Results of the Second Feature Interaction Contest

Contest Committee
Mario Kolberg(1), Evan H. Magill(1) (chair), Dave Marples(2) and Stephan Reiff(3)

(1)University of Strathclyde,
Department of Electronic and Electrical Engineering,
204 George Street, Glasgow G1 1XW, Scotland, UK
{m.kolberg, e.magill}@eee.strath.ac.uk

(2)Telcordia Technologies
445 South Street 1J226B, Morristown, New Jersey 07960
dmarples@research.telcordia.com

(3)University of Glasgow,
Department of Computing Science,
8-17 Lilybank Gardens, Glasgow G12 8RZ, Scotland, UK
sreiff@dcs.gla.ac.uk

Abstract. The Second Feature Interaction Contest, held in conjunction with the Sixth International Workshop on Feature Interactions in Telecommunications and Software Systems, attracted several submissions. This document summarises the submitted results and draws some conclusions.

1 Introduction

Five teams participated and submitted to the Second Feature Interaction Contest. However, only four qualify for full evaluation. The submission by the team of the University of Uppsala (Naeser et. al.) only submitted to the second phase and hence could not formally take part in the contest. However, as their submission contains interesting aspects we include their results in this document. Table 1 lists the participating teams.

Entry	Contestants	Organisation	Tool
H	Robert J. Hall	AT&T Labs Research (USA)	ISAT
N	Masahide Nakamura Tianbao Ding Jacques Sincennes Xiaotong Liu Luigi Logrippo	University of Ottawa (Canada)	LOTOS
P	Malte Plath Mark Ryan	University of Birmingham (UK)	FDR and ProBE
S	Dimitri Samborski	LORIA (France)	(no tool name given)
X	Gustaf Naeser	University of Uppsala (Sweden)	

Table 1: List of Contestants

The remains of this document mainly consists of excerpts from the five submissions explaining the applied method. Those sections are largely unedited and hence all credit goes to the contest participants. The full submissions from each participant can be found online. The corresponding URLs are noted in the relevant sections.

Specifying the features and the overall telephone system was not easy and a number of changes to the specifications were necessary. As a first result, all applied tools and methods could obviously cope with the changes to the specifications. We would like to thank the contestants for their effort in identifying and pointing out problems with the initial service specifications and also for taking part in this contest – thus bringing the area of feature interaction research closer to the required benchmark and test suite for tools.

2 Entry H: Robert J. Hall[1]

The full results and a complete list of all interactions as well as illustrative scenarios for this entry are available at [1].

This study was performed using the *ISAT* (Interactive Specification Acquisition Tools) prototype reactive system design environment. The use of ISAT in detecting feature interactions is described in [2]:

The behaviour of a feature (in the contest description) is to be understood as the behaviour of the BCM but with explicit overriding of state transitions and outputs by behaviour of the feature. This lends itself naturally to the foreground/background model approach of [2]: The BCM is the background model and each feature is a separate foreground model. Feature combination is done via the "Foreground merge" described in [2]: the two (or more) foreground feature models are merged textually, producing a new foreground model whose effects (state transitions and outputs) are the union of those of the constituent models. This new model then becomes foreground and the BCM remains the background.

Naturally, merging two models may possibly create inconsistencies, for example, when one model dictates a change to one state in a given situation while the other dictates a change to a different state in the same situation. Besides inconsistent next states, the two models can also dictate inconsistent output events, such as sending both ring-tone and busy-tone at the same time to the same phone. These types of inconsistencies are termed TYPE I interactions. ISAT detects these with the tool CHECK-FOR-BINARY-CONFLICTS. Among the twelve features, there were ten such interactions detected.

[2] discusses as well TYPE II interactions, which occur when a state invariant (or other correctness property) that is proven true of one of the features can no longer be proven true of the merged model. For this study, I chose not to use this type of detection, because proving the invariants is too time consuming for the amount of time I had to do this study. The potential return on such activity is not high enough, in my judgement, to warrant the effort. Given a more realistic scenario, where one is actually investing the time and effort to thoroughly validate each feature to store in a reuse library, this technique may once again be useful.

Finally, TYPE III interactions are defined as other interactions not detectable as TYPE I or II. These range from low-level problems such as undesired infinite wait states

[1] Robert J. Hall; hall@research.att.com; AT&T Labs Research; 180 Park Ave, Bldg 103; Florham Park, NJ 07932 USA

to higher level variations from desired behaviour. All interactions found other than the ten TYPE I interactions are TYPE III.

A shortcoming of my contest entry for the 1998 contest was the opportunistic way in which I detected TYPE III interactions. Essentially, I detected one only if an illustrative scenario for it was suggested by the use of some other tool or technique. This year, I have tried to systematise the search for TYPE III interactions by the following method. For each feature pair, $<F1, F2>$ I tried to design a set of test scenarios that each used at least one instance of F1 while the set together 'covered' all behaviours of F2. I then considered the pair $<F2, F1>$ in this way as well. [3] describes this method in more detail. Any undesirable behaviours discovered in this way were deemed a feature interaction and reported. All interactions for $<F1, F2>$ and $<F2, F1>$ were reported together below. ISAT includes tools for measuring statement and branch coverage of the models.

For each interaction discovered, I have given an illustrative scenario that gives both the sequence of input (trigger) events as well as the resulting state changes. Using the foreground/background models, ISAT simulates the action of the model on the scenario, thus providing proof that the interaction exists. For each interaction, the result table gives an identifier for the interaction (such as TCS/CNDB(1)) and a brief textual description of the interaction. The tables can be found in [1]. The detected interactions are marked in Table 3.

To help generating scenarios, I have implemented some high-level Lisp macros that expand into the desired sequence of trigger events. For example, (CALL "A" "B") expands into the usual sequence of seven trigger events from A going *offhook* through the *i_free* sent by B to A. This should end with A getting *ring-tone* and B *ringing*. The (ANSWER "A" "B") macro then expands into a normal answer sequence, namely *OFFHOOK* and *I_CONNECT*. I have used these shorthand notations to write down the scenarios and then had ISAT automatically expand them, simulate them, and print out the actual low-level sequence of trigger events interleaved with observations of the state variable changes resulting from each. That way the detailed scenarios given in [1] were produced.

2.1 Modelling Notes Important for This Summary

1. In the scenarios, the *PROGRESS* event is a dummy event that causes untriggered transitions to occur (e.g. BC2 to BC3). I will term states having untriggered transitions "ephemeral states". The contest description has dictated that the *PROGRESS* event triggering the transition out of an ephemeral state happens immediately (prior to all true trigger events anywhere in the system). Thus, these states are not really observable. All other states I term "persistent states" in that they last until the next true trigger event.

2. I have not reported any "interactions" that occur because some true trigger event occurs while a user is in an ephemeral state, since the contest rules define this as impossible. Note that this makes some of the "BUSY" arcs on ephemeral states impossible to take. It also rules out interactions involving nonfunctioning busy treatments during those states.

2.2 Effort Estimate

These are rough estimates of the numbers of hours for various tasks. For this summary, I have added the times for the two phases.

Modelling:	12h (initial modelling), 6h (debugging, fixing)
Binary Interactions: Type I: Type III:	 1h (finding), 1.5h (checking (incl scenario generation)) 12h (finding), 22.5h (checking (incl scenario generation))
Trinary Interactions: Type III:	 3h (finding), 3h (checking)
Total (Both Phases):	61 hours
Writeup:	6 hours (includes documentation types which probably would not normally be part of the natural software process in dealing with interactions)

Table 2: Hall's Time Estimate.

3 Entry N: Masahide Nakamura et. al.[2]

The full report for this entry are available at [4].

3.1 Overview

The University of Ottawa Team adopted a Lotos-based method for feature interaction detection. Use Case Maps were also used in a preliminary filtering phase. The tools used for the project were: Cadp Toolkit [5] , LOtos LAboratory (Lola) [6], Xeludo [7] and Use Case Map Navigator [8]. Figure 1 shows an overview of our method. In the figure, rectangles denote inputs and ouputs, ovals represent processes and arrows represents workflow. Using Figure 1, we outline our project.

(a) First, we specified the features in the Lotos language following the contest instructions. The feature definitions were given by Communicating Finite State Machine (CFSM), and we used the Lotos state-oriented style. Therefore, the mapping to our Lotos specification is faithful to the feature definitions.

(b) A validation process was performed, using Xeludo and Lola's step-by-step execution, to check if the Basic Call model and individual features were correctly specified.

(c) According to the contest instructions, we combined multiple features in a Lotos specification.

(d) In parallel with the Lotos specification, we derived the abstract behaviour of the features, by a kind of reverse engineering process. The feature behaviours were captured in terms of Use Case Maps (UCMs) representation. The tool Use Case Map Navigator helped in this process.

[2] Masahide Nakamura, Tianbao Ding, Jacques Sincennes, Xiaotong Lu and Luigi Logrippo; School of Information Technology and Engineering; University of Ottawa; 150 Louis Pasteur; Pav. MacDonald Hall; PO Box 450, Stn. A, Ottawa, Ontario, Canada; {nakamura, tding, jack, xlu, luigil}@site.uottawa.ca

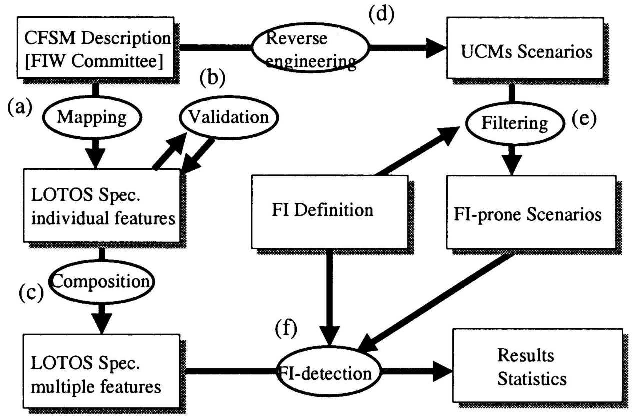

Figure 1: Outline of the Approach used.

(e) Applying a feature interaction filtering method to the UCMs, we derived FI-prone combinations and scenarios which are likely to lead to interactions.

(f) Feature Interactions were detected according to a definition based on LOTOS test equivalence. Intuitively, all possible behaviours (traces) of single features must still be valid after feature composition. Otherwise, FI occurs. Thus, we used the traces from single features as test sequences for compound features. The CADP toolkit provided automatic trace derivation, whereas LOLA was used to perform testing. For difficult cases, step-by-step simulation of scenarios found by using the filtering method was applied. In addition to that, semantic analysis for the billing database was conducted.

This work led to the discovery of 71 interactions among all possible pairs of given features. No interactions involving three features were found. Some feature combinations were analysed automatically, while others were analysed by step-by-step execution based on scenarios discovered during the filtering process.

3.2 Our Definition of Feature Interaction

Our definition of Feature Interaction is based on LOTOS testing equivalence [9], and is also similar to the definition of [10]. Intuitively speaking, for two given features F1 and F2, if the behaviour of F1 (or F2) is not accepted by the compound feature F1+F2, it can be said that F1 interacts with F2. Specifically, we use a set TF1 (and TF2) of traces derived from a single feature F1 (and F2, respectively) as testers against F1+F2.

In process algebra [11], a pair of processes P and T, where T is taken to be the tester of P, forms a system P$\|$T, where $\|$ is the parallel composition operator by which all common events in P and T are synchronised. The tester T contains a special event, success, indicating a successful termination. During the execution of P$\|$T, if T reaches success, then we say that P **passes** T. If P passes T for all possible execution paths in P, we say that P **must** T. So, the detection process would be to check a condition [F1+F2 must TF1] and [F1+F2 must TF2]. The tool LOLA supports these definitions.

However, F1+F2 will contain some events that are uniquely contained in F2 but not in F1. Hence, the execution of TF1 might be blocked because of such unique events, even though TF1 is successfully terminated after those events. Therefore, we define a process UF2, which recursively performs events unique to F2, and let UF2 skip the unique events that might block TF1[3]. Our definition of feature interaction then becomes:

> F1 interacts with F2 $\Leftrightarrow$ For systems (F1+F2) $\|$ (TF1$|||$UF2) and (F1+F2) $\|$ (TF2$|||$UF1), $\neg$ [F1+F2 must (TF1$|||$UF2)] or $\neg$ [FI+F2 must (TF2$|||$UF1)] where $|||$ is the interleaving operator of two processes.

3.3 Method

3.3.1 Feature Interaction Detection

In order to detect FIs between FI and F2, we need the specification of the combined features F1+F2, the sets TF1 and TF2 of traces derived from the single features F1 and F2, and processes UF1 and UF2 containing unique events of F1 and F2, respectively. The traces are derived by the tool, CADP. This tool expands a global labelled transition system and derives traces from a given LOTOS specification. Then, by comparing events in TF1 and TF2, we obtain UF1 and UF2. Finally, the tool LOLA performs the testing.

An example will be useful to illustrate the method. In the following, [] and ; respectively denote choice and concatenation operators. Consider interaction detection between Reverse Charge (RC) and Teenline(TL), where user 2 has RC and user 1 has TL. Then, the combined specification RC2+TL1 might be "RC2+TL1=...[] offhook1; enter TLPIN1; dialtone1; dial12; billing_reverse12; alert12...", and the trace from RC2 could be "TRC2=offhook1; dialtone1; dial12; billing_reverse12; alert12; ...". When applying TRC2 to RC2+TL1, the testing is blocked by enter_TLPIN1. However, enter_TLPIN1 is a unique event in TL1, it is skipped by the process UTL1 and the testing continues. Eventually, TRC2 is accepted, and there is no interaction.

Next, consider the interaction between RC2 and SB2 (Split Billing). Then "RC2+SB2 = offhook1; dialtone1; (dial12; billing_reverse12; alert12; ... ; [] dial12; billing_split12; alert12; ...) ..." and "TRC2 = offhookl;dialtonel ;diall2;billing_reversel2; alertl2;...". In the testing, if RC2+SB2 proceeds in the second choice, then the testing fails. Even if billing_split12 is skipped by USB2, TRC2 still has to consume billing_reverse12. This is an interaction.

Although our method requires a global model to derive the traces, we only need it for single features. Note that most state-space based methods require the global model for the compound features.

3.3.2 Feature Interaction Filtering

The detection method we have described can be completely automated, however in practice it meets a scalability problem. For complex cases, even trace derivation takes a huge state space. Also, it is very difficult to interpret which traces are essential to reveal FI. To cope with this problem, we have used FI filtering. FI filtering is a rough pre-evaluation to estimate which feature combinations have a possibility of FI. Based on the feature descriptions and FSM specifications, we abstracted the features in terms

[3] The UF2 plays a similar role to the projection in Nancy Griffeth's definition [10].

of a high-level representation, called Use Case Maps (UCMs). This reverse-engineering process was done manually by focusing on the functionality of the features and eliminating message exchanges based on causal relationships between events. We applied the filtering method proposed in [12] and derived FI-prone scenarios. For example, an interaction between CFBL (Call Forwarding Busy Line) and RBWF (Ring Back When Free) was identified by the filtering method, since both are features for handling busy conditions. Therefore, if a user X has both features and a user Y calls X, then interaction might occur. These FI-prone scenarios were used in order to find interaction scenarios for guided simulation, as well as to detect interactions in complex cases.

3.3.3 Analysis of the Billing System

The contest committee imposed special requirements on the billing system, which cannot be expressed with CFSM specifications only. Hence, a DB process was composed in parallel with the CFSM system in order to implement the billing operations issued by features (see 3.3 in [13]). A similar idea was described in [14].

Two databases are deployed: the User Billing Record (UBR) database and the User Billing History (UBH) database. The UBR database is used to record payer's info. Each user has one record (as payer) in the UBR database, and this record can have multiple billing items (which consist of caller, called, beginTime, endTime). The UBH database is used to track billing operation histories. Each user has one record in the UBH database, which consists of split-billing, reverse-charging, call-forwarding, billing-start, billing-stop, called (for storing the forwarded/bypassed number). By introducing the UBH database, we are able to track billing-forward/billing-reverse/billing-operations that occurred prior to billing-start/billing-stop operation, so that the billing machine can correctly charge the relevant users. This also makes it possible to detect any missing billing-start/billing-stop pairs of operations, so that we can remove the associated billing records from the UBR database upon receiving billing-onhook signal.

An End process is supplemented to monitor the end of a call and output the UBR and UBH database results. Provided that the end of a call can be reached, this billing analysis method enables us to examine the databases and detect any billing violations that indicate FI.

4 Entry P: Malte Plath and Mark Ryan[4]

The full report for this entry is available at [15].

The tools we used are FDR[16] and ProBe[17], developed by Formal Systems Ltd. We also used the Python programming language. FDR and ProBe are tools for model-checking and animating specifications written in CSPM, a machine-readable version of Hoare's CSP.

We described the Basic Call Model (as given in the Contest Instructions) in CSP_M. Our call model is parameterised by a finite set of subscribers. We found we could verify deadlock freedom of the basic call model with four subscribers comfortably within minutes. The time memory required to check the model grows rapidly with the number of users. However, we believe that most of the interactions can be witnessed with three or four subscribers. The effect of adding features, on the other hand, proved rather moderate.

[4]Malte Plath and Mark Ryan; School of Computer Science; University of Birmingham; Birmingham B15 2TT; UK; http://wwv.cs.bham.ac.uk/(mcp, mdr)

We developed an extension of CSP_M for describing features, as additions or replacements of transitions in the Basic Call Model, and wrote a script in Python to apply a feature to the Basic Call, resulting in a new version of the Basic Call which incorporates the feature. The script also takes arguments giving the name of the subscriber, and other parameters (such as the forwarding target for CFB, etc). The script can be used to apply a feature to an already-featured system, giving systems with several features. In general, the order of integrating features is significant.

Some effort was spent debugging the basic model, and ensuring that each feature could be integrated individually without producing deadlocks.

To find feature interactions among pairs of features, we used the script to obtain the CSP_M model for the basic call model with both features integrated. We then used FDR to detect deadlocks. A deadlock represents a potential interaction, which has to be investigated. We could also use ProBe to simulate scenarios, and use FDR and ProBe in conjunction with observer processes to restrict attention to certain types of paths. Our list of pairwise interactions is presented in Table 3 and also in̈citeplath-contest.

We also attempted to find feature interactions among triples of features, which were not interactions between pairs. We set about this as follows: we looked for triples of features which were pairwise interaction free (taking into account constraints on subscriber identity). This resulted in approximately 10 triples. We eliminated those which contained features which our experience of the pairwise case had shown to be relatively benign. We ended up with just one triple of features: CW/TCS/VM (each with a different subscriber). We were not able to find any interaction (given that constraint on the subscribers). Our conclusion is that the given set of features exhibits so many pairwise interaction cases that there are nearly none to look for in the triple case.

5 Entry S: Dimitri Samborski[5]

The full report for this entry is available at[18].

5.1 Conditions

Ten telephone services were tested. These services are: CFB (Call Forwarding on Busy line), CW (Call Waiting), TWC (Three Way Call), TL (Teen Lirne), TCS (Terminatimg Call Screening), RC (Reverse Charging), CNDB (Caller's Number Delivery Blocking), RBF (Recall When Free), VM (Voice Mail) and SE (Split Billing).

We test how different services work together depending on how many instances of each of them are in the system. We suppose the system is symmetric, that is, if we have already tested the combination "the phone A has the service S_1 and the phone B has the service S_2", then we will not test the combination "the phone A has the service S_2 and the phone B has the service S_1". All the phones are supposed to have the BasicCallModel. Thus, there are 55 combinations to test for two way interactions and 220 for three way interactions, but any combination can present several cases depending on service parameters or service disposition. We test combinations by series, putting together a selected service with all the other services one by one. We note $S_1 \oplus S_2$ the composition of services S_1 and S_2. This includes all possible combinations of S_1 and S_2, so the composition operator $\oplus$ is commutative and associative.

[5]Dimitri Samborski, B.P.239 Loria, 54506 Vandoeuvre-les-Nancy, FRANCE, samborsk@loria.fr

Testing services consists of specifying their invariants, choosing a configuration and running the model-checker on it. The model-checker verifies that all invariants are preserved. Sometimes (especially in cases of paying services) we verify properties by analysing all visited states of the system (the properties like "is there a state where A pays for a call to B while B is supposed to pay the incoming calls since he is subscriber of RC?").

5.2 Non interacting cases

Before starting to test combinations of services, let us see first whether we can eliminate some cases because the corresponding services would never create any interaction.

Any service has three functionalities: it receives tokens, it sends tokens and it sets some variables. Let $T_{in}(S)$ be the set of tokens the service receives and processes and $T_{out}(S)$ – the set of tokens resulting from the processing of $T_{in}(S)$. Consider the following system of conditions:

$$\left\{ \begin{array}{lcll} T_{in}(S_1) & \cap & T_{in}(S_2) & = \emptyset \\ T_{in}(S_1) & \cap & T_{out}(S_2) & = \emptyset \\ T_{in}(S_2) & \cap & T_{out}(S_1) & = \emptyset \end{array} \right. \tag{1}$$

The first condition means that the services don't compete for the same tokens and the last two conditions mean that any service doesn't process tokens issued from the other service.

We cannot affirm yet that these conditions are sufficient to guarantee the absence of interactions between S_1 and S_2. In fact, S_2 can set a variable whose value violates the invariant of S_1. It can also send a token that passing through S_1 will make BCM modify a variable and thus violate the invariant I_{S_1}, of S_1. We require the following conjunction be true:

$$\begin{array}{ll} \wedge & I_{S_1}(T_{out}(S_2)) \\ \wedge & I_{S_1}(T_{out}(S_1)) \\ \wedge & I_{S_1}(BCS(T_{out}(S_2))) \\ \wedge & I_{S_1}(BCS(T_{out}(S_1))) \end{array} \tag{2}$$

where $I_S(M)$ is true if the service's S invariant is preserved when any token from the set M arrives to the service. Note that $I_{S_i}(BCS(T_{out}(S_i)))$ is true, because this is the condition of service S_i validation (it doesn't violate its own invariant).

We can now state that if the conditions (1) and the equation (2) are verified, then S_1 and S_2 don't interact. We call such services **independent**.

After having analysed the specifications of the given services, we conclude that CNDB is independent from all other services and that there are thirteen other pairs of independent services: TCS and RC, TCS and SE, TCS and TL, TL and RC, TL and SE, TL and VM, TL and RBF, TL and CFB, CFB and VM, VM and RC, VM and SE, RBF and RC, RBF and SE. As for the other cases, we tested them with our model-checker.

6 Entry X: Gustaf Naeser[6]

6.1 Service Description

Services of the contest are described as extensions to a basic call. This is done by replacing or adding transitions in the Basic Call model (BC). This has the drawback that the activation of services is limited to where they are defined to start in BC. For example, the Reverse Charging service (RC) should, following it's definition, only be started when a dial is made in the node BC3. However, there are other services that allow dials from users, e.g. Three Way Calling (TWC), and having the services as extensions to the BC will not activate RC on dials made in TWC. Also, the RC service will not work with forwarded calls since there dial does not directly concern the phone subscribing to the RC service. There is no notion of service composition (apart from one service and BC) in the contest description.

To cope with service activation and service composition, we describe services as stand alone features (processes). Features have an activation condition and at least one transition. The activation condition describes the situations when the feature can be activated.

In the case of RC, the activation condition is that an *x_alert* (for an explanation of the *x_alert* notion see section 6.2) is issued in which the subscriber of RC is the terminating party. The *x_alert*, not the *dial*, is used in order to handle all calls not initiated by a *dial*, e.g. forwarded calls. The usage of *x_alert* for activating the feature also has one big advantage, it can place the service at the terminating side of the call, not on the originating side as in the contest description[13].

Another obvious condition for activation is that the service is subscribed and that any other system conditions the feature places are met, but there can be others as well, like that the service is not already activated (we only want at most one active instance of TWC for a user).

Features are run in the system as separate processes. They are created when their activation conditions are met, coexist with the rest of the system during their life and they terminate when they are done.

The coexistence part introduces problems. Our model does not allow features to directly modify other features, i.e. it is not possible for a service to destroy another feature or make it jump from one state to another. However, services may block other services. Blocking can be made on either the transition level (preventing a process to react to an incoming event) or on the event level (screening one responding event of a process but letting the other ones through). With the blocking ability it is then left to features to guide other features behaviour, e.g. if BC is in BC5 but no normal call is set up the service responsible for this new behaviour will also be responsible for guiding the "stuck" BC to it's termination.

6.2 The Base System

We have introduced voice into the system as we regard voice as one main property of telephony systems. We also use it to find interactions. All tones, bells and voice are explicitly started and explicitly ended.

[6] Gustaf Naeser, Jan Nyström, Bengt Jonsson, Tiziana Margaria, Dept. of Computer Systems, Uppsala University, Box 325, S-751 05 Uppsala {bengt,gaffe,jann,tiziana}@docs.uu.se and Bernhard Steffen 3 LS V, Universität Dortmund, Baroper Str.301, D-44221 Dortmund, steffen@cs.uni-dortmund.de

Calls in our model consist of one or more connection legs. Connections are given a unique connection number to allow multiple connection legs between a pair of users. Features are aware of connection legs, e.g. BC does not demand that the party accepting the call is the same as the one the call was placed to.

The usage of connection legs (and leg awareness) removes spurious startup interactions by separating the activation of the terminating and originating side features.

We do not have a switching system since its only functionality is to convert *o_events* into *i_events*. The only real need of the switching system seems to be for handling anonymous calls, but this is strange since the real identity of a caller is $number(X)$ and not X. (The switch is supposed to hold the association between anonymous callers and the return address for message exchange, cf. [13] section 3.2. This is done for all calls and hence it can use something else instead of X's identity in switching messages to the other side. This introduces the need to lookup the identity of X when desired, which seems like a normal way of handling the retrieval of a callers identity.)

6.3 Definition of Interactions

Interactions are identifies by:

1. (Internal) Loops are e.g. CFBL loops. Loops are identified by recognising that the exact same responding events are generated twice without an external event. (External events are events from the users, e.g. hook events.)

2. Inapplicable Events are when a feature want an event to be applied to the system to generate the new state, but the event cannot be applied since the conditions for its application are not met. E.g., a tone cannot be stopped if it has not been started, billing cannot be ended if not started, a voice cannot be started is a tone is present and so on.

3. Inconsistent Events are events that cannot be applied at the same time. E.g., two different tones cannot be started on the same phone.

4. Priority Conflicts in Blocks occur when two features try to block each other.

6.4 Searching for Interactions

Interactions are searched for using state space exploration and the exploration is pruned when an interaction is detected. The searching is depth first so traces to the interactions are rather obscure and hence not presented.

7 Summary of Results of Two Way Interactions

All contestants have detected two way interactions and the compiled results can be found in Table 3. All contestants assumed symmetric systems, hence we only require one half of the table. The service abbreviations are self explanatory in the context of the contest description[13]. Each cell in the table contains a combinations of blanks and the four characters H, N, P and S. The characters represent the participant as indicated in the list of contestants (Table 1). An occurring character represents that the respective contestant has detected an interaction between the services, blanks mean that no interaction was detected.

2 way		Phase 1										Phase 2	
		CFB	CNDB	CW	RBWF	RC	SB	TCS	TL	TWC	VM	CT	GR
Phase 1	CFB	HNPS											
	CNDB	NP	N										
	CW	HNP	NP	HNPS									
	RBWF	HNPS	HNP	HNPS	HNPS								
	RC	HNP			HNP								
	SB	HNP			HNP	HNPS							
	TCS	HNP	HNP	HNP	HNPS								
	TL	HNP		H	HNP								
	TWC	HNP	HNP	HNPS	HNPS	HNP	HNP	HNPS	HNPS	S			
	VM	HNP		H P	HNPS	NP	NP	P	HN	HNP	P		
Phase 2	CT	HNPS	HNP	HNPS	HNP	HNP	HNP	HNPS	HNPS	HNPS	HNP	HNPS	
	GR	HNPS	NP	HNP	HNPS	HNP	HNP	HNP	N	HNP	HNP	HNP	HNPS

Table 3: Results of Two-Way Interactions

This table is rather simplified, as some contestants detect several interactions between two features, whereby we only show whether or not an interaction was detected. For detailed results we refer back to the original submissions as available on the contestants web pages.

8 Summary of Results of Three Way Interactions

All contestants attempted to detect three way interactions. Plath and Ryan as well as Nakamura et. al. did not detect any three way interactions, as they did not test feature triples where pairs already produced two way interactions. The results of Hall and Samborski can be found in Tables 4 – 9.

When evaluating the results to find a winning team, all fairness was attempted. Nevertheless, this is very complicated, due to the difference in the individual methods and notations.

TCS	3 way		Phase 1										Phase 2	
			CFB	CNDB	CW	RBWF	RC	SB	TCS	TL	TWC	VM	CT	GR
	Phase 1	CFB												
		CNDB												
		CW												
		RBWF		S										
		RC	H	H	H	S								
		SB	H	H	H	S	S							
		TCS												
		TL				S								
		TWC												
		VM												
	Phase 2	CT												
		GR	H			H	H	H			H		H	

Table 4: Results for Three-Way Interactions – Part 1

		3 way	Phase 1										Phase 2	
			CFB	CNDB	CW	RBWF	RC	SB	TCS	TL	TWC	VM	CT	GR
R B F	Phase 1	CFB												
		CNDB	s											
		CW		s										
		RBWF		s										
		RC				s								
		SB					s							
		TCS												
		TL	s		s	s								
		TWC		s										
		VM		s		s	s			s		s		
	Phase 2	CT												
		GR												

Table 5: Results for Three-Way Interactions – Part 2

		3 way	Phase 1										Phase 2	
			CFB	CNDB	CW	RBWF	RC	SB	TCS	TL	TWC	VM	CT	GR
C F B	Phase 1	CFB												
		CNDB	s											
		CW												
		RBWF												
		RC												
		SB												
		TCS												
		TL	s											
		TWC												
		VM	s											
	Phase 2	CT												
		GR												

Table 6: Results for Three-Way Interactions – Part 3

		3 way	Phase 1										Phase 2	
			CFB	CNDB	CW	RBWF	RC	SB	TCS	TL	TWC	VM	CT	GR
C W	Phase 1	CFB												
		CNDB												
		CW		s										
		RBWF												
		RC												
		SB												
		TCS												
		TL			s									
		TWC		s										
		VM			s									
	Phase 2	CT												
		GR												

Table 7: Results for Three-Way Interactions – Part 4

		3 way	Phase 1										Phase 2	
			CFB	CNDB	CW	RBWF	RC	SB	TCS	TL	TWC	VM	CT	GR
RC	Phase 1	CFB												
		CNDB												
		CW												
		RBWF												
		RC												
		SB		s										
		TCS												
		TL												
		TWC												
		VM						s		s				
	Phase 2	CT												
		GR												

Table 8: Results for Three-Way Interactions – Part 5

		3 way	Phase 1										Phase 2	
			CFB	CNDB	CW	RBWF	RC	SB	TCS	TL	TWC	VM	CT	GR
CNDB	Phase 1	CFB												
		CNDB												
		CW												
		RBWF												
		RC												
		SB												
		TCS												
		TL												
		TWC								s	s			
		VM												
	Phase 2	CT												
		GR												

Table 9: Results for Three-Way Interactions – Part 6

References

[1] R.J. Hall, *Submission to the Second Feature Interaction Contest*, ftp://ftp.research.att.com/dist/hall/papers/isat/fiw00-contest-data.txt

[2] R.J. Hall, *Feature Combination and Interaction Detection via Foreground/Background Models*, In V. K. Kimbler and L.G. Bouma, editors, Feature Interaction in Telecommunications and Software Systems V, pages 232 - 246, IOS Press, Lund, September 1998.

[3] R.J. Hall, *Feature Interactions in Electronic Mail*, In M. Calder and E.H. Magill, editors, Feature Interaction in Telecommunications and Software Systems VI, IOS Press, Glasgow, May 2000.

[4] M. Nakamura, T. Ding, J. Sincennes, X. Lu and L. Logrippo, *Submission to the Second Feature Interaction Contest*, http://lotos.csi.uottawa.ca/FI/FIW00/Results/

[5] CADP (Caeser/Aldebaran Development Package), http://www.inrialpes.fr/vasy/cadp.html

[6] LOLA (LOtos LAboratory), http://yeti.dit.upm.es/ lotos/tools/lola.html

[7] The XEludo Toolkit, http://lotos.csi.uottawa.ca/eludo/

[8] UCMNav (UCM Navigator), http://www.UseCaseMaps.org/tools/ucmnav/

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

[10] A. Aho, S. Gallangher, N. Griffeth, C. Schell and D. Swayne, *SCF3^{TM}Sculptor with Chisel: Requirements engineering for communications services*, In V. K. Kimbler and L.G. Bouma, editors, Feature Interaction in Telecommunications and Software Systems V, pages 45 - 63, IOS Press, Lund, September 1998.

[11] M. Hennessy, *Algebraic Theory of Processes*, MIT Press, 1988.

[12] M. Nakamura, T. Kikuno, J. Hassine and L. Lofgrippo, *Feature interaction filtering with Use Case Maps at requirements stage*, In M. Calder and E.H. Magill, editors, Feature Interaction in Telecommunications and Software Systems VI, IOS Press, Glasgow, May 2000.

[13] M. Kolberg, E.H. Magill, D. Marples and S. Reiff, *Second Feature Interaction Contest*, In M. Calder and E.H. Magill, editors, Feature Interaction in Telecommunications and Software Systems VI, IOS Press, Glasgow, May 2000.

[14] Q. Fu, P. Harnois, L. Logrippo and J. Sincennes, *Feature Interaction Detection: a LOTOS-based approach*, to appear in Computer Networks.

[15] M. Plath and M. Ryan, *Entry for FIW'00 Feature Interaction Contest*, Technical Report, available at: http://www.cs.bham.ac.uk/ mdr/papers.html

[16] Formal Systems Ltd, *FDR Tool*, for more information see http://www.formal.demon.co.uk/FDR2.html

[17] Formal Systems Ltd, *ProBe Tool*, for more information see http://www.formal.demon.co.uk/probe.html

[18] D. Samborski, *Submission to the Second Feature Interaction Contest*, http://www.loria.fr/ samborsk/Pub/SSM.html