is one active observer for each feature**.**

4.3.3 Method 2: Watchdogs

Instead of using observers each one monitoring a specific property, this method uses a Watchdog process, which looks for violation of a number of general system properties, i.e.:

- Double billing for any one call. This is done simply by checking when a new billing action is started.
- Inappropriate operation on user. The watchdog checks for three specific sequences of events:
    - successive rings from different sources
    - screen message and audible ring
    - line busy tone and audible ring

    For example, CFBL may forward a call to C, yet Terminating Call Screening (TCS) rejects the call since the caller is in the Screenlist. The caller has two conflicting signals, *ScreenMessage*, and *Audible_Ringing* from C. This is taken as a feature interaction, because the interleaved integration of these two features gives a user contradictory signals, a violation of the system property by which the system should not generate conflicting signals.

    If any such sequence occurs, an error is signaled, but the system continues, looking for other errors.

In order to obtain finiteness, the description of the system is executed in parallel with other processes that constrain its behavior. These processes consist of the scenarios that describe the test sequences of the features.

4.4 Testing

4.4.1 Method 1

For each pair of features to be tested, a LOTOS specification is assembled with the processes of the basic telephone system, the pair of features and their observers. Synchronized execution of the processes is specified. The expansion of the resulting specification is obtained. The LTS is then searched for traces leading to the *error* event.

The gates involving the observers are the observable gates G of the phone network N[G] and a new gate e used to fire *error* events. All observers O_i[G,e] accept any action on G such that the behavior of N[G] remains unaffected.

For each test involving a pair of features F_1 and F_2, the observers O_1[G,e] and O_2[G,e], corresponding to the requirements of the features F_1 and F_2, are synchronized with the telephone system N[G] as follows:

$$O_1[G,e] \;|[G]|\; O_2[G,e] \;|[G]|\; N[G]$$

The LOTOS behavior resulting from this synchronization accepts any action on the set of gates G and executes the *error* event e when one of the properties is not satisfied.

4.4.2 Method 2

Method 2 does run-time checking of trace properties by using watchdogs and scenarios, and also post-test analysis of billing record properties.

Run-Time Checking

Similarly to Method 1, a LOTOS specification is assembled for each pair of features to be tested. But scenarios are synchronized with the system to guarantee that desired sequences of events are executed. Such sequences are chosen manually for each feature. They are taken from the Chisel diagrams and represent typical sequences of actions in the feature. This has the benefit of concentrating the analysis on meaningful scenarios, while preventing state space explosion.

Observers are replaced by watchdogs. Instead of being dedicated to specific feature requirements like observers and scenarios, watchdogs are general to the system and monitor the same events for all pairs of features:

- contradictory signals
- double billing
- end of scenario reached

In addition to this run-time analysis, each time the end of a scenario is reached, Method 2 also saves data records to be analyzed in a post-test analysis.

Post-Test Analysis

Only Method 2 does post-test analysis. The billing records and traces collected during run-time checking are examined by using a PERL script, as follows:

- Billing records. Feature properties that can be checked on the basis of these records are:
  - in the case of INFB, that the called subscriber has been charged
  - in the case of INFR, that a call started during the redirect time is appropriately forwarded
  - in the case of INCF, that a call to the subscriber is forwarded as desired.
- In the case of certain features, at the end of the run, traces are examined. Typical things that are checked are:
  - if the caller's number is displayed in the case of CND
  - in the case of CC and INTL, that a valid PIN was entered

This post-test analysis technique proves particularly useful when data is involved but it could be tailored to traces too.

4.5 Report of Interactions

4.5.1 Description of the Findings

The tests that run on the system report on traces that lead to interactions. Those are detected by checking the behavior and the data of the system:

Behavior

An interaction may cause the disruption of the behavior (traces) of the system. Two types of disruption are monitored. Roughly speaking, in the first method the disappearance of an expected event points to an interaction. In the second method, the appearance of an unexpected event is reported as an interaction.

Data

During testing, values agreed upon by events are logged. Afterwards, these logs are analyzed to detect inconsistencies caused by interactions. For instance, some features state that explicit billing norms have to be applied. The analysis of the logs can detect billing inconsistencies that would have resulted because of interactions. As mentioned, this analysis is done only in Method 2.

In both cases, the net result is a trace leading to an interaction.

4.5.2 Presentation of the Results

A requirement of the contest was to present the traces leading to interactions in the form of MSCs. Starting from execution traces, this is obtained in two steps.

Traces leading to interactions are translated into the MSC-PR notation of the Telelogic SDT tool using an in-house PERL script.

The rules of the contest dictate the direction of the events. For instance, it is stated that "Off-hook X:Address" goes from entity User to entity Switch; "Announce X:Address M:Message" goes from entity Switch to entity User. Since LOTOS expresses events and values exchanges in an undirected manner, a gate naming scheme was selected to ease the translation of traces into MSCs. Events that exchange experiments do so through gates that bear the name of the involved components. For instance, communicating events between a *user* and the *switch* go through the gate (or interface) *user_sw*. Furthermore, specific components may be identified from experiments. For example, the event **`user_sw !DialTone !A`** refers to the *switch* sending a *DialTone* signal to *user(A)*.

The MSC entities used are those specified in the contest rules. The LOTOS gate names disappear and experiments become the *messages* of the MSC transitions.

The additional information necessary is the order of the MSC transitions. These evolve in time in the same order as the events found in the traces. Due to the semantics of MSC-PR, it suffices to use an integer counter that is incremented after the translation of each event.

The MSC Editor of the Telelogic SDT tool is then used for the production of the conventional MSCs required by the contest. This MSC Editor supports two notations of MSC: PR and GR. It is just a matter of importing the traces in the internal PR notation and exporting them in the external GR format. Figure 4 shows a typical MSC obtained by using this technique. Each MSC is annotated with the description of the test environment, i.e. initial states of the telephone systems involved and active features.

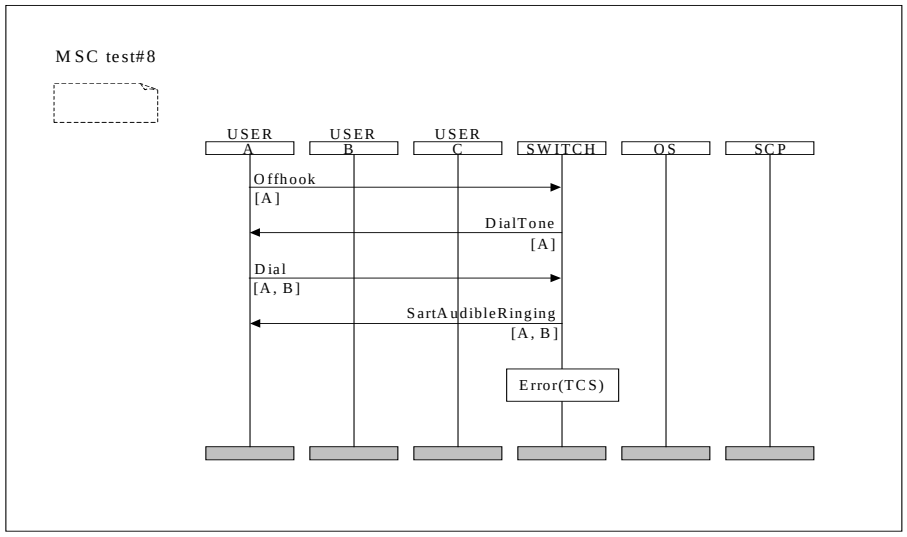

Figure 4: Message Sequence Chart

Tables 1 and 2 provide an overview of the method coverage in terms of numbers of interactions found for all possible pairs of features. It should be noted that many of these interactions (especially for Method 2) were not recognized as valid by the contest committee. They depend on certain assumptions on the architecture of the system, and mostly relate to conflicting signals that result from the interleaving integration of the features (Sections 4.3.3 and 4.4.2), which apparently was not an intended integration method. Cases of interactions of a feature with itself were excluded in our analysis, since we considered these to be design errors rather than interactions.

	CFBL	CND	INFB	INFR	INTL	TCS	TWC	INCF	CW	INCC	RC	CELL
CFBL	0	1	2	2	-	1	1	0	-	1	0	0
CND		0	2	2	-	1	1	2	-	1	1	0
INFB			0	1	-	2	1	2	-	1	1	1
INFR				0	-	0	0	0	-	1	1	1
INTL					-	-	-	-	-	-	-	-
TCS						0	1	1	-	1	1	0
TWC							0	1	-	0	0	0
INCF								0	-	1	1	1
CW									-	-	-	-
INCC										0	0	1
RC											0	0
CELL												0

Table 1: Summary of FI detected with Method 1

	CFBL	CND	INFB	INFR	INTL	TCS	TWC	INCF	CW	INCC	RC	CELL
CFBL	0	3	5	11	0	4	1	8	1	1	1	3
CND		0	1	9	0	2	1	9	1	1	2	0
INFB			0	12	0	4	1	10	1	1	2	2
INFR				0	0	8	2	8	1	1	2	3
INTL					0	0	2	0	0	0	2	0
TCS						0	2	5	1	1	2	0
TWC							0	2	0	0	0	3
INCF								0	1	2	2	4
CW									0	0	0	2
INCC										0	0	2
RC											0	0
CELL												0

Table 2: Summary of FI detected with Method 2

5 Related Information

The forthcoming Master's theses of P.Harnois and Q.Fu will explain in detail the two methods used. They will be placed on the Web page that already contains our contest submission: http://www.csi.uottawa.ca/~lotos/

6 Conclusions

The contest committee identified 97 pairwise interactions in the set of 12 features. Although logically well constructed, Method 1 does not appear to be strong. It discovered only 39 interactions overall. It excluded CW and INTL and it discovered only 31 of the 75 remaining interactions (recall that interactions between a feature and itself were not considered by our group). 8 additional interactions were detected by this method, which were not in the committee's list. Method 2 discovered 154 interactions in all, of which 73 were in the committee's list. Among the 24 in the list that were not detected, 12 are between a feature and itself and 12 involve TWC or CW or both, and have at least four users. Neither method succeeded in finding conflicts in cases where 4 users are involved. Shortcomings seem to have been due more to lack of time and computing power, or perhaps lack of optimization in the algorithms, than to intrinsic limitations of the methods. This seems to be particularly true for Method 1.

More interactions could be detected by combining the features differently, such as combining more than two of them at a time. Checking additional feature properties is another avenue. The method of generating global LTSs also showed its limits when the analysis focussed on features that generate very large state spaces, such as TWC and CW. At that point, it was no longer possible to expand the whole system behavior, and it became necessary to start looking at parts of it only. A method based on the use of heuristically generated test cases probably would have been more productive [14].

Detection times varied greatly, from a few seconds for the simplest features combinations to about 12 hours in cases where the most complex features (TWC and CW) were exercised against all others. These times were on low-end SPARC stations.

The use of a common specification of the system under test by the two different methods created some difficulties at the beginning. But in the end, it proved to be an efficient route. Other detection methods could have been tried without multiplying the total effort.

The detection of feature interactions would be facilitated greatly if the systems and features were specified formally, in a format suitable for automated analysis. This is because one of the key operations resides in the formalization of the requirements, which includes formalization of the system architecture. The informal nature of the descriptions increases the risks of misinterpreting the feature properties and slows the process, not to forget the possibility of missing some properties. As mentioned, important decisions had to be taken in the process of translating Chisel into LOTOS, and probably different participants took different decisions. Although this problem would be eliminated by using an FDT in the formal specifications of the system to start with, this would establish a major advantage for contestants choosing the same FDT, and other contestants would feel discouraged from entering. The best compromise seems to be striving for the greatest possible precision in the informal specifications.

The need to respect the deadlines of the contest did not allow us to try and integrate better the two methods that were used, or to integrate them with use case-based techniques under development in a related project [3][10]. This is the subject of current work in our group.

These methods appear to be ready for experimental use in industrial environments, and in fact they are being pursued with two industrial collaborators.

Acknowledgements

We are grateful to the organizers of the contest for presenting us with such an interesting challenge. We are also grateful for financial support by Motorola, the Natural Sciences and Engineering Research Council of Canada, Nortel, and Communications and Information Technology Ontario. Laurent Andriantsiferana provided the first draft of the LOTOS specification. Daniel Amyot contributed with many helpful suggestions and comments.

7 References

[1] Aggoun, I., and Combes, P. Observers in the SCE and the SEE to Detect and Resolve Service Interactions. In: Dini, P., Boutaba, R., and Logrippo, L. Feature Interactions in Telecommunication Networks IV. IOS Press, 198-212.

[2] Aho, A., Gallagher, S., Griffeth, N., Scheel, C., and Swayne, D. Sculptor with Chisel: Requirements Engineering for Communications Services. In Kimbler, K. and Bouma, W. (eds) In: Kimbler, K., and Bouma, L.G. Feature Interactions in Telecommunications Systems V, IOS Press, 1998, 45-63. http://www-db.research.bell-labs.com/user/nancyg/sculptor.ps

[3] Amyot, D., Buhr, R.J.A., Gray, T., and Logrippo, L. Use Case Maps for the Capture and Validation of Distributed Systems Requirements. To appear in: ISRE'99, Fourth International Symposium on Requirements Engineering, Limerick, Ireland, June 1999.

[4] Blom, J., Johnsson, B., Kempe, L. Using Temporal Logic for Modular Specification of Telephone Services. In: Bouma, W., and Velthuijsen, H. (Eds.) Feature Interaction in Telecommunications Systems. IOS Press, 1994, 197-216.

[5] Blom, J. Formalisation of Requirements with Emphasis on Feature Interaction Detection. In: Dini, P., Boutaba, R., and Logrippo, L. Feature Interactions in Telecommunication Networks IV. IOS Press, 61-77.

[6] Bolognesi, T. and Brinksma, E. Introduction to the ISO Specification Language LOTOS. Computer Networks and ISDN Systems14 (1987) 25-59.

[7] Bouma, W. and Zuidweg, H. Formal Analysis of Service/Feature Interaction using model checking. TI-PU-93-868, PTT Research, Leidschendam, The Netherlands, 1993.

[8] Boumezbeur, R.. and Logrippo, L. Specifying Telephone Systems in LOTOS and the Feature Interaction Problem. International Workshop on Feature

Interactions in Telecommunications Systems (St.Petersburg, Fla., 1992), 95-108.

[9] Boumezbeur, R., and Logrippo, L. Specifying Telephone Systems in LOTOS and the Feature Interaction Problem. IEEE Communications Magazine 8 (1993) 38-45.

[10] Buhr, R.J., Amyot, D., Elammari, M., Quesnel, D., Gray, T., Mankowski, S. Feature Interaction Visualization and Resolution in an Agent Environment. In: Kimbler, K., and Bouma, L.G. Feature Interactions in Telecommunications Systems V, IOS Press, 1998, 135-149.

[11] Combes, P., and Pickin, S. Formalization of a user view of network and services for feature interaction detection. In: Bouma, L.G. and Velthuijsen, H. (eds.) Feature Interactions in Telecommunications Systems. IOS Press, 1994, 120-135.

[12] Diaz, M., Juanole, G., Courtiat, J.-P. Observer – A Concept for Formal on-line Validation of Distributed Systems. In IEEE Transactions on Software Engineering, Vol. 20, N. 12, 900-913 (December 1994). Also LAAS 89.266.

[13] Dssouli, R. and Bochmann, G. V. Error Detection with Multiple Observers, Protocol Specification, Testing and Verification. V, Elsevier Science Publishers B. V., 1986.

[14] du Bousquet, L., Ouabdesselam, F., Richier, J.-L., Zuanon, N. Incremental Feature Validation: A Synchronous Point of View. In: Kimbler, K., and Bouma, L.G. Feature Interactions in Telecommunications Systems V, IOS Press, 1998, 262-275.

[15] Ehrig, H. and Mahr, B. Fundamentals of Algebraic Specifications 1. Springer, Berlin, 1985.

[16] Faci, M., and Logrippo, L. An Algebraic Framework for the Feature Interaction Problem. Proc. of the 3rd AMAST Workshop on Real-Time Systems, Salt Lake City, 1996, 280-294.

[17] Faci, M., and Logrippo, L. Specifying Features and Analyzing their Interactions in a LOTOS Environment. In: L.G. Bouma and H. Velthuijsen (eds.) Feature Interactions in Telecommunications Systems. IOS Press, 1994 (Proc. of the 2nd International Workshop on Feature Interactions in Telecommunications Systems, Amsterdam) 136-151.

[18] Faci, M., Logrippo, L., and Stépien, B. Formal Specification of Telephone Systems in LOTOS: The Constraint-Oriented Approach. Computer Networks and ISDN Systems 21 (1991) 53-67.

[19] Faci, M., Logrippo, L., and Stépien, B. Structural Models for Specifying Telephone Systems. Computer Networks and ISDN Systems 29 (1997) 501-528.

[20] Fernandez, J.-C., Garavel, H., Kerbrat, A., Mateescu, R., Mounier, L., and Sighireanu, M. CADP (CAESAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, Proceedings of the 8thConference on Computer-Aided Verification (New Brunswick, New Jersey, USA), vol. 1102 of Lecture Notes in Computer Science, 437-440. Springer Verlag, August 1996.

[21] Ghribi, B., and Logrippo, L. A Validation Environment for LOTOS. In: Danthine, A., and Leduc, G. (eds.) Protocol Specification, Testing, and Verification, XIII (Proc. of the 13th International Symposium on Protocol Specification, Testing, and Verification, organized by IFIP WG 6.1, Liège) North-Holland, 1993, 93-108.
[22] Griffeth, N.D., Tadashi, O., Grégoire, J.-C. and Blumenthal, R. First Feature Interaction Detection Contest. In: Kimbler, K., and Bouma, L.G. Feature Interactions in Telecommunications Systems V, IOS Press, 1998, 327-359. http://www.tts.lth.se/FIW98/contest.html
[23] Hoare, C.A.R. *Communicating Sequential Processes.* Prentice-Hall, 1985.
[24] International Telecommunications Union, Telecommunication Standardization Sector (ITU-T) Z.120. Series Z: Programming Languages. Message Sequence Charts (1996).
[25] ISO, International Organization for Standardization - Information Processing Systems, Open Systems Interconnection. LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, IS 8807, 1989 (E. Brinksma, ed.).
[26] Kamoun, J., and Logrippo, L. Goal-Oriented Feature Interaction Detection in the Intelligent Network Model. In: Bouma, W. and Kimbler, K. (Eds.) Feature Interactions in Telecommunications, V, IOS Press, 1998, 172-186 (IEEE Conference).
[27] Logrippo, L., Faci, M., and Haj-Hussein, M. An Introduction to LOTOS: Learning by Examples. Computer Networks and ISDN Systems 23(5) (1992) 325-342. Errata in 25(1) (1992) 99-100.
[28] Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
[29] Pavón, S. and Llamas, M. "The testing Functionalities of LOLA". In: J. Quemada, J. A. Mañas and E. Vázquez (Eds), Formal Description Techniques, III, IFIP/North-Holland, 1990, 559-562
[30] Quemada, J., Pavón, S., Fernández, A. Transforming LOTOS Specifications with LOLA - The Parameterized Expansion. In Turner, K. (Ed.) Formal Description Techniques I, North-Holland 1988, 45-54.
[31] Stépien, B., and Logrippo, L. Feature Interaction Detection Using Backward Reasoning with LOTOS. In: Vuong S. (ed.)Protocol Specification, Testing, and Verification, XIV, 1995, 71-86.
[32] Stépien, B., and Logrippo, L. Representing and Verifying Intentions in Telephony Features Using Abstract Data Types. In: Cheng, K.E., and Ohta, T. (Eds.) Feature Interactions in Telecommunications, III. IOS Press, 1995, 141-155.
[33] Telelogic (1998). Telelogic ORCA and SDT 3.4, Telelogic AB, Box 4128, S-203 12 Malmoe, Sweden, 1998.
[34] Thomas, M. Modelling and Analysing User Views of Telecommunications Services. In: Dini, P., Boutaba, R., and Logrippo, L. Feature Interactions in Telecommunication Networks IV. IOS Press, 1997, 168-182.
[35] Turner, K.J. (Ed.) Using Formal Description Techniques - An Introduction to ESTELLE, LOTOS and SDL. Wiley, New York, 1993.

[36] Turner, K.J. Validating Architectural Feature Descriptions Using LOTOS. In: Kimbler, K., and Bouma, L.G. Feature Interactions in Telecommunications Systems V. IOS Press, 1998, 247-261.
[37] Visser, C., Scollo, G., van Sinderen, M. and Brinksma, E. Specification Styles in Distributed Systems Design and Verification. Theoretical Computer Science 89 (1991) 179-206.