Defining Features for CSP: Reflections on the Feature Interaction Contest

Malte Plath[1] and Mark Dermot Ryan[1]

School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, England. mcp,mdr@cs.bham.ac.uk

1 Introduction

The second Feature Interaction Contest was held in conjunction with the 6th International Workshop on Feature Interactions in Telecommunication and Software Systems (FIW'00) [1]. The aim of the contest was to compare various methods and tools for detecting feature interactions. To enable a comparison, the contest's objective was to detect interactions among a given set of features for a given telephone system. The contest instructions contained detailed (albeit imprecise) specifications of the base system and twelve features.

In this paper we describe how we tackled the contest and the experiences we had in the process. In the following two sections we briefly describe how the contest was set out and what implications that had for detecting feature interactions. We then detail the methods we used and the results we obtained before summing up our experiences in section 7.

We used a combination of two techniques: static (syntactic) analysis and model checking using the model checker FDR [5]. While we had initially planned to integrate the features into the base system and then to detect interactions by model checking, it became clear very quickly that a simple syntactic analysis of the features was sufficient to detect a large proportion of the interactions. Hence, in this paper, we first describe this syntactic method (section 4), before showing how we used model checking to find more interactions (section 5). Finally, we give an overview of the full results of our analysis in section 5.4; a complete and detailed description of the results can be found in [4].

We are delighted to report that our entry won the contest in the category of two-feature interactions. For a summary of the results produced by each of the contestants, see [2].

2 The Contest Model

The base system for the contest is a model of the plain old telephone service (POTS) given as a labelled transition diagram (Figure 1) for a single telephone line (basic call model, BCM) and a description of the network architecture, a simple star network, with a single switch relaying information

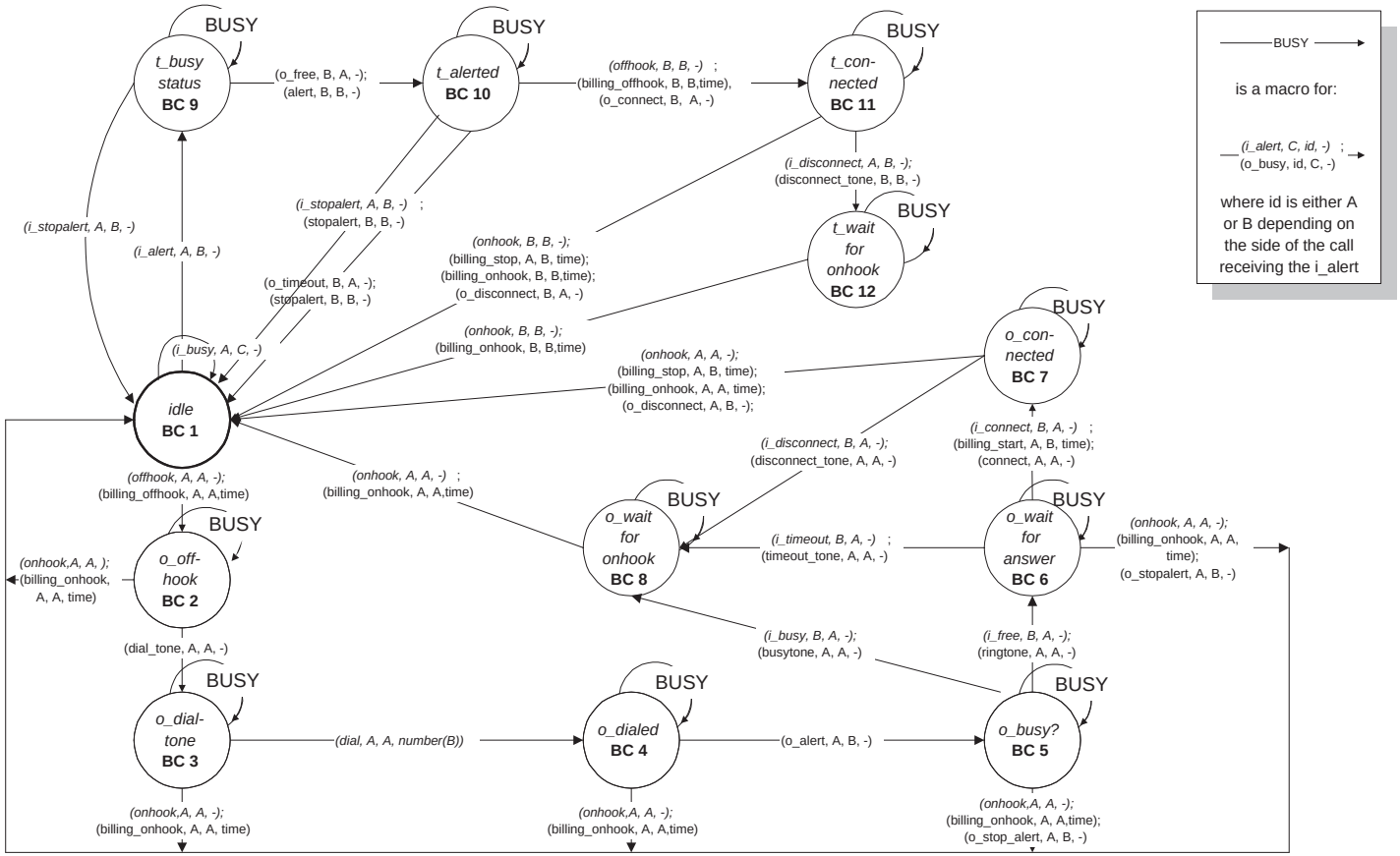

Fig. 1. The Basic Call Model [3]

between lines, and a billing database, of which no details are given in the contest instructions [3].

Most transitions are labelled with multiple messages, some of which denote local events, such as user input or signals to the user, others stand for messages sent to or received from other phones in the network; finally, there are billing messages.

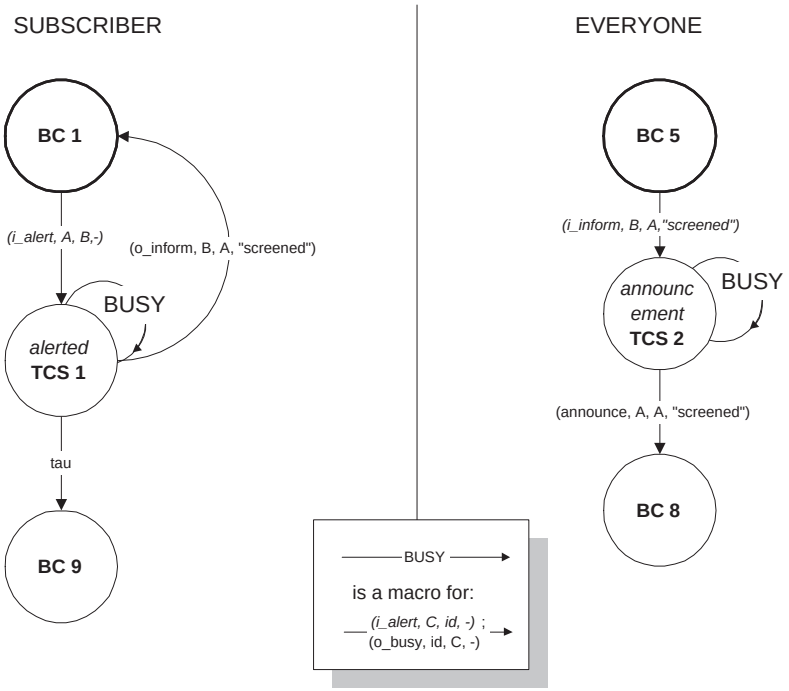

Fig. 2. Feature Terminating Call Screening [3]

The features, too, are given as labelled transition diagrams, with some BCM states and (usually) several new states (feature states). Feature integration is defined by adding or replacing, for a given basic call state, the transitions which are given in the feature definition. Here, a transition is replaced if the feature's transition has the same triggering event as the corresponding transition in the basic call; otherwise it is added. In the diagram for the feature Terminating Call Screening (Figure 2), for example, the transition from the idle state to BC9 of the basic call is replaced by the left diagram (for the subscriber), while in all basic call processes a new transition is added to the state BC5 ("o_busy?"). This transition is labelled with the message (i_inform,B,A,"screened") and leads to the new state TCS2.

The contest comprised the following features:

CFB Call Forward on Busy – divert calls when the line is busy
CNDB Calling Number Delivery Blocking – do not display the caller's number to the callee
CT Call Transfer – transfer an active call to another telephone
CW Call Waiting – allow subscriber to take a second call and switch between first and second call
GR Group Ringing – make three telephones ring for an incoming call to one of them, stop the ringing when one of them is answered
RBWF Ring Back When Free – ring back callers that got the busy tone (also known as Automatic Call Back)
RC Reverse Charging – callee pays for call
SB Split Billing – callee pays part of the call
TCS Terminating Call Screening – block calls from certain phones to the subscriber's phone
TL TeenLine – a PIN must be entered before calls can be made
TWC Three-Way Calling – allow the subscriber to initiate a second call and let all three parties talk with each other (also known as Conference Call)
VM Voice Mail – callers can leave a message if the phone is not answered (also known as CallMinder)

3 Observations

Analysing the feature definitions, we realized that the model given in the contest instructions was extremely prone to interactions. With a little practice we could anticipate many interactions by just looking at the features' diagrams. To explain how this worked, we classify interactions according to their causes:

1. one feature overrides trigger of the other feature (e.g. TCS & GR)
2. one feature bypasses trigger of the other feature (e.g. TCS & CW)
3. one feature sends a message that the other feature cannot process (e.g. TCS& TWC)
4. other causes (e.g. TCS & RBWF, RC & RBWF)

In the first two cases one of the features is not invoked when it should be; in the latter two, both features are active but interfere with each other in some way.

Interactions of type 1 are the easiest to detect: both features are triggered in the same (basic call) state by the same event. Due to the method of feature integration, the feature integrated later overwrites the transition introduced or altered by the earlier one. Interactions are usually serious because they are due to some fundamental incompatibility. In this case the interaction can only be resolved by limiting or refining the behaviour of one or both features.

Type 2 interactions are also not hard to recognize. Whenever a triggering message for one feature can occur in a feature state of another feature, this

may lead to the former feature not being invoked even though it should. In the contest, all interactions of this type occurred because of the method of feature integration, and could clearly be avoided by a design that did not put so much emphasis on states. It is very likely that some of the type 2 interactions would then become type 4 interactions, i.e. the type 2 conflict indicated a serious interaction but for trivial reasons.

Interactions of the third type occur when a feature introduces a new message to the system, which may be received in a feature state of another feature. Since only basic call states are altered to be able to handle the new message, a feature on the receiving end will not be able to handle the new message. For these interactions, the same observations hold as for type 2: in a better, feature-oriented architecture, they would not occur.

The remaining interactions (type 4) are particularly interesting from the point of view of feature interaction detection. They indicate deeper problems in the way that features affect processing and distribution of information in the telephone network. There are no generic tests to detect such interactions, however they violate some 'sensible assumptions' about the working of the system. Furthermore, they only surface in the actual execution of the system, so they are not amenable to static analysis methods, such as proposed in the following section.

4 Static Analysis

The observations above prompted us to look for syntactic criteria to detect the first three classes of interactions. We introduce the following notation.

Let S be the set of all states (both basic call and all possible feature states), and E the set of all events (messages). To simplify the presentation we omit the subscriber parameters unless absolutely necessary.[1] Feature n is denoted by F_n; Sys may be a feature, or the Basic Call possibly augmented with a number of features.

$\text{Trans}(Sys) \subseteq S \times E$	one pair (s, e) for each transition in Sys, such that in state s, event e commences the transition
$\text{Triggers}(F_n) \subseteq S \times E$	the basic call states in which the F_n can be triggered, with the corresponding triggering event
$\text{Msgs}(Sys) \subseteq E$	all i_xxx and o_xxx messages that appear on transitions of Sys (ignoring the prefixes "o_" and "i_")

Note that $\text{Msgs}(Sys)$ contains information about all transitions, while $\text{Triggers}(F_n)$ only records the new or altered transitions from basic call states introduced by F_n. Let $\sigma(s, e) = s$ and $\varepsilon(s, e) = e$ denote the projections onto the first and second component, i.e. states and events, respectively.

[1] To be fully correct, one would need to take account of several variables, at least subscriber and partner, for each state.

Example 1. For the Terminating Call Screening feature we have:

$$\begin{aligned}\text{Trans(TCS)} = \{&(\text{BC1}, \text{i\_alert}), (\text{BC5}, \text{i\_inform}),\\ &(\text{TCS1}, \text{i\_alert}), (\text{TCS1}, \text{o\_inform}), (\text{TCS1}, \text{tau}),\\ &(\text{TCS2}, \text{announce}), (\text{TCS2}, \text{i\_alert})\}\end{aligned}$$

and

$$\text{Triggers(TCS)} = \{(\text{BC5}, \text{i\_inform}), (\text{BC1}, \text{i\_alert})\}$$

4.1 Interactions

With this notation, the following criteria characterise the first three classes of interactions given on page 4.

1. Later feature overrides earlier one:
   $\exists\, (s, e) \in \text{Triggers}(F_1) \cap \text{Triggers}(F_2)$
2. One feature bypasses a trigger of another feature:
   $\exists\, e \in \varepsilon(\text{Triggers}(F_1)) \cap \varepsilon(\text{Trans}(F_2) \setminus \text{Triggers}(F_2))$
3. F_1 may send a message which F_2 cannot handle ("Message not understood"):
   $\exists\, e \in \text{Msgs}(F_1) \setminus \text{Msgs}(F_2)$

These criteria will flag some potential interactions that *cannot* occur during normal execution of the featured system. In our experience with the contest model, however, they were quite accurate.

It is important to note that these criteria point to *causes* for interactions. There may be more than one actual, observable interaction for the same event or state-event pair satisfying one of the criteria.

Example 2. Since the Group Ringing feature, like TCS, is also triggered by an "i_alert" message in state BC1, we get a type 1 interaction:

$$\text{Triggers(TCS)} \cap \text{Triggers(GR)} = \{(\text{BCi}, \text{i\_alert})\}$$

Thus we can conclude, when TCS and GR are added to the same subscriber's telephone, whichever feature is added later will disable the earlier one, since it will override the triggers for the feature added earlier.

Furthermore we also get a type 3 interaction, since

$$\text{o\_inform} \in \text{Msgs(TCS)} \quad \text{but} \quad \text{i\_inform} \notin \text{Msgs(GR)}.$$

Note that a message "o_xxx" becomes "i_xxx" for the receiving phone, hence the third criterion says, *if* there is a situation in which the TCS feature sends an "o_notify" while the other phone is in an GR feature state, then that message cannot be processed (by the GR feature), leading to undefined behaviour. As our method stands at the moment, it is up to the user to check whether such a situation can actually arise. With these two features, the interaction can happen, in the following scenario. Assume that subscriber A has GR, and B and C are in the 'group'. If, for example B subscribes to TCS

and has A on its screening list, then a call to A will result in an i_alert from A to B, which B will answer with (o_inform,B,A,"screened"). At this point A will be in a state introduced by Group Ringing, and will not be able to deal with that message.

Roughly 90% of the interactions we detected were discovered by applying these simple criteria. Yet, the interesting interactions are those that are not detected by these tests. The way that the contest model was set up, there were very few of these 'tricky' interactions, since most features clashed quite badly on the simple criteria. Assuming a good software engineering approach, though, which would take account of the 'easy' interactions and aim to avoid them in the first place, the 'tricky' ones, i.e. those not detected by our syntactic criteria, become crucial. At this point, simulation, testing and model checking come into their own again, because the interactions that cannot easily be detected by syntactic criteria are likely to show up only in longer runs of the system.

5 Modelling the system in CSP

5.1 A Network of Basic Call Processes

Modelling a single basic call in CSP is very simple: the states become CSP processes and all messages become events, the send and receive operations fit very nicely. A transition with multiple messages is split up into a sequence of events.

The problems start, however, when one composes several such basic call processes into a network. Now, the different processes may 'de-synchronize' since there is nothing to enforce the atomicity of the transitions in the diagram. Hence one line could embark on a transition if this was triggered by a local event and thereby prevent another line from sending a message to it. In other words, the naive, literal translation from the contest instructions allowed some internal choice between transitions that need to synchronize with other transitions, leading to possible deadlocks.

It turned out, though, that this problem was not too hard to solve: it was necessary to reorder the events labelling each transition, so that all external communication events came first on the respective transitions. Eventually, all transitions started with one input or output event, if they contained one at all, followed by only internal (local) events.[2] In CSP terms this meant offering all events that the line processes had to synchronize on in external choice constructs.

[2] For some features this meant introducing extra states and transitions, since some of them used multiple send operations in one transition.

```
-------------------------------------------------------------------
-- Feature Terminating Call Screening
-------------------------------------------------------------------
-- This is designed for a maximum of four phones: any phone above C
-- gets $TCSSetD as its screening set (only relevant if in subscriber
-- set $TCSSubs).
-------------------------------------------------------------------
$TCSSubs = {C}
$TCSSetA = {}
$TCSSetB = {}
$TCSSetC = {A}
$TCSSetD = {}

screen(A) = $TCSSetA
screen(B) = $TCSSetB
screen(C) = $TCSSetC
screen(_) = $TCSSetD

BCM(state:{idle},x,x)  +=  if member(x,$TCSSubs)
                           then i.m_alert?y:Lx({x})!x!none -> TCS1(x,y)
                           else BCM(idle,x,x)

TCS1(x,y) =    (if member(y,screen(x))
                then o.m_inform.x.y.screened -> BCM(idle,x,x)
                else BCM(t_busys,x,y))
            [] i.m_alert?z:Lx(x)!x!none -> o.m_busy.x.z.none -> TCS1(x,y)

BCM(state:{o_busyp},x,y)  +=  i.m_inform.y.x.screened -> TCS2(x,y)

TCS2(x,y) =    announce.x.tcs -> BCM(o_wait_onhook,x,y)
            [] i.m_alert?z:Lx(x)!x!none -> o.m_busy.x.z.none -> TCS2(x,y)

-- end of feature -----------------------------------------------
```

`BCM(state,subscriber,partner)` denotes a basic call state,
`$TCSSubs` is the set of subscribers to TCS (a parameter of the feature),
`$TCSSetA` through `$TCSSetD` represent the screening lists of the respective subscribers (feature parameters).

Fig. 3. CSP_M^{FC} code for Terminating Call Screening

5.2 A Feature Construct for CSP

To automate feature integration we extended the CSP_M syntax with a simple feature construct for the textual representation of the feature diagrams. (CSP_M is the subset of CSP accepted by the FDR tool.) We will denote the extended language by CSP_M^{FC}.

A feature definition may contain any number of standard CSP_M definitions. These will simply be added to the base system, and the user is responsible for avoiding name clashes (re-definitions).

On top of plain CSP_M, there are two new constructs:

- Transitions can be added or replaced by means of special definitions with the following syntax:
  $process\,(p_1, \ldots, p_n)$ `+=` *new definition*

where *process* is a process name from the base file. The actual parameters can be 'captured' and are referred to by $p_1, \ldots, p_n$ in the *new definition*. It is also possible to restrict the range of parameters in certain cases.[3]

- Feature variables or parameters can be defined. Any name prefixed with `$` is interpreted as a feature variable. Such variables can be assigned values in the feature file or on the command line when invoking the integrator (see below). All (non-defining) references in the feature file are textually replaced by the value thus given.

5.3 Feature Integration

We wrote a Python script (which we call the "feature integrator") which combines a feature definition (written in CSP_M^{FC}) and a base system in a CSP_M file and produce a new CSP_M file defining the featured system. Features may be parametrized, and the feature integrator allows us to set values for the parameters at integrate time, overriding any default values given in the feature definition.

The current prototype relies quite heavily on syntactic conventions in our POTS model, e.g. `+=` definitions only work for BCM states and the number of parameters is fixed. On the other hand this enabled us to substitute sensible defaults for the 'don't care' place holder (`_`).

After integrating a feature, the resulting CSP_M file can be used as the base file for further feature integrations or, of course, be analysed using ProBE and FDR2 [4] [5].

5.4 Detecting Interactions

To detect feature interactions in the CSP_M model, we applied several techniques.

First we used FDR2 to check the featured systems for deadlock. Obviously the telephone system should never deadlock – it must always be possible for every line to get back to the initial state. However, since we were working on a system with four phones, we could not detect local deadlocks, i.e. situations in which one or more phones had no more enabled transitions but where there was at least one which could still move – even if that only meant going offhook and onhook repeatedly. If FDR2 allowed the user to impose fairness constraints, such situations could be detected.

Secondly, we explored the behaviour of the system using ProBE, to test if the featured system *could* behave in the intended way. ProBE allows the user at every step to choose one of the enabled events and thus to simulate a

[3] This is rather an *ad hoc* solution to deal with the specific form our POTS model takes.

[4] `www.formal.demon.co.uk/FDR2.html`

run of the system. Hence it cannot be used to verify the absence of undesirable behaviour.

This is where FDR2 comes in again. While the previous two techniques are mainly for debugging, FDR2 can explore all possible executions of a system. The central method used for this is refinement checking in one of three models.[5] However, since features both add and remove behaviours, refinement is not such a useful notion.

Instead we coded desirable or undesirable patterns of behaviour as observer processes and composed them with the system in question, synchronising on the events that were relevant for the behaviour we wanted to test for. The observer processes were designed in such a way that the presence of the behaviours they represented lead to a deadlock in the composite system. This allowed for exhaustive checking of properties.

5.5 Limitations

The expressive power of observer processes in CSP is rather weak. Unlike 'never claims' in SPIN/PROMELA, which uses Büchi automata to deal with infinite behaviours (e.g. liveness properties, in general recognition of ω-regular traces), observer processes in CSP can only be used to detect the presence of finite traces.

As with SPIN, some observer processes led to a huge blow-up in the state space, which made it impossible to verify the properties.[6]

Another drawback of using observer processes is that they need to be coded by hand which is a rather error-prone procedure. Contrast this with expressing a property in temporal logic, with subsequent automatic translation to the corresponding Büchi automaton.

This list of drawbacks might give the impression that the model-checking approach is deeply flawed. However, we would like to point out that the fact that a simple static analysis detected such a large proportion of the feature interactions hinged on the specific model the contest defined. Also, the expressiveness of model-checking languages varies greatly, with regards to both the description of models and the properties that can be checked. Indeed we would argue that model checking is still an invaluable tool in proving the absence of unwanted behaviour, and in finding deep errors.

6 Feature Interactions

A full list of the (two-way) interactions we detected among the twelve features of the contest, with explanations can be found in [4]. Table 1 may give an impression of the sheer number of interactions that we found. We only elaborate on a small selection here.

[5] Traces, failures and failures-divergence model.

[6] ... at least with the computing resources available to us.

	CFB	CNDB	CW	RBWF	RC	SB	TCS	TL	TWC	VM	CT	GR
CFB	××	–	–	–	–	–	–	–	–	–	–	–
CNDB	××		–	–	–	–	–	–	–	–	–	–
CW	××	×	×	–	–	–	–	–	–	–	–	–
RBWF	×××××	××	×××	×	–	–	–	–	–	–	–	–
RC	××			××		–	–	–	–	–	–	–
SB	××			××	×		–	–	–	–	–	–
TCS	×	×	×	×××				–	–	–	–	–
TL	×			××				×	–	–	–	–
TWC	×××	×	×××	××××	×	×	×	×		–	–	–
VM	×		××	××	×	×	×		××	×	–	–
CT	××××	×	×××	××××	×	×	××	××	×××	××	××	–
GR	××	×	×××	×××	×	×	××		××	××	××	×

Each × in the table stands for a distinct interaction.
We did not distinguish feature combinations by the order of feature integration, hence the top right half of the table is not used.

Table 1. Feature Interactions Phase 1

- **TCS & GR:** Both features are triggered by an incoming alert message in the idle state (type 1 interaction). Therefore they cannot both be active on the same line.
- **TCS & CW:** When the a subscriber of Call Waiting is in a call, further incoming calls (i_alert message) are not screened, since TCS is triggered only in the idle state; this is a type 2 interaction.
- **TCS & TWC:** A typical type 3 interaction occurs because Terminating Call Screening introduces a new message (i_inform). All lines in the network are upgraded so that they can deal with this new message, but only in BC states. So if someone uses Three Way Calling to call a Call Screening subscriber, an 'i_inform' message from that line to the (TWC) caller cannot be processed by the TWC feature.
- **RBWF & RC:** Ring Back When Free initiates the ring-back without a dial message, therefore Reverse Charging will not be triggered. This is a type 4 interaction, however, if the Ring Back feature were redefined to use the dial message, we would still get an interaction, namely of type 2.

7 Conclusions

Taking part in the contest was fun, but also quite exasperating at times, due to ambiguities in the contest instructions and bugs in the specifications of POTS and the features. We spent a lot of time on getting our model to work at all – many problems were synchronisation problems and mismatches of sending and receiving, i.e. bugs in the protocol.

The verification task was made difficult by the lack of a clear description of what was considered incorrect or undesirable behaviour on the one hand, by certain shortcomings of the model checker FDR2 on the other. The former is probably quite realistic, because at specification time, the requirements are often not fixed, and only in the process of "playing with the system" does one discover contradictory requirements, or additional assumptions that need to hold.

We faced the usual problems of underspecification and ambiguity when dealing with a natural language description. For example, what constituted a correct billing record was not stated. The CNDB feature relied on some internal bookkeeping in the switch, which was never made explicit. However, to assess the effects of 'anonymous' messages, one needs to make assumptions about the capabilities of the switch in this respect. It is unlikely that all contestants will make the same assumptions, hence it becomes almost impossible to compare their results about this feature.

While the success of the syntactic method described in section 4 seems to cast doubt on the value of model checking for the detection of feature interactions, we would not have reached an 'implementation' of POTS and the features without the aid of model checking. If ProBE and FDR2 had been nothing more than debugging aids in the development of our system,

they would have been invaluable in getting the system right, and moreover gaining a good understanding of it.

Furthermore, we believe that the syntactic method captures mainly the 'obvious' interactions. Once these are out of the way, one needs a way to find those interactions that can result, e.g. from different interpretations of the same data in different features. These interactions may result in strange behaviour or simply in the violation of invariants, but this will only become visible in runs of the system (model).

Acknowledgments. Thanks to anonymous referees for helping us to improve the paper. Financial support from the EU through Esprit working groups FIREworks (23531), and from British Telecom is gratefully acknowledged.

References

1. M. Calder and E. Magill, editors. *Feature Interactions in Telecommunications and Software Systems VI.* IOS Press, 2000.
2. M. Kolberg, E. Magill, D. Marples, and S. Reiff. Results of the second feature interaction contest. In Calder and Magill [1], pages 311–325.
3. M. Kolberg, E. Magill, D. Marples, and S. Reiff. Second feature interaction contest. In Calder and Magill [1], pages 293–310.
4. M. C. Plath and M. D. Ryan. Entry for FIW'00 Feature Interaction Contest. Technical report, School of Computer Science, University of Birmingham, February 2000. Available from `ftp://ftp.cs.bham.ac.uk/pub/authors/M.D.Ryan/00-fiw-contest.ps.gz`.
5. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1999.