FORMAL CRITERIA FOR FEATURE INTERACTIONS IN TELECOMMUNICATIONS SYSTEMS

Jan Bredereke

University of Kaiserslautern, Postfach 3049, D-67653 Kaiserslautern, Germany
E-mail: bredereke@informatik.uni-kl.de, Phone: +49 631 205-3287, Fax: -2640
URL: http://www.informatik.uni-kl.de/aggotz/bredereke

Abstract

The feature interaction problem in telecommunications systems increasingly obstructs the evolution of such systems. We develop formal detection criteria which render a necessary (but less than sufficient) condition for feature interactions. It can be checked mechanically and points out all potentially critical spots. These have to be analyzed manually. The resulting resolution decisions are incorporated formally. Some prototype tool support is already available. A prerequisite for formal criteria is a formal definition of the problem. Since the notions of feature and feature interaction are often used in a rather fuzzy way, we attempt a formal definition first and discuss which aspects can be included in a formalization (and therefore in a detection method). This paper describes on-going work.

1. INTRODUCTION

Telephone switching systems are a classical example for long-lived and perpetually evolving software in the telecommunications domain. The first software controlled switching exchanges essentially still provided the plain old telephone service (POTS). Step by step, new features have been added since then which were supposed to offer added value to the customer (e.g. by call forwarding) and/or to the service provider (e.g. by improved accounting).[1]

Up to now a large number of features has been developed (several hundred, [Bo$^+$89]). Therefore the probability is high that augmenting such a system by one more feature will influence another feature, especially in a negative way. This is called a feature interaction. Both the notion of feature and the notion of feature interaction are used quite fuzzy and informal most of the time. This does not make a solution of the problem easier. The formalization in this paper is intended as an attempt to improve this situation.

Since we do not know how to *avoid* feature interaction problems in telecommunications systems altogether, we need to *detect* and *resolve* them ([BoVe94]). Detection of feature interactions may be done in different ways, for example by simulation or by online behaviour analysis, but we will concentrate on an off-line *verification* approach here. Usually, a verification approach works like this (compare, e.g., the contributions in [BoVe94]): we write a high-level, property oriented description for each feature and for the basic system (e.g., in temporal logic), and we write a lower-level, constructive[2]

[1] Example: the features defined in the ITU-T recommendations for Intelligent Networks (IN) [ITU93b].
[2] "Constructive" means "executable" or that an implementation can be derived directly.

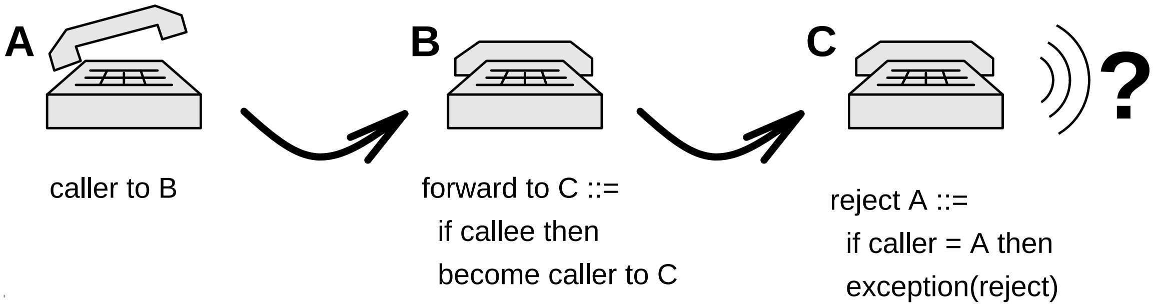

Figure 1. Feature interaction between call forwarding and terminating call screening.

description for them (e.g., in Estelle [ISO89] or SDL [CCI87]). Then we define an "implements" relation and prove mathematically that the combined system built from the constructive description of the basic system and the added features still implements all high-level properties.

This approach surely is promising, and it needs to be studied further. But it also has some limitations and prerequisites. If the proof is done by logical reasoning, the work is painstaking even with tool support. Furthermore, one can only verify those properties that are explicitly specified. And in general, it is practically impossible to formalize *all* expectations at the system which the service provider or the customer may have.

Even worse, existing property descriptions may become "incomplete" or "inaccurate" by a later addition of a new feature: Imagine that you have expressed all relevant properties concerning a "caller" in a telephone system. Then, add a new call forwarding feature to this system. When a caller A calls a callee B, then B may forward this call to a user C (compare Figure 1). This way B may become a caller, too, in a sense. Who is the caller for C? Therefore, the notion of "caller" has become a little muddy, and our carefully written formulae on properties of a caller may not denote the meaning anymore which we wanted them to have.

In this paper, we propose a different verification approach which may help finding feature interactions that could go unnoticed by other verification approaches. We have proposed formal definitions for the terms "feature" and "feature interaction" elsewhere [Bre95], which we sketch (restricted by space limitations) in Section 2. In Section 3, we describe detection criteria for feature interactions; in Section 4, we sketch a way to resolve the detected interactions; in Section 5 we discuss briefly how our generic formalism can be mapped onto real formal description techniques (FDTs); in Section 6 we present the tools developed up to now and outline the practical applications done so far; and the paper is concluded by a summary and an outlook.

2. DEFINITION OF "FEATURE INTERACTION"

2.1. What to Define and What to Leave Out

Before we tackle the feature interaction problem, we will give our definitions of the terms "feature" and "feature interaction" first. As has been pointed out by Cameron and Velthuijsen [CaVe93], it is crucial to define the problem first before trying to solve it.

Our formal definitions surely do not cover all the aspects which are commonly associated with the informal terms, as no formal definition probably ever will be able to. But they state more clearly what the achievements are and which limitations have to be taken into account while they are used.

Cameron and Velthuijsen [CaVe93] describe two views on the problem. In the business view, "a feature is a tariffable unit", and "a feature interaction occurs when the behaviour of one feature is altered by the use of another". "A second kind of interaction occurs when the use of one feature should alter the behaviour of another, but does not." In the implementor's view, "a feature is any increment of functionality added to an existing system". "Just as in the business view, a feature interaction occurs when one feature's behaviour is altered by the use of a second."

Even if there does not exists a clear (formal) concept of what a feature and a feature interaction are, the notion of "*behaviour* of a feature" is surely a central one. It is a necessary precondition for a feature interaction that this behaviour is changed. In this paper, we disregard the case where an interaction occurs because a change in behaviour does *not* take place when a change in the intensions requires it to do so, since the notion of intended behaviour is still a topic of research ([CaVe93]). Maybe a property oriented approach (compare Section 1) will be able to handle it. In the remainder of this section, we will develop a formal definition which is based on the central notion of the change in behaviour.

When we have to decide if the behaviour is still the same as before, there are different aspects of the behaviour. First, there are the functional aspects. This concerns the sequences of possible states/events. But the non-functional aspects can become important, too. This concerns any real time or performance aspects. Also, there can be important properties on knowledge, e.g. security properties. Unfortunately, the formal treatment of non-functional aspects is much harder than that of functional aspects, and many areas are still a topic for research [BCN95]. Therefore, we restrict our notion of feature interactions to functional aspects. This excludes, for example, the treatment of a case where one feature is impaired because another feature causes an overload on some shared resource.

We will not define what a service is. Commonly, it is some collection of features, but only if this collection is somehow self-contained. Since we will concentrate on functional aspects anyway, we have no need to take the pains with designing a formal definition of a service; we will stick to the task of defining a feature and its interactions.

For the formalization of the functional aspects of the behaviour, we advocate the use of automata [HoUl79]. The *computations* (execution sequences) of an automaton are a good and formal way to capture this kind of behaviour. An automaton is a finite description that specifies a behaviour. There are many variants on the exact definitions; we will present a suitable one in the following.

2.2. Suitable System Architectures

One may choose to use a collection of concurrent communicating automata, or a single global automaton (comprising different local automata which, again, have some form of concurrency and communication). If we take the first alternative, we still have to define formally what concurrency and communication is. Since a modelling of concurrency by interleaving is sufficient for our purposes, this question will inevitably lead to an

$$
\begin{aligned}
s^g \;=\; & ((\text{waitForAck}, 42, \texttt{'abcdef'}),\\
& (\text{connected}, 41, \texttt{'hijklm'}),\\
& (\text{idle}, 0))
\end{aligned}
$$

Figure 2. Example of a global state.

answer that equals the second alternative. Therefore, and since the semantics of the standardized formal description technique Estelle ([ISO89]) is defined in a very similar way, we take the second alternative. As soon as we want to apply our results to practical system specifications, we have to transfer the results to practical specification languages; a similarity in the underlying semantics will facilitate this.

The computations of an automaton can be formalized by a tree of reachable states or by a set of linear sequences of reachable states. Since we are not interested in terminating computations, we take the simpler second alternative.

In the effort of opening up existing telecommunications systems, abstract models have been developed, e.g. the intelligent network (IN) conceptual model [ITU92, DuVi92] by the ITU-T. The basic call state machine (BCSM) in the IN conceptual model (INCM) is designed in a way that the control flow remains as much as possible in the BCSM. Among others, this results from the communication overhead involved in passing away the control. On the other hand, this system design results in a rather complex semantics of the *detection points* (DPs) where control may be given away. It turned out that such a complicated architecture obstructs the formal analysis of the system. Therefore, we use a simplified IN architecture. We do not use explicit detection points, we just specify a homogenous automaton, and extensions result in the addition of more transitions and states of the same kind as the old ones. A closer adaption to the IN architecture (or *of* the IN architecture?) is a topic for future research.

2.3. Formalization by Automata

For space reasons, we cannot present our entire detailed formalization [Bre95]. Therefore, we will describe just the highlights and some important decisions which were taken during the formalization.

We start out by constructing a state space. It consists of a fixed set of system components which each have a fixed set of variables (Figure 2). Next, we define a simple global automaton, consisting of a state space, a set of "simple" state transitions and an initial state. Simple transitions have the disadvantage that their number can become infinite as soon as the domain of a variable is infinite, e.g., like for the natural numbers. Therefore, we allow to specify many simple transitions at the same time by a "transition". The number of these must be finite. The same concept to achieve a finite representation can be found in Estelle and SDL specifications, too. Next, we define a global automaton $A = (S^g, T, s_0)$ based on a simple global automaton, consisting of a state space S^g, a finite set of transitions T and an initial state s_0.

We define a computation of a global automaton to be a (possibly infinite) sequence of states which can be constructed using the set of simple transitions, starting from the initial state. Each global automaton defines a set of possible computations.

Now we come to the part of the formalism that will define what a feature is. For this, we describe how we add something to a global automaton. A prerequisite for the detection of interactions between features is to specify different features separately. Since we model the entire system by computations, it must be possible to associate each transition with a single feature[3]. For this, every transition must contain only functionality of a single feature. This implies that we have to use a *specific specification style* when we specify the system and its extensions. Two basic rules are that we *modify only on a coarse-grained level*, and that we *only add* to the specification. This means for the behaviour part that we never modify an existing transition to achieve the behaviour of a new feature, but we only add an entirely new transition. Also, we do not delete any obsolete transition (see Section 4 for how to get rid of it). The only way to extend a global automaton in our formalism is to add new possible computations.

We designed our formalism in such a way that it supports these stylistic rules for the state space part, too [Bre95]. One of the ideas is to use a state space extension function. It tells where new elements ("variables") are inserted into the system components, and where entire new system components ("modules"/"processes") are inserted into the global state space. Based on this, we define what an extension of a global automaton is [Bre95]. The extension may comprise both the state space and the behaviour. The extended global automaton must have all the computations which the non-extended automaton has, modulo a state space extension, and it may have new computations.

Finally, we define a feature $f = (\phi^g, T^e, d^0)$ for a global automaton by a global state extension function ϕ^g, a set of new transitions T^e and the initial state d^0 for the new state parts. Therefore, if we add this feature to a global automaton, the resulting automaton will be an extension of the original one in the above sense. Through this definition, a feature is an *increment* in the possible behaviour. This is a deliberate restriction. This way, we still have all the computations of the other features and can find deviations. (Compare the definition of a feature interaction below.) If the purpose of the new feature is to remove the other computation, then this is certainly an interaction, and it should be handled explicitely. In Section 4, we will discuss how this resolution may take place.

We require a global automaton to have at least one transition so that there is at least one feature name. By convention, the "basic system" of the global automaton which is always present carries the name B.

Now we consider different cases of feature interactions. If there is a computation c, whose steps out of simple transitions are partially associated with feature f, and partially associated with feature g, then this computation is only possible if both features are present in the global automaton. We cannot add both features independently to the automaton without influencing the behaviour of the other feature. Therefore, both features interact.

The condition that a computation is associated with two different features (not equal to the basic system B) implies a feature interaction, but this is not true vice versa. There may be two different computations belonging to two different features, but they have a common prefix belonging to the basic system where it is not decided which one of the two will be taken eventually. If there is such a non-deterministic choice for the

[3]For orthogonality, we consider the basic system as just another feature, too.

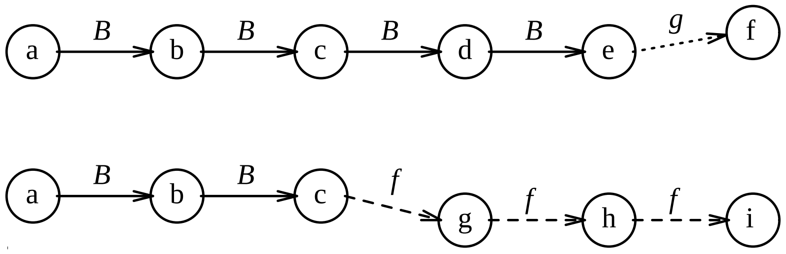

Figure 3. Feature f may divert the execution away from a computation of feature g.

automaton then it may be possible that the original behaviour of the first feature does not take place anymore. If this choice is only due to the addition of the second feature, this denotes a feature interaction (see Figure 3).

A feature with some terminating computations may lead to such a diversion state even without a non-deterministic choice. Nevertheless, all of these cases are handled by the following definition:

Definition 1 *A feature f in the global automaton A interacts with a feature g in A* iff $f \neq g$ and there exists a computation c^f that comprises at least one simple transition from f and a computation c^g that comprises at least one simple transition from g and an index $i \in \mathbb{N}$ such that the prefixes of c^f and c^g up to the i-th state are equal, i.e, $< c_0^f, \ldots, c_i^f > = < c_0^g, \ldots, c_i^g >$, and the simple transition (c_i^f, c_{i+1}^f) belongs to feature f.

Every feature which is added on top of the basic system will of course modify its behaviour. This is the intention behind adding the feature. Therefore, for practical purposes we do not count this feature interaction in the following definition.

Definition 2 *A feature interaction among the features of a global automaton A occurs* iff there exist two features f, g in A such that f interacts with g and where both are not the basic system B.

3. AUTOMATIC DETECTION OF POTENTIAL FEATURE INTERACTIONS

In Definition 2, we have defined what a feature interaction among the features of a global automaton is. Since computations are of infinite length in general, the criterion in Definition 1 cannot be used constructively (and efficiently). Therefore we have to find criteria which

1. can be computed efficiently,

2. are still necessarily fulfilled for feature interactions (wide enough), and

3. do include as few as possible potential interactions (specific as possible).

From Theorem 1 on (see below), we will present criteria which all fulfill the first two items and which become increasingly better at the third item.

3.1. Necessary Conditions for Feature Interactions

Under the assumption of infinite computations[4], there may be some feature interaction only if there is a reachable state s in which there is a non-deterministic choice between at least two transitions belonging to different features.

Proof: a feature interaction among the features of A implies that a feature f in A interacts with a feature g in A (Definition 2). So it remains for us to prove that this plus the premise that every computation is infinite implies the existence of the mentioned non-deterministic choice. Definition 1 provides us already with the state $s = c_i^f = c_i^g$ and the transition t^f. The only item still needed is a continuation transition t^g starting from s for the other computation. Since c^g cannot be finite, t^g must exist.

The condition of non-determinism is necessary for feature interactions, but (in general) less than sufficient because we did not count the interactions with the basic feature B. If a feature f interacts with B, then there may or may not be an interaction with another feature g, as shown in Figure 3. Therefore, if some feature f interacts with B, we are warned and have to check manually if there are transitions of g later in the affected computations. (We will give more specific criteria on this in the following subsections.)

Furthermore, the above criterion is still not efficiently applicable because of the condition that the state s must be reachable. This would imply a complete state space search. Therefore we weaken the condition further and drop that condition:

Theorem 1 Be A a global automaton such that its set of all possible computations contains only infinite computations. A feature interaction among the features of A may occur only if there exist a feature f in A, a transition t^f of f, a feature g in A with $g \neq f$, a transition t^g of g and a global state s such that both t^f and t^g are enabled (non-deterministically) in s.

Note that f or g may equal B in Theorem 1. Informally, Theorem 1 says that we can find any feature interaction if we compare all pairs of transitions from different features and check if their enabling sets overlap. If no such pair exists, then no feature interaction can occur. The proof is a straightforward extension of the above one.

3.2. Excluding Concurrent Transitions

The above criteria warn us of potential feature interactions. But even if the computations of a feature are affected by another feature, this may be harmless in some cases. E.g., think of two transitions belonging to different features which are enabled in the same states and where the resulting state when both have fired does not depend on the order of firing. (Compare Figure 4.)

Definition 3 *A potential feature interaction as determined by the criterion of Theorem 1 is harmless* iff the interacting transitions t_f and t_g are independent, i.e. iff firing one of them does not disable the other one and if both can fire then the firing order is irrelevant.

[4]The assumption that the basic system for itself is free of deadlocks can be justified by the applications in mind, which are reactive systems. A first feature interaction may invalidate the above assumption for a second, but we can always detect the first one and after a resolution also the second one.

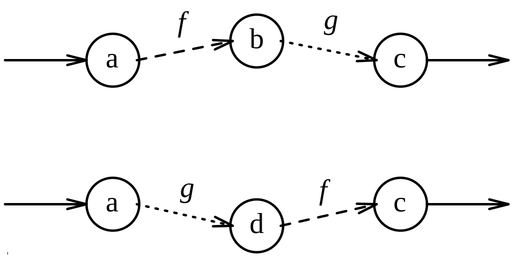

Figure 4. The firing order of these transitions is irrelevant.

This declares non-determinism as harmless if it, e.g., results from our modelling of concurrency of the local automata by interleaving. The condition of Definition 3 is usually hard to decide for two given transitions. Thus, we reformulated it based on the local state components modified by one transition and the local state components used by the other transition. As soon as we start to use a specific (constructive) formal description technique, these two notions are quite straightforwardly applied to specific language constructs. For example, an assignment to a local variable will lead to a modification of this local state component, and the use of the value of a local variable will be a use of this local state component.

The formal definition [Bre95] of the *output modification pattern* describes which local state components can possibly be modified by a transition. The *input pattern* describes which local state components are used at all by a transition. This comprises both the enabling conditions and the computation of the output. Output dependencies on the input in the fashion of the identity function are not counted. A transition t_2 has a *state space dependency* of a transition t_1 iff the intersection of the output modification pattern of t_1 and the input pattern of t_2 is not empty. Now we can give our reformulation of harmless potential feature interactions which should be easier to check for any specific formal description technique.

Theorem 2 A feature interaction as determined by the criterion of Theorem 1 is harmless if the interacting transitions t_f and t_g mutually have no state space dependency.

3.3. Excluding Interactions with the Basic System

Theorem 1 from Section 3.1 is a criterion that warns us of all possible feature interactions. As already mentioned there, it also warns us if a feature f interacts with the basic system B, because there may be situations like the one in Figure 3 where this influences another feature g indirectly. But since a feature which does not interact with the basic system at all is rather useless, we have to differentiate further cases when f interacts with B.

To this end, we use our knowledge of the typical structure of a telecommunications system (compare, e.g., [ITU93b, ITU93a]). Call processing is described there by one extended finite state automaton (EFSA) (if we take a global view) or by two EFSAs (in a distributed view). These typically have a *null state* where no call is established and which is reached again after the processing of one call has been completed. Usually, this is also the initial state. This allows us to reduce the analysis of the computations c^f and

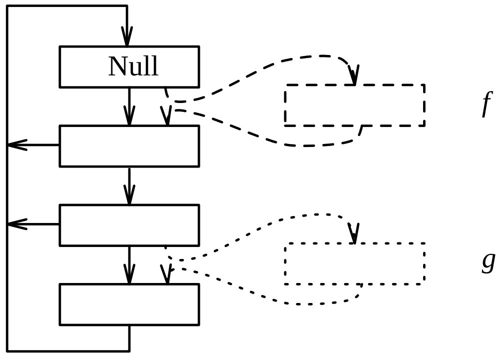

Figure 5. A typical situation with two new features.

c^g in Definition 1 to the end of the prefix where the null state has been reached the next time.

In order to reflect the use of EFSA in our formal modelling, we adopt the convention that the first component of the local state tuple describes the *major state*. Any sensible specification of a system will not only have a finite, but also a small domain of values for each major state. This allows us to do an abstraction step and reduce a global automaton to a simpler one with the same number of local components, but where these local components only have a major state and no extended state. This abstraction step to major states and the limitation of the analysis up to the next null state allows us to do an exhaustive reachability analysis without the tractability problems that a state space explosion imposes.

We start a reachability analysis at the state s where a transition of a feature f shows non-determinism with a transition of the basic system B (compare Theorem 1), and we follow all transitions from this state which belong to B. If we cannot find any transition not belonging to B or f before we reach the null state again, it is impossible that the detected non-determinism signifies a feature interaction among the features of the global automaton (Definition 2). The sketched search algorithm is worked out formally in [Bre95]. If we can find such a transition, there may be a harmful feature interaction and we have to analyse this situation manually. The manual analysis is supported with the information where the non-determinism showed up, which was the offending computation (prefix) and which were the possibly interacting transitions, these in turn provide information on the features which are concerned.

3.4. Excluding Independently Inserted Features

We may use another criterion that reduces the class of potential feature interactions further which have to be analysed manually. For this, we partially take into account the extended state space, too. A common situation in a local call processing FSA is shown in Figure 5. Two short automata extensions/features are inserted into the mostly sequential control flow of the basic local automaton. Examples for such a feature may be "abbreviated dialling" which translates a single digit into a full directory number, or "reverse charging" which changes some charging control variables. The first extension can never disable the second because it returns control back to nearly the same point where it took it away. This return situation may be detected easily during our exploration of the major state space, and be treated accordingly.

We call a transition t *major state preserving* iff t either does not modify the major states at all or there is some transition t_B of the basic system such that t_B modifies the major states in exactly the same way as t. Note that this definition does *not* comprise the case depicted in Figure 5. There, more than one transition is inserted before control is returned to the basic system. Such patterns are only sensibly defined for a local atomaton (see Section 3.5).

The only undesired thing that may happen now is that a first transition t_f modifies a part of the extended state space on which a second transition t_g depends. Here, we have to take into account indirect data dependencies, too. An example would be the case where the effects of a transition t_B associated with the basic system B depend on a state component which was modified by the first transition t_f, and where in turn a second transition t_g depends on these effects of t_B. To resolve this situation, we need to do some simple data flow analysis.

A sequence $< t_1, \ldots, t_n >$ of transitions contains an *indirect state space dependency of a transition* t_0 iff there exists a transition t_j, $1 \leq j \leq n$, which does neither belong to the same feature as t_0 nor to the basic feature B such that the intersection of the output modification pattern of t_0 (compare Section 3.2) and the input pattern of the concatenation of $t_1, \ldots, t_j$ is not empty. As mentioned before, such state space dependencies are quite easy to determine for the language constructs of any specific (constructive) formal description technique.

Again, we start a reachability analysis at the state s where a transition t_0 of a feature f shows non-determinism with a transition from the basic system B (compare Theorem 1), and we expore the major state space up to the null state. If t_0 is major state preserving, and if we do not find any sequence of transitions which contains a state space dependency of t_0, then it is impossible that the detected non-determinism signifies a feature interaction among the features of the global automaton. The sketched search algorithm is worked out formally in [Bre95]. If we can find such a transition, there may be a feature interaction and we have to analyse this situation manually, again. Note that we did not require the sequence of transitions to be executable by the global automaton. Only the major states had to "fit". This makes the criterion weaker than it could be, but it saves us from exploring the full state space.

3.5. Analysis on a Local Level

Since we describe distributed and concurrent systems, the specified systems consist of local components (modules/processes) which operate with a certain degree of independence. (Except for descriptions on the global functional plane.) Many transitions have effects only on one local component. This is supported by most of the specific formal description techniques. Their syntax of transitions allows only local effects except for some special communication commands. This can be exploited by performing individual checks for feature interactions on a local level first. Due to the more restricted state space and the smaller degree of concurrency, this allows a deeper and more thorough automatic analysis without a state space explosion.

All detection criteria from the preceding subsections may as well be applied only to one local component. This can already discover those feature interactions which happen only locally. For example, an interaction may be discovered (and resolved) which results from an unspecified order of originating call screening and abbreviated dialling, in an appropriate local setting. We can determine if a transition has only local effects by the output modification pattern introduced in Section 3.2. In Sections 3.3 and 3.4, we proposed some (major state) data flow analysis after the detection of non-determinism between two transitions of different features. This analysis may be restricted to a single local component as long as only transitions with local effects are involved. If we reach a transition with non-local effects we have three options: we can study the consequences manually; we can apply the following analysis globally, as described in the preceding subsections; or we can continue with another local analysis. These secondary level local analyses start with any transition that depends on the state component in question, as determined by the input pattern defined in Section 3.2.

A local analysis has a further advantage: in Section 3.4 we discussed major state preserving transitions. On a global level, we could include only those transitions into the definition which either didn't modify the major states at all or for which existed a "similar" transition of the basic system. This did not comprise the case depicted in Figure 5. On a local level, we can sensibly extend the notion of major state preserving to entire groups of transitions. We replace the single transition of above by a sub-automaton consisting of several transitions and states, and we only require that the entry and exit of this sub-automaton exposes the properties described above.[5]

4. RESOLUTION OF FEATURE INTERACTIONS

To resolve detected feature interactions, generally high-level design decisions on the policy of the service provider or on the needs of the users have to be made. And since the detected interactions are the typical cases nobody had thought of before, it is quite probable that a satisfactory answer cannot be deduced from any existing behavioural description (compare Section 1). Even if some solution can be deduced, we do not know if it also really satisfies the implicit and unexpressed requirements of the users and the provider.

But even if an automatic resolution is impossible in many cases, our approach to specify features and detect feature interactions allows for different methods to resolve the interactions. In the simple case, we have detected direct non-determinism between two transitions associated with different features. This is also the case when a new feature is "plugged" into the old system since, up to now, we do not have introduced a way to disable the old transition. We simply get this non-deterministic choice instead.

So, after the detection methods have been applied, and after we are sure that it is the right thing to override the old transition, we specify corresponding *precedences* between the transitions. Each transition of a feature is given the same precedence, therefore we only have to decide which feature takes precedence over which. This is done by setting up a precedence matrix for features. The matrix does not need to contain an entry for every pair of features, it is sufficient to enter the explicit design decisions we actually made.

Here is a sketch how the precedences are incorporated into our formalism. We define

[5] Some more details have to be considered, e.g., there must not be a deadlock, and the notions of input pattern and output modification pattern have to be extended suitably.

a global automaton with priorities $A^p = (S^g, T^p, s_0, p)$ for the hitherto existing global automaton $A = (S^g, T, s_0)$, where p is a partial order on the set of feature names. This also renders a partial order on the set of transitions T and it thereby allows us to construct the new global automaton A^p which is identical to A except that we remove any simple transition where all the corresponding transitions were overridden by other transitions with a higher priority.

Of course, there may be other cases where precedences are not sufficient. Then, we need to specify additional behaviour which resolves the conflict. We specify this behaviour formally like any other new feature, and we add it to the system iff both of the interacting features are added. This approach for the integration of the resolving behaviour allows us to investigate even interactions which may happen between this newly introduced behaviour and a third feature.

More details on the resolution of feature interactions may be found in our technical report [BrGo94a]. There, we apply our ideas to the formal description technique Estelle.

5. USING EXISTING FDTs

Up to now, we introduced a generic formalism for the specification of telecommunications systems, and we included only those language aspects which were necessary for our discussion. To make our ideas applicable, we need a full-fledged formal description technique which offers the appropriate expressive power that allows a system developer to specify real systems. Our ideas have to be adapted for such a FDT. But we designed our generic formalism in such a way that this adaption is supported well. E.g., the FDT Estelle is rather rich in language constructs, but its semantic model ([ISO89]) is very similar to our global automaton. Therefore it is possible to find syntactic criteria in Estelle for possible feature interactions that correspond to our definitions in the generic formalism. We have already investigated the notion of non-determinism between Estelle transitions in depth [The95]. The richness of Estelle provided a rather large number of special conditions to consider, but this only justified our decision to investigate appropriate feature interaction criteria in a simpler formalism first and to apply them to real FDTs later.

An application to the FDT SDL should pose no fundamental problems as well since the semantic models of SDL and Estelle are very similar. Concerning the FDT LOTOS, our simple global automaton is basically already a labelled transition system which is the semantic model of LOTOS.

6. TOOLS AND CASE STUDIES

Several tools for the FDT Estelle are in use now. E.g., there is a tool that computes Estelle priority values from a precedence matrix and generates the appropriate Estelle code. Our tool CONFINE [ThBr95] detects non-determinism between Estelle transitions of different features; and it detects and automatically eliminates Estelle transitions which have become non-executable because they are completely overlapped by new, higher-priorized Estelle transitions. The Pet/Dingo toolset [SiSt93] from NIST automatically generates executable code from an Estelle specification and animates the execution.

A first case study [BrGo94a][6] used a simple global service specification of a telephone switching system. Our detection tool CONFINE did not only find the already known feature interactions but also two interferences which escaped us while we specified this simple example. A manual analysis led to the conclusion that both cases are harmless if an implementation is sufficiently fast. Nevertheless, we achieved a deeper understanding of some underlying problems. Furthermore, CONFINE detected and removed automatically the inactive transitions which had become obsolete through the resolution procedure for the known feature interactions. Currently, we are working on a second case study which is based on a simplified version of the IN conceptual model [ITU92], specified in Estelle again, which takes into account the distribution aspect, too.

7. SUMMARY AND FUTURE WORK

The feature interaction problem in telecommunications systems currently obstructs the evolution of such systems more and more. We presented an approach for the offline detection of feature interactions which is different from other verification approaches. It does not rely on a high-level, property oriented description against which a lower-level, constructive description is checked. We observed that the behaviour of a feature is a central notion, and that feature interactions are indicated by a change in this behaviour. We formalized the notion of change in a definition of feature interactions. From this, we derived a necessary (but less than sufficient) condition for feature interactions. It can be checked mechanically and points out all potentially critical spots. These have to be analyzed manually. We proposed further criteria to exclude some cases where the interactions are harmless, and we made use of our knowledge of typical telecommunications systems to define criteria for more cases in which there cannot be any negative influence even if the basic criterion indicates a possible problem. This led to a restricted reachability analysis.

Since the notions of feature and feature interaction, as they are widely used, had been too fuzzy to have hope for a resolution of the problem, we had to start out with an attempt of a formal definition and discussed which aspects could be included in a formalization (and therefore in a detection method). E.g., we restricted ourselves to the functional aspects of a system. This allowed us to use the computations (possible execution sequences) of a global automaton to catch the notion of behaviour formally. We formalized a feature as an increment in the possible behaviour of a global automaton. A specific specification style was a prerequisite to construct an unambiguous association of the state transitions to the different features.

The result of our analysis is a list of pairs of transitions from different features which may possibly interact. This list may be condensed to a list of pairs of features which may possibly interact. Such a condensed list is of value for the management of detected possible feature interactions: for each entry, we need to find an expert who knows both features and can determine how the interaction is resolved best. This is an improvement to the current situation. Up to now we need an expert who knows all features in the system, if we want to add one new feature. And the larger the system becomes, the harder it is to find such an overall expert.

Next, we sketched an approach for the resolution of detected feature interactions,

[6] An overview of this study is more widely available [BrGo94b].

which we have detailed in [BrGo94a]. We put up a precedence matrix for features and thereby specify a partial order on them. This allows us to disable unwanted transitions. There is also the possibility to specify new behaviour to resolve an interaction. Furthermore, we discussed how our generic formalism can be mapped onto real FDTs.

Some of the detection criteria are already supported by an automated tool. A first case study based on the FDT Estelle used a simple global service specification of a telephone switching system. Our detection tool did also find two (relatively harmless) interferences which escaped us while we specified this simple example. Currently, we are working on a second case study which takes into account the distribution aspect, too.

This paper describes on-going work. We are working on even more detailed detection criteria. They would reduce the manual work of, e.g., dismissing certain classes of harmless interactions. Parallely, we plan to incorporate more criteria into our detection tool, and we will continue to perform case studies. The application of our method to other FDTs than Estelle (e.g., SDL) should be straightforward.

REFERENCES

[BCN95] Bucci, G., Campanai, M., and Nesi, P. *Tools for specifying real-time systems*. Real-Time Systems Journal **8**, 117–172 (1995).

[Bo+89] Bowen, T. F. et al.. *The feature interaction problem in telecommunication systems*. In "Seventh IEEE International Conference on Software Engineering for Telecommunication Systems" (July 1989).

[BoVe94] Bouma, L. G. and Velthuijsen, H., editors. *Feature Interactions in Telecommunications Systems*. IOS Press, Amsterdam (1994).

[Bre95] Bredereke, J. *Automata-theoretic criteria for feature interactions in telecommunications systems*. Tech. Rep. 273/95, Univ. of Kaiserslautern, Dept. of Comp. Sci. (1995).

[BrGo94a] Bredereke, J. and Gotzhein, R. *A case study on specification, detection and resolution of IN feature interactions with Estelle*. Tech. Rep. 245/94, Univ. of Kaiserslautern, Dept. of Comp. Sci. (May 1994).

[BrGo94b] Bredereke, J. and Gotzhein, R. *Specification, detection and resolution of IN feature interactions with Estelle*. In Hogrefe, D. and Leue, S., editors, "Seventh International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols — FORTE '94, Proceedings", pp. 366–368, Berne, Switzerland (4–7 Oct. 1994).

[CaVe93] Cameron, E. J. and Velthuijsen, H. *Feature interactions in telecommunications systems*. IEEE Commun. Mag. **31**(8), 18–23 (Aug. 1993).

[CCI87] CCITT SG X, Contribution Com X-R15-E. *Recommendation Z.100: Specification and Description Language SDL* (1987).

[DuVi92] Duran, J. M. and Visser, J. *International standards for Intelligent Networks*. IEEE Commun. Mag. **30**(2), 34–42 (Feb. 1992).

[GoBr95] Gotzhein, R. and Bredereke, J., editors. *5. GI/ITG Workshop on Formal Description Techniques for Distributed Systems*, Univ. of Kaiserslautern, Dept. of Comp. Sci. (22–23 June 1995). URL http://www.informatik.uni-kl.de/aggotz/fachgespraech95.

[HoUl79] Hopcroft, J. E. and Ullman, J. D. *Introduction to Automata Theory, Lan-*

guages and Computation. Addison-Wesley (1979).

[ISO89] ISO/TC 97/SC 21, ISO 9074. *Information Processing Systems — Open Systems Interconnection — Estelle: A Formal Description Technique Based on an Extended State Transition Model* (1989).

[ITU92] ITU-T, Recommendation Q.1201. *Principles of Intelligent Network Architecture* (Oct. 1992).

[ITU93a] ITU-T, Recommendation Q.931. *DSS 1 — ISDN User-Network Interface for Basic Call Control* (Mar. 1993).

[ITU93b] ITU-T. *Q12xx-Series Intelligent Network Recommendations* (1993).

[SiSt93] Sijelmassi, R. and Strausser, B. *The PET and DINGO tools for deriving distributed implementations from Estelle*. Comp. Networks and ISDN Syst. **25**(7), 841–851 (Feb. 1993).

[ThBr95] Thees, J. and Bredereke, J. *A tool for the analysis of feature interactions in IN*. In Gotzhein and Bredereke [GoBr95], pp. 199–208. URL ftp://www.informatik.uni-kl.de/aggotz/fachgespraech95/thees.ps.gz (in German).

[The95] Thees, J. *Design and implementation of a tool for the analysis of feature interactions in Estelle specifications*. Master thesis (in German), Univ. of Kaiserslautern, Dept. of Comp. Sci. (Apr. 1995).