Automata-Theoretic vs. Property-Oriented Approaches for the Detection of Feature Interactions in IN

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 Intelligent Networks obstructs more and more the rapid introduction of new features. Detecting such feature interactions turns out to be a big problem. The size of the systems and the sheer computational complexity prevents the system developer from checking manually any feature against any other feature. We give an overview on current (verification) approaches and categorize them into property-oriented and automata-theoretic approaches. A comparison turns out that each approach complements the other in a certain sense. We propose to apply both approaches together in order to solve the feature interaction problem.

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 thousands of features have been developed [BrAt94, DSW$^+$94, BDC$^+$89] (the DMS-100 switch of Northern Telecom alone supports over 800 telephony features [Nor92]). Therefore the probability is high that augmenting such a system by one more feature will influence another feature, especially in an undesired way. This is called a feature interaction. Both the notion of feature and the notion of feature interaction are used quite fuzzily and informally most of the time. This does not make a solution of the

[1]Example: the features defined in the ITU-T recommendations for Intelligent Networks (IN) [ITU93a].

problem easier. (Our formalization in [Bre95, Bre96] is intended as an attempt to improve this situation.)

A simple example of a feature interaction is the joint subscription to Originating Call Screening (OCS) and Abbreviated Dialling (ABD). The OCS feature allows to enter directory numbers into a list, e.g., by a parent. The feature aborts a call if a number is dialled, e.g., by an adolescent, which found on this list. The ABD feature translates a list of short, i.e. one-digit, numbers into full directory numbers. If both features are executed in the wrong order, i.e., OCS before ABD, the OCS feature can be circumvented by the adolescent. He registers the desired number as an abbreviated number, and then dials the abbreviated number, reaching a destination which he should not reach. A resolution to this problem is to execute the OCS feature only after all number translating features have been executed.

Another example of a feature interaction may occur between Credit Card Calling (CCC) and VoiceMail. Pressing the # button is interpreted by CCC as a signal to continue with a new call without redoing authorization. But the same symbol is recognised by a VoiceMail box as a command. It can be interpreted only once; no matter which feature interprets it, the other one cannot work properly.

Some features are just incompatible, e.g., Call Waiting (CW) and Call Forward on Busy (CFB). When a customer is already engaged in a conversation, and another call arrives, the CFB feature sends a signal tone. The customer may then switch to this other call (and back again). The CW feature is activated in a busy state, too. If a call arrives when the customer is busy, this call is forwarded to another directory number. Only one of the two features can get control over the newly arriving call, and the other one must fail to do so.

As has been pointed out by Cameron and Velthuijsen [CaVe93], it is crucial to define the problem before trying to solve it. They 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." Other authors provide similar descriptions, compare e.g. Kimbler and Velthuijsen [KiVe95, page 2], Aho and Griffeth [AhGr95], and Mierop et al. [MTJ93].

Even if they cannot present 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 [Bre95, Bre96], we developed a formal definition which is based on the central notion of the change in behaviour. We restricted our formalization of feature interactions to the well-investigated functional aspects of the behaviour[2], and made use of automata [HoUl79].

[2]This excludes, for example, the treatment of a case where one feature is impaired because another feature causes an overload on some shared resource.

<table>
<tr><td>avoidance</td><td colspan="2">• open telecommunications platforms
• signalling protocols
• service creation guidelines
• service creation processes
• service management processes
• ...</td></tr>
<tr><td>detection</td><td>• verification tools
• validation tools
• simulation tools
• ...</td><td>• behaviour analysis</td></tr>
<tr><td>resolution</td><td>• redesign
• customer care</td><td>• FI manager
• negotiation
• event-based resolution
• ...</td></tr>
<tr><td></td><td>off-line</td><td>on-line</td></tr>
</table>

Figure 1: Categorization of approaches to addressing feature interactions from [BoVe94].

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

Bouma and Velthuijsen [BoVe94] have proposed a categorization of approaches to addressing feature interactions, which may be found in Figure 1. *Avoidance* would be the best way to deal with the problem; unfortunately, a full coverage is impossible due to the diversity of the problem. Therefore, we (additionally) need a *detection* step, and then a *resolution* step. Both steps may be performed either *off-line* or *on-line*. Experience has shown that detection is harder than resolution in the off-line case, and resolution is harder than detection in the on-line case.

In the remainder of this paper, we will concentrate on the *off-line verification approaches* for the detection of feature interactions. We will categorize these approaches further into property-oriented approaches (Section 2) and automata-theoretic approaches (Section 3). In Section 4, we will propose to integrate these approaches, and we will present conclusions in Section 5.

2 Property-Oriented Approaches

In Section 2.1, we will give a sketch of the mostly used property-oriented approach; in Section 2.2, we will list several detailed references and describe the differences among these approaches; and in Section 2.3, we will discuss some limitations of the property-oriented approach.

2.1 The Standard Approach

The combinatory complexity of the problem (compare Section 1) strictly demands efficient tool support. Automated tool support is only possible if the entire system is described formally. Therefore, a formal description of the telephone system and of its features is a basic prerequisite for any further work.

The level of such a description varies from one approach to another. Most work is done on either the global functional plane (GFP) or on the distributed functional plane (DFP) of the IN conceptual model (INCM) [ITU92].

The standard property-oriented approach requires two separate formal descriptions. One is written in a lower-level, constructive[3] description technique. This may be, e.g., SDL [CCI87, FaOl94], Estelle [ISO89], LOTOS [ISO88], or Promela [HoPe95, Gré95]. Often, this description is used later to generate implementations automatically. Such a description contains sufficient details which enable a compiler tool to generate relatively efficient code. On the other hand, there is still a formal semantics for the description which allows the further verification steps.

The second formal description is on a higher abstraction level and more property-oriented. Generally, it abstracts away from the "how" of internal details and tries to take a users' view of the system. The expectations of the customers (and of the service provider) at the system and at the additional features should be expressed as directly as possible. The description language is mostly some variant of temporal logic [Pnu77, MaPn92]. Sometimes, LOTOS is used at this level, too. As with many property-oriented description approaches in other fields of application, this description needs not to be complete in most FI detection approaches. The specifier just writes down any relevant property he can think of, and any property he has found out by interviewing customers.

The next step is to define an "implements" relation between the lower-level and the higher-level specification of the system. This relation decides mathematically if the lower-level specification fulfills all properties from the higher-level specification. (Compare Figure 2.) Often, this "implements" relation is only assumed implicitely by the authors. It exists nevertheless.

Of course we expect that this "implements" relation is fulfilled. This can be checked either by mathematical proof (manually or with tool support) or by model checking (with tool support). A model checking tool generates every possible execution trace that the lower-level description is capable of, and checks if all properties are true in all of the states. An alternative approach to this is testing, where only a subset of the execution traces is

[3] "Constructive" means "executable" or that an implementation can be derived directly.

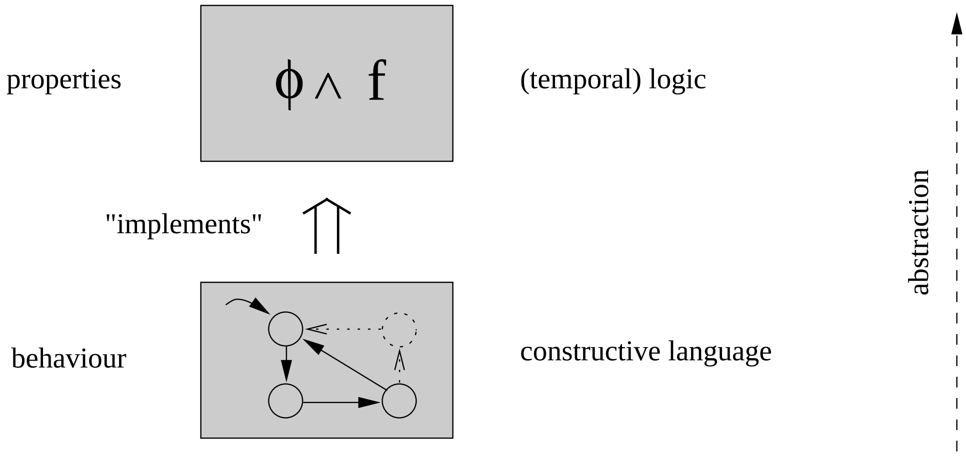

Figure 2: The property-oriented verification approach.

investigated. Obviously, this is weaker than a proof, but it may be the only choice left when the state space is too big which has to be explored.

The interesting point comes when we add a new feature f to the system. We do this by simultaneously adding a new part to the lower-level description and adding some more properties to the higher-level description. Now, the extended lower-level description should still satisfy all the properties of the higher-level description, and additionally all the new properties of the new feature. This is checked by the same methods as above.

If we cannot prove the "implements" relation anymore, we may get hints instead in which situations which property does not hold. E.g., a model checker may print out an execution trace where a specific property has been violated. Such information may be used for the (manual) analysis and resolution of the problem.

2.2 Overview of Several Specific Approaches

In the approach of Lin and Lin [LiLi94], specifications are written in temporal logic, implementations are written in Promela, and the tool SPIN allows model-checking of the properties.

Braithwaite and Atlee [BrAt94] describe a telephone system by automata, and they perform an exhaustive state space analysis. They look for some automata-theoretic properties as non-determinism in the processing as well as for the violation of explicitely specified properties (assertions). They restrict themselves to certain common classes of of feature interactions, and they use a semi-formal tabular specification technique. There is no tool support up to now.

Gammelgaard and Kristensen [GaKr94] write specifications in a more restricted form (only predicates), and implementations are expressed by transition rules. The strict formalization is still for future research.

Combes and Pickin [CoPi94] write their lower-level descriptions in SDL. Since message sequence charts (MSCs) [ITU93b] are a standard and well-supported method of specifying test cases for SDL specifications, they write their higher-level specifications in temporal logic and translate them to message sequence charts. The tool GEODE [Enc89] then allows model-checking of the properties. It is planned to translate the higher-level specifications to so-called SDL-observers which allow for more expressive power than MSCs.

Bouma and Zuidweg [BoZu93, BoZu92] write specifications in branching-time temporal logic, and they write implementations in Lotos. (Manual) model checking is done with the tool LITE. Korver [Kor93] extends this work by automatic model checking with the tool Cæsar/Aldébaran [FGM+92].

Faci and Logrippo [FaLo94, Fac95, BoLo93] write both the specifications and the implementations in Lotos.

Blom et al. [BJK94] have only treated specifications up to now, and no implementations. They express everything in the Lamport's temporal logic of actions (TLA) [Lam91]. They check by logical reasoning if the combined system can have a legal implementation. The SCORE project [SCO95], too, proposes the idea that one can cross-check the high-level specifications of the features beforehand. If the properties are inconsistent when put together, one can save the pains of looking at the implementations.

2.3 Limitations

The property-oriented approach is promising, but it also has some limitations and prerequisites. If the proof is done by logical reasoning, the work is painstaking even with tool support, and the method is only applicable to small systems. Symbolic model checking, used, e.g., in the SPIN tool (see, e.g., [HoPe95, Gré95, LiLi94]), has brought some progress with this respect, lately.

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. E.g., Dankel et. al. [DSW+94] present a natural language-based system that converts English-based telephony requirements into a knowledge-based representation. Goals of this conversion are, according to them, to create an unambiguous understanding of the requirements of the described telephony feature, and to create [at least] less ambiguous written requirements documents. Even such massive methodical and tool support does not allow them to write property-oriented specifications which are complete.

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, in order to specify a Terminating Call Screening (TCS) feature. The TCS feature allows the customer to specify a list of callers by whom he does not want to be disturbed. Then, add a new Call Forwarding (CF) feature to this system. When a caller A calls a callee B, then B may forward this call to a customer C (compare Figure 3). This way B may become a caller, too, in a sense. Who is the caller for C? Is it A, who initiated the original call, or is it B, who initiated

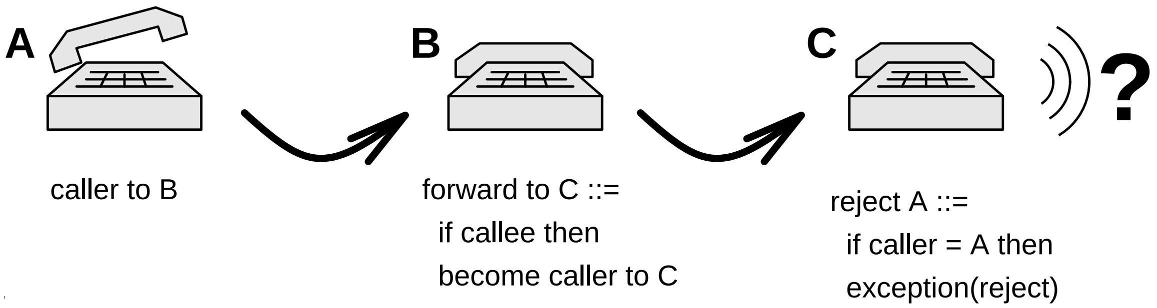

Figure 3: Feature interaction between call forwarding and terminating call screening.

the forwarding call? 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. As a consequence, customer A may get through to customer C, who might feel upset about the failure of his TCS feature.

3 Automata-Theoretic Approaches

Besides the property-oriented approaches discussed in Section 2, there is also a second line of ideas to tackle the detection of feature interactions by a verification approach. It does not use two descriptions of the system on different levels (compare Figure 2), but only a single constructive description.

Any formal specification may be investigated for properties which are generally undesirable. E.g., a specification which allows the system to deadlock is incorrect with high probability. Similarly, an unspecified reception[4] should never happen. But in the specific application domain of Intelligent Networks, one can determine more specific automata-theoretic criteria that direct our attention to both errors and to feature interactions in the system specification.

In Section 3.1, we will give an overview of some first ideas on such an automata-theoretic approach; in Section 3.2, we will describe a formal framework for this approach; and in Section 3.3, we will discuss the achievements and limitations of the automata-theoretic approaches.

3.1 Overview

Cheng [Che94] proposes to use a specific layered architecture for the specification of the system (in LOTOS). For the analysis of interactions, he concentrates on the links between these layers. He assumes that an interaction occurs if there exists non-determinism between the entrance events of different features.

[4]An unspecified reception is similar to a deadlock in that there is no more progress possible, but the message queues are not empty. Often, a missing reception transition is the cause.

Besides for non-determinism, Ohta and Harada [OhHa94] check a state transition system for several general criteria, such as deadlocks, unreachable transitions, undefined states, duplicated identifiers etc.

Dworak [Dwo91] proposes informally to look for non-determinism and to look for state combinations and transitions which violate constraints. (The latter is some rudimentary kind of property-oriented approach as discussed in Section 2, again.)

Lee [Lee92] uses an object oriented specification language and analyses if two features share the same data, which might be a source of feature interaction.

Kuisch et al. [KJMK93] analyse (informally) if two features can be active at the same time and can access the same resources. This idea is extended by Kimbler, Kuisch et al. [KKM94] who categorize features and then check only pairs of features in interaction prone category combinations. The aim of this idea is to reduce the manual part of the analysis. Kimbler and Søbirk [KiSø94] combine the idea with an analysis of shared data between features for those features which are used most often together.

3.2 An Approach Based on a Formal Definition for FI

Automata-theoretic criteria for feature interactions (and other errors) have the advantage that they can be checked by relatively simple means.[5] E.g., non-determinism among the transitions of a state transition system may be found by comparing all transitions pair-wise. No execution sequences have to be considered or constructed for this. A disadvantage of the approaches described in the previous subsection is that they don't really look for feature interactions but only for some criteria which may indicate feature interactions. If a feature interaction occurs, these criteria need not neccessarily be fulfilled, so that some feature interactions may not be found. These approaches are able to find some critical spots, but not all of them.

Therefore, we developed a formal definition of the notion of "feature" and "feature interaction" in [Bre95, Bre96], at least for the functional aspects of a system. The definition is based on the notion of "behaviour of a feature" which we discussed in Section 1. The telephone system is formalized by a global, structured automaton. Both the (data) structure of the automaton and the set of state transtions is partitioned into different features. Adding a feature is realized by adding transitions and (maybe) extending the state space. The behaviour of a feature is defined by the set of execution traces in which some transition of this feature takes part. A feature interaction occurs if some other feature modifies this behaviour of the feature. Based on the formal definition, we derive more sophisticated detection criteria, which, e.g., make advantage of the typical structure of a call processing automaton.

All the previous approaches had the disadvantage that they could not guarantee that no more feature interactions are present in the system after the detection scheme has been applied and after all the found feature interactions have been resolved. This is different with our approch. It allows to prove that no more feature interactions are present in the

[5]As long as we don't have to explore the entire state space, which may lead to an exponential growth in processing time with increasing size of the specification.

behaviour of the system. This achievement is only restricted by the limits of our definition of feature interaction.[6]

After the feature interactions have been detected, it is comparably easy to resolve them. Details on the formal resolution procedure may be found in [Bre95, Bre96].

Our formal framework can be mapped onto real formal description techniques. There is already some tool support, applied to the FDT Estelle. The tool CONFINE detects non-determinism between Estelle transitions of different features [ThBr95, The96]). (Such non-determinism plays an important role in the detection criteria.) Furthermore, CONFINE also detects and automatically eliminates Estelle transitions which have become non-executable because they are completely overlapped by new, higher-priorized Estelle transitions. (This may happen due to feature interaction resolution steps.) Currently, we work on incorporating more criteria into CONFINE [Bar96]. The tool set Pet/Dingo from NIST [SiSt93] automatically generates executable code from an Estelle specification and animates the execution.

A first case study [BrGo94, BrGo95] 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, and which takes into account the distribution aspect, too [Ill95, Jer96].

3.3 Achievements and Limitations

The result of an automata-theoretic analysis is a list of pairs of features which may possibly interact. If there are sufficiently few cases left to analyze, the traditional manual analysis can become effective again.

Such a 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.

Since the detection criteria presented in [Bre95, Bre96] render *all* potential feature interactions, we have the chance to catch *all* feature interactions, even those that would go unnoticed by property-oriented verification approaches because of incomplete descriptions as described in Section 2.3.

[6] The most notable restriction is that we excluded non-functional aspects such as an overload on some shared resource from the definition.

property-oriented approach	**automata-theoretic approach**
two different specifications	one specification
adding features incrementally	adding features incrementally
needs formal specifications	needs a formal specification
finds violations of users' intentions	only *possible* interactions found
incomplete list of properties / feature interactions	can find *all* (possible) feature interactions
expensive proof of properties	computationally cheap[a]

[a] if no complete state space exploration

Table 1: Comparison of the two verification approaches

The main limitation of the automata-theoretic approach is that the detection criteria point out *possible* feature interactions, but that they cannot tell if the resulting behaviour is undesirable or even desired. The reason for this is that there is no additional information on the intentions of the customers and of the provider, as with the property-oriented approach discussed in Section 2.

Most automata-theoretic criteria are computationally cheap, e.g., the check for non-determinism (compare the beginning of Section 3.2). Some of the more advanced automata-theoretic criteria may become computationally expensive. E.g., a check if the system can deadlock may need to explore the entire state space, which can be huge even for small systems.

4 Integrating the Approaches

Table 1 compares the key features [sic!] of the two approaches discussed in Sections 2 and 3. It can be seen that both approaches complement each other in a certain sense. On the one hand, the automata-theoretic approach can only find possible interactions, while the property-oriented approach can decide if actually harm is done. On the other hand, only the automata-theoretic approach can find all interactions. Furthermore, it is computationally cheaper.

Kimbler, Kuisch et al. [KKM94] (compare Section 3.1) have already proposed informally that the manual analysis should not be applied to any combination of features but only to the interaction prone ones.

The property-oriented approach allows to check bigger specifications than manual work, but it puts still severe restrictions on the size of the system which can be analyzed. This leads to the idea that the property-oriented verification should be restricted to combinations of features which are interaction prone, too, and that it need not be applied for those combinations where we can exclude feature interactions by simpler means. These simpler means can be automata-theoretic criteria. At least our approach [Bre95, Bre96] can definitely exclude many combinations of features which surely cannot interact.

Automata-theoretic criteria can determine which pairs of features cannot interact by any means. For example, if two features work in different "parts" of the system, are never active at the same time, and have no other "relation among each other", then we can be sure that they don't interact.

Such formal criteria may restrict the $n \times n$ space of possibly interacting features to a much smaller list of possibly critical pairs of features. Then, this list can be analyzed much more thoroughly. For this, we can apply the property-oriented approach.

Therefore, both the automata-theoretic and the property-oriented approach should be applied together in order to achieve optimal results.

Of course, there is still the problem that the property-oriented approach may use an incomplete property description or one that has become inacccurate, as discussed in Section 2.3. Nevertheless, the use of a second, independent and complementary approach has lowered the risk, and the higher degree of mechanisation which is possible now allows for a deeper manual study of remaining problems.

5 Conclusions

We have introduced to the feature interaction problem in Intelligent Networks, which obstructs more and more the rapid introduction of new features into telephone switching systems. Since those feature interactions cannot be avoided (entirely), they have to be detected and then resolved. The detection of feature interactions turns out to be the hardest part in this, if we want to tackle them before the system is installed. The size of the systems and the sheer computational complexity prevents the system developer from checking manually any feature against any other feature. Most approaches to the detection of feature interactions are off-line verification approaches. We categorized them into property-oriented approaches and into automata-theoretic approaches. We sketched both approaches and presented an overview of previous work.

We compared both verification approaches, the results may be found in Table 1. It turned out that each approach complements the other in a certain sense.

This led to the idea that the computationally cheaper automata-theoretic approach should be applied first, in order to reduce the number of pairs of features which can possibly interact, and then the property-oriented approach should be applied to find out if the found possible feature interactions are really undesired.

Therefore, both approaches will have to be applied together in order to solve the feature interaction problem.

References

[AhGr95] Aho, A. V. and Griffeth, N. D. *Feature interactions in the global information* *infrastructure*. In "Proceedings of the 1995 Conference on Foundations of Software Engineering" (1995).

[Bar96] Barthel, D. *Implementation of criteria for the detection of feature interactions*. Master thesis (in German), Univ. of Kaiserslautern, Dept. of Comp. Sci. (1996). To appear.

[BDC+89] Bowen, T. F., Dworack, F. S., Chow, C. H., Griffeth, N., Herman, G. E., and Lin, Y.-J. *The feature interaction problem in telecommunication systems*. In "Seventh IEEE International Conference on Software Engineering for Telecommunication Systems" (July 1989).

[BJK94] Blom, J., Jonsson, B., and Kempe, L. *Using temporal logic for modular specification of telephone systems*. In Bouma and Velthuijsen [BoVe94], pp. 197–216.

[BoLo93] Boumezbeur, R. and Logrippo, L. *Specifying telephone systems in LOTOS*. IEEE Commun. Mag. **31**(8), 38–45 (Aug. 1993).

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

[BoZu92] Bouma, W. and Zuidweg, H. *Formal analysis of feature interactions by model checking*. In "1st International Workshop on Feature Interactions in Telecommunications Software Systems", St. Petersburg, Florida, USA (Dec. 1992).

[BoZu93] Bouma, W. and Zuidweg, H. *Formal analysis of feature interactions by model checking*. Technical Report TI-PU-93-868, PTT Research, The Netherlands (1993).

[BrAt94] Braithwaite, K. H. and Atlee, J. M. *Towards automated detection of feature interactions*. In Bouma and Velthuijsen [BoVe94], pp. 36–59.

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

[Bre96] Bredereke, J. *Formal criteria for feature interactions in telecommunications systems*. In Iversen, V. B. and Nørgaard, J., editors, "Intelligent Networks and New Technologies". Chapman & Hall (1996). To appear.

[BrGo94] 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).

[BrGo95] Bredereke, J. and Gotzhein, R. *Specification, detection and resolution of IN feature interactions with Estelle*. In Hogrefe and Leue [HoLe95].

[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).

[Che94] Cheng, K. E. *Towards a formal model for incremental service specification and interaction management support*. In Bouma and Velthuijsen [BoVe94], pp. 152–166.

[CoPi94] Combes, P. and Pickin, S. *Formalization of a user view of network and services for feature interaction detection*. In Bouma and Velthuijsen [BoVe94], pp. 120–135.

[DSW+94] Dankel II, D. D., Schmalz, M., Walker, W., Nielsen, K., Muzzi, L., and Rhodes, D. *An architecture for defining features and exploring interactions*. In Bouma and Velthuijsen [BoVe94], pp. 258–271.

[Dwo91] Dworak, F. S. *Approaches to detecting and resolving feature interactions*. In "Proceedings of the IEEE Global Telecommunications Conference GLOBECOM'91", vol. 2, pp. 1371–1377, Phoenix, Arizona (2–5 Dec. 1991).

[Enc89] Encontre, V. *Geode: An industrial environment for designing real time distributed systems in SDL*. In "4th SDL Forum, Proceedings". North-Holland (1989).

[Fac95] Faci, M. *Detecting Feature Interactions in Telecommunications Systems Designs*. PhD thesis, University of Ottawa, Dept. of Comp. Sce. (1995).

[FaLo94] Faci, M. and Logrippo, L. *Specifying features and analysing their interactions in a LOTOS environment*. In Bouma and Velthuijsen [BoVe94], pp. 136–151.

[FaOl94] Faergemand, O. and Olsen, A. *Introduction to SDL-92*. Comp. Networks and ISDN Syst. **26**, 1143–1167 (1994).

[FGM+92] Fernandez, J.-C., Garavel, H., Mounier, L., Rasse, A., Rodríguez, C., and Sifakis, J. *A toolbox for the verification of LOTOS programs*. In "14th International Conference on Software Engineering ICSE'14, Proceedings", Melbourne, Australia (1992).

[GaKr94] Gammelgaard, A. and Kristensen, J. E. *Interaction detection, a logical approach*. In Bouma and Velthuijsen [BoVe94], pp. 178–196.

[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.

[Gré95] Grégoire, J.-C., editor. *1st SPIN Workshop (SPIN95)*, Montréal, Canada (16 Oct. 1995).

[HoLe95] Hogrefe, D. and Leue, S., editors. *Formal Description Techniques VII*. Chapman & Hall (May 1995).

[HoPe95] Holzmann, G. J. and Peled, D. *An improvement in formal verification*. In Hogrefe and Leue [HoLe95].

[HoUl79] Hopcroft, J. E. and Ullman, J. D. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley (1979).

[Ill95] Illerich, J. *Design of a telephone switching system in Estelle*. Projektarbeit (in German), Univ. of Kaiserslautern, Dept. of Comp. Sci. (Oct. 1995).

[ISO88] ISO/TC 97/SC 21, ISO 8807. *Information Processing Systems — Open Systems Interconnection — Lotos: A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour* (1988).

[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. *Q12xx-Series Intelligent Network Recommendations* (1993).

[ITU93b] ITU-T. *Recommendation Z.120: Message Sequence Chart (MSC)* (1993). Revised Version.

[Jer96] Jerusalem, D. *Extension of a telephone switching system in Estelle*. Projektarbeit (in German), Univ. of Kaiserslautern, Dept. of Comp. Sci. (1996). To appear.

[KiSø94] Kimbler, K. and Søbirk, D. *Use case driven analysis of feature interactions*. In Bouma and Velthuijsen [BoVe94], pp. 167–177.

[KiVe95] Kimbler, K. and Velthuijsen, H. *Feature interaction benchmark*. Discussion paper for the panel on benchmarking at FIW'95, Kyoto, Japan (Oct. 1995).

[KJMK93] Kuisch, E., Janmaat, R., Mulder, H., and Keesmaat, I. *A practical approach to service interactions*. IEEE Commun. Mag. **31**(8), 24–31 (Aug. 1993).

[KKM94] Kimbler, K., Kuisch, E., and Muller, J. *Feature interactions among pan-European services*. In Bouma and Velthuijsen [BoVe94], pp. 73–85.

[Kor93] Korver, H. P. *Detecting feature interactions with CÆSAR and ALDÉBARAN*. Report CS-R9370, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands (Dec. 1993).

[Lam91] Lamport, L. *The temporal logic of actions*. Report 79, Digital Equipment Corp., Palo Alto, California, USA (Dec. 1991).

[Lee92] Lee, A. *Formal specification — a key to service interaction analysis*. In "Proceedings of the Eight International Conference on Software Engineering for Telecommunication Systems and Services (SETSS 1992)", pp. 62–66 (30 Mar.–4 Apr. 1992).

[LiLi94] Lin, F. J. and Lin, Y.-J. *A building block approach to detecting and resolving feature interactions*. In Bouma and Velthuijsen [BoVe94], pp. 86–119.

[MaPn92] Manna, Z. and Pnueli, A. *The Temporal Logic of Reactive and Concurrent Systems*. Springer (1992).

[MTJ93] Mierop, J., Tax, S., and Janmaat, R. *Service interaction in an object-oriented environment*. IEEE Commun. Mag. **31**(8), 46–51 (Aug. 1993).

[Nor92] Northern Telecom. *DMS-100 Meridian Digital Centrex Library*, 50039.08/12-92 issue 1 ed. (1992).

[OhHa94] Ohta, T. and Harada, Y. *Classification, detection and resolution of service interactions in telecommunication services*. In Bouma and Velthuijsen [BoVe94], pp. 60–72.

[Pnu77] Pnueli, A. *The temporal logic of programs*. In "19th Annual Symposium on Foundations of Computer Science", pp. 46–57, Providence, RI, USA (1977). IEEE.

[SCO95] SCORE-Technology Transfer. *Service Creation in an Object-Oriented Reuse Environment*. Deliverable D409 – R2017/SCO/WP4/DS/P/029/b1, RACE Project 2017 (SCORE) (4 Jan. 1995).

[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 http://www.informatik.uni-kl.de/aggotz/fachgespraech95/thees.ps.gz (in German).

[The96] Thees, J. *Confine – A Tool for the Analysis of Estelle Specifications (Tutorial)*. Univ. of Kaiserslautern, Dept. of Comp. Sci. (in German) (1996).