What Use are Formal Design and Analysis Methods to Telecommunications Services?

Muffy Calder
Department of Computing Science
University of Glasgow
Glasgow, Scotland.
muffy@dcs.gla.ac.uk

Abstract. Have formal methods failed, or will they fail, to help us solve problems of detecting and resolving of feature interactions in telecommunications software? This paper contains a SWOT (*Strengths, Weaknesses, Opportunities*, and *Threats*) analysis of the use of formal design and analysis methods in feature interaction analysis and makes some suggestions for future research.

1 Introduction

Over the last few years, a very active research area has been the investigation of *formal methods*, or *formal description techniques*, for solving (some of) the problems of detecting and resolving of feature interactions in telecommunications software. The results of that research will not be enumerated here (see the last few Feature Interaction Workshops [12, 13, 15] for a good selection of papers in the area) but much of the research has focussed on the formal specification of services and features, and the verification of properties of those specifications.

While this research has been in response to a clearly identified need, after an initial flush of enthusiasm, many in the community are beginning to wonder if formal methods have failed, or will fail, to deliver. Some of the questions being asked are:

- can they ever scale up to industrial requirements,
- can they reveal unknown interactions,
- can they help us understand the nature of interactions, better than with informal techniques,
- can the benefits outweigh the costs,
- does the inherent complexity, particularly space complexity, prohibit effective use of automated reasoning tools,
- can they deal with the many difficult characteristics of the domain; e.g. distributed control and data, influences from environment, multi-vendor, multi-platform?

To an extent, these questions merely illustrate the well known tensions between *theory* and *practice*, or between *dreams* and *reality*. Some of them have been examined elsewhere, and specific remedies proposed (for example in [20, 30]). But, I suggest that the situation is somewhat more serious than has been acknowledged and warrants further debate. In particular, whereas some of the reservations behind the questions given above are specific to the field of feature interactions, others are more generic. This is worrying because formal methods have in several areas failed to deliver the dream that was promised to software engineering two decades ago. Does this mean they could, or even that they *must*, fail again here too?

This paper aims to open up a debate of this issue, beginning with a SWOT (*Strengths, Weaknesses, Opportunities*, and *Threats*) analysis of the use of formal design and analysis methods in feature interactions. This is followed by a brief overview of the broader roles of formal design and analysis in both software and hardware engineering, including a closer inspection of a few specific areas. In light of the analysis and the historical context, the final section explores possible new roles for formal design and analysis methods and suggests areas where we should be aiming our research efforts.

It is important to note that I have deliberately avoided the phrase "formal methods" in the title. This is because I want to avoid the conventional and narrow interpretation of formal methods as just *specification* and *verification*; rather I mean the employment of formal, symbolic notations, theories and tools for both *describing* and *prescribing* emergent as well as obligatory behaviour. Also, while I refer to the field as "feature interactions", I intend it to mean something broader than just detection and resolution; rather, it is intended to encompass service creation, analysis and subsequent system development, maintenance, evolution, and even de-commissioning.

2 SWOT Analysis

2.1 Strengths

- The process of developing formal models forces contexts and assumptions to be made explicit – confusion about these is often a source of unpredicted interactions.
- Descriptive models faithful to operational service behaviour can provide platforms for experimentation with and exploration of new and tentative behaviours, as well as for established and legacy systems.
- Abstract models may allow us to make generic definitions of (classes of) interactions.
- Automated analysis of models is possible through the use of symbolic simulation, reasoning and prototyping tools.
- Formal design and analysis methods are the basis of nearly all off-line approaches.

2.2 Weaknesses

- Finding appropriate levels of abstraction is very hard. Moreover, the chosen level of abstraction may prove to be inappropriate and subsequently there can be no effective utilisation of the analysis.
- The activity of formal modelling and analysis is very time consuming; this is traditionally a problem for industry, and increasingly one for the academic community.

- New features are usually non-conservative extensions of existing services; we do not generally have good mechanisms for dealing with this. (A particular case of this is referred to as non-monotonicity in [30].)
- There is little evidence of formal design and analysis methods uncovering unknown interactions.
- In a multi-vendor market, source specifications, whether informal or formal, may not be available.
- Many services and the systems in which they are components involve a high proportion of undocumented legacy code. This code may have to be re-engineered.
- There may be a large gap between *prescriptive* specifications of what services *should* do and what what they *actually* do.
- Most techniques have been designed for expressability rather than for tool support.
- There are widespread beliefs that specifications should be readable by a wide audience.
- There are widespread beliefs that a formal approach needs to be taken for a whole problem.
- Most formal analyses can only consider features pairwise.

2.3 Opportunities

- The problems have not been solved by other approaches.
- In a multi-vendor, multi-platform market, developers may realise that in order to integrate their products with those from other vendors, they require more formal descriptions from the other vendors.
- Recent (finite-state) tool developments are very promising, for example model-checkers [18] can now deal with an order of 10^{100} states.
- While formal techniques have been predominantly employed in off-line approaches, they might also be employed in on-line approaches.
- Projects may be able to identify smaller, crucial problems where it is cost effective to apply formal analysis techniques.
- Formal techniques may be most attractive when offered as one of several complementary techniques.

- Progress is simply too slow.
- On-line approaches may prove to be more adaptive and effective; for example, negotiating agents may become one such approach.
- Common architectures and changing underlying technology may make interactions irrelevant.

This analysis might at first sight appear to have omitted several generally accepted motherhoods. For example, some "standard" strengths such as the provision of unambiguous, rigorous descriptions are not included; nor are some of the traditional weaknesses, such as lack of evidence of ability to scale up, cultural resistance, inadequate tools, relationships between models in different formalisms, education, etc. These are deliberately omitted, since I believe that in the past, the community has been trying to captilise on the *wrong* strengths (and consequently weaknesses, opportunities, etc.). Moreover, under those assumptions, the techniques are almost *bound* to fail to deliver. For example, one often cited advantage of formal definitions is that there is "only one correct way to interpret the behaviour defined". Although this is a contemporary comment (and a direct quote, though it seems unfair to single out the authors), the sentiment has been discredited on many occasions. For example, during the 70's there was a raging debate about the meaning of the infamous stack abstract data type. Eventually it was found that the usual array and index implementation was not a valid implementation of the standard specification which included the equation *pop(push(s,x))=s*! Clearly this is not what the specifiers *meant* when they were writing defining the equations, and the debate was finally resolved by a general recommendation that initial algebra semantics is not appropriate for composite data types.

The last threat is a bit of a joke, nonetheless there is a serious aspect to it. In such a rapidly changing technology, it is quite possible that the problem will just go away before we solve it. We must remain vigilant to this possibility and aware that we are problem-driven. The application of formal design and analysis methods is *not* the end but the *means* to achieve better telecommunications services.

On a more optimistic note, it is almost inevitable that further problems will be thrown up by new technologies. For example, already the service management frameworks such as TINA (*Telecommunications Information Network Architecture*) and intelligent agents [17] present us with new challenges.

3 Historical Perspective

The formal methods *dreams* of the 70's and early 80's failed to make the promised impact on software engineering, as practised in most commercial software development. Although there have not been many successful adoptions of formalisms, such as context-free grammars and finite-state diagrams, they are in general not the consequences of the formal methods "movement". For example, consider compilers. While the impact of formal grammars and parser generators has been great, i.e. they are almost universal, formal programming language semantics and semantics-based compiler generators have made little impact outside academia. That is not to say that formal semantics are not useful, but that they have not had the commercial impact envisioned (for example, Java was developed and released without a formal semantics).

Many of the early proponents of formal design and analysis methods recognise that this failure can, in part, be attributed it to the "totalitarian" nature of the programme of the 70's and 80's. This programme called for the integration of formal design and analysis methods into every stage of the software lifecycle, i.e. into the requirements – specification – design – implementation – testing process. Particular emphasis was placed on the early requirements and specification stages, on totality (all aspects of a system must be formally specified), and formal transformations between each stage.

But, the community is beginning to move away from this approach. For example, in [10], Cliff Jones explains how his views have changed from the "austere suggestions" of over a decade ago, and Jackson and Wing comment that by "promoting full formalisation in expressive languages, formalists have unwittingly guaranteed that the benefits of formalisation are thinly spread". Each conclude that the trend now is towards *formal methods light*, where partiality and focussed application are key, with minimum emphasis on notational detail.

There are several such focussed applications; one notable example, which is very relevant to our domain, is the area of protocol validation and testing, e.g. [18]. Testing in this context has proved to be very important. Whereas the role of testing was minimised in the early approach to formal methods, (the emphasis was primarily at the front end of the lifecycle) in the protocol domain, *formally based* testing has turned out to be one of the greatest benefits of using formal design and analysis methods (for examples, see [4, 21]). Undoubtedly, this success has been largely a consequence of the development of tractable finite-state verification techniques.

Other successful applications include safety-critical applications (e.g. railway signalling [26], medical applications [19]), security protocols (state exploration [23], authentication logics [1], induction [27]), re-engineering large legacy astronomical software [28], and hardware verification [6]. We note that, again, many of these successes are a result of effective finite-state verification techniques.

The last area is very interesting because it raises some strong parallels (and differences) worth noting. While formally based hardware design and verification is a long-established field, recently a new enthusiasm, particularly from industry, can be detected as the challenge moves on from gate level behaviour to the *system level* design, i.e. system-chips and chip sets. Moreover, we can detect signs of the *formal methods light* approach here. For example, Gadi Singer, general manager of design at Intel says in [11] (my emphasis):

> We believe that the technology and the knowledge required to do verification *right* will be a differentiator among companies and will be a competitive advantage for Intel. We think that Intel will have the advantage ... because of the ability to put together a complete set of *complementing* technologies that will allow our designs to be more reliable.

Clearly, there are key similarities between the two fields and therefore we should learn from their experience. For example, some striking similarities are that components are being developed and offered by multiple vendors, and developers require well-designed architectures to ensure correct operation, as well as well-documented architectures which can interoperate with others. In common with other areas of software engineering, there is an increasing recognition of the need for reusable components, i.e. reusable cores and IP (*intellectual property*) blocks. To this end, more than 100 electronics firms have recently joined the VSI (*Virtual Socket Interface*)) Alliance – a group which aims to develop standards for interfacing IP blocks from multiple sources.

On the other hand, we must be aware that there are also some key differences. For example, unlike hardware design, features do not represent users' ultimate intentions; they are merely a way of achieving some of those intentions. Moreover, features are being added at an exponential rate, features extend systems in a non-conservative way, and hardware does not get re-released.

4 Discussion and Challenges

The original formal methods contribution to software engineering attempted to integrate formal analysis and design methods into every stage of the requirements – specification – design – implementation – testing process. I suggest that this is a narrow view of the role of formality and one which is *bound* to fail, largely because it fails to address the weaknesses identified earlier and in particular, because it does not target resources effectively and it presupposes that requirements are fixed in a predetermined and static way.

But, this limited role is not the only possible one; Pamela Zave remarked in [10] that:

> Finding the best way to use formal methods in an application domain is research, not development.

So, what kind of research is required in order to address the points raised in our analysis so far? What *use* are formal design and analysis methods to telecommunications services?

I suggest that they *can* be of great use, but we need to use formal design and analysis methods in ways which capitalise upon the strengths and opportunities identified earlier, while avoiding the weaknesses and threats. In order to achieve this, some recommendations for future research directions are described below. The description is not exhaustive and some issues are interrelated.

Experimentation

While we still require specifications to document interfaces, software components and IP blocks, we need to move beyond the desire for *prescriptive* specification, towards *experimental* modelling of emergent behaviour. This may require a change of "spirit" in the way we approach mathematical models: a mathematical model is a *prototype* and one in which we can *test*, *explore*, and *learn* about behaviours.

The feature interaction "problem" presents us with a unique domain where we not only seek to validate desired interactions, but most importantly we seek to uncover and resolve undesirable and *unpredicted* interactions. *Unpredictability* is key, and formal design and analysis methods have not traditionally been used in such an experimental context. By such a context, I mean both using a model to prove or disprove a hypothesis, as well experimentation with a model to derive, or uncover, further hypotheses. Formally based approaches to experimentation, rather like formally based testing, can lead the experimenter to uncover crucial properties, theorems or interactions. This is particularly relevant when dealing with emergent, or unknown behaviour, and I believe that this is exactly where formal techniques may offer us a significant gain over other techniques. We should therefore capitalise on this opportunity and capitalise on the strengths of formal modelling in this context.

Understanding the problem domain

Interactions may be uncovered and resolved through experimentation, as described above, or through instantiation of generic characterisations. The latter requires a deep understanding of the problem domain, and while there are several excellent informal characterisations of classes of interactions (e.g. [7]), and several specific formal characterisations within particular models (e.g. [5, 29]) we are still lacking good, generic, formal characterisations.

Levels of abstraction will be particularly important when making such generic definitions, as well as learning from other fields (both within software and other forms of engineering) where interaction and interoperability are key issues (do we want a virtual *feature* alliance?).

Modelling

We need to work on developing models which are abstract enough to uncover properties and commonalities, yet concrete enough to inform the operational world. We need models which inform us about what *is*, rather than what we would like, in an idealised world. Moreover, we need to target our efforts so that the formal models express critical or poorly understood components, and at times when the process of developing the model will help us to become more confident about both the individual component and the overall system. The emphasis should not be on complete models, but on those which are focussed on aspects where formal modelling and analysis will provide additional insight (c.f. the "filtering" approach of [21]).

This will involve identifying the relevant aspects, or *dimensions*, and most importantly, critical intersections of those dimensions. These dimensions may be generic, or domain-specific. Examples of the former include functional behaviour, component range (e.g. scheduler, router, etc.), software lifecycle, product lifecycle, user or network views, service environment, safety-criticality, and fault-tolerance; examples of the latter include billing and charging behaviour, data transformations, user intentions, and temporal relationships between features.

Analysis without behavioural models

Telecommunications services are rarely a "green field" site. It is much more likely that we are integrating, modifying and extending a mixture of existing software and new functionality. So, the existing software may well be poorly documented legacy code, or just third party software, and we need good ways of dealing with that. Approaches which depend on pre-existing behavioural models usually involve re-engineering, or specifying the legacy system. There appears to be little work on alternative approaches, though one notable example is the approach suggested in [25], where the legacy code is embedded in a transactional model.

The problem of dealing with legacy code is not the only motivation for analysis techniques which do not depend on behavioural models. For example in [21], Kimbler outlines the "vicious" circle of finding criteria for interaction analysis which depends on behavioural specifications.

On-line techniques

So far, formal methods have been used primarily for off-line interaction detection and resolution, and clearly more research effort should continue in this area. However, with the emergence of multi-vendor markets and evolving architectures, the need for on-line techniques seems certain to increase. How can formal design and analysis contribute to the design, implementation, and run-time functioning of on-line feature management? Here perhaps is an ideal application for a combination of experimental and formal approaches. For example, how can a formally based model of service behaviour inform an on-line feature manager, in real-time? What kinds of theories of services, call models, features and interactions would we need, and what kinds of analyses? How could we avoid the vicious circle mentioned above, and how and when would the feature manager make use of those theories? Would the techniques be tractable?

The approach of Aggoun and Combes in [3], which introduces the concepts of passive observers in the service creation environment and active observers in the service execution environment, with feedback from the latter to the former, is a good example of such a combination approach. Further research in this area is needed and could motivate the development of some quite novel roles for formal design and analysis in telecommunications services.

Acknowledgements

Some of these ideas have developed through discussions with Evan Magill and I would like to acknowledge his contribution; thanks also to Kris Kimbler and Tom Melham for their comments on an earlier draft of this paper.

References

[1] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols. The spi calculus. In Fourth ACM Conference on Computer and Communications Security. ACM Press, 1997. In press.

[2] A. Alfred, N. Griffith. Feature Interactions in the Global Information Infrastructure. In Proceedings of 3rd ACM Sigsoft Symp. on Foundations of Software Engineering, *Software Engineering Notes*, Vol. 20, No. 4, Oct. 1995.

[3] I. Aggoun and P. Combes. Observers in the SCE and the SEE to Detect and Resolve Service Interactions. In [15].

[4] N. Arakawa, M. Phalippou, N. Risser, and T. Soneoka. Combination of conformance and interoperability testing. *Formal Description Techniques*, pp. 397-412, V. M. Diaz and R. Groz (eds.), Elsevier Science Publishers, B.V. (North-Holland) 1993.

[5] P. Au and J. Atlee. Evaluation of a State-Based Model of Feature Interactions. In [15].

[6] G. Barrett. Model checking in practice: the t9000 virtual channel processor. *IEEE Trans. on Software Engineering*, volume 21, number 2, 1995.

[7] E.J. Cameron, N.D. Griffeth, Y.J. Lin, M.E. Nilson, W.K. Shnure, and H. Velthuijsen. A feature interaction benchmark in IN and beyond. In [13].

[8] E.M. Clarke and J.M. Wing. Formal Methods: State of the Art and Future Directions. Report by the Working Group on Formal Methods for the ACM Workshop on Strategic Directions in Computing Research, *ACM Computing Surveys*, vol. 28, no. 4, December 1996, pp. 626-643. Also CMU-CS-96-178.

[9] P. Combes and S. Pickin. Formalisation of a User View of Network and Services for Feature Interaction Detection. In [13].

[10] Formal Methods: Point-Counterpoint. Round table discussion on formal methods, pp. 18-30, *IEEE Computer*, April 1996.

[11] EETIMES, 26 January, 1998, TechWeb News. http://www.techweb.com/se/directlink.cgi?EET19980126S0017.

[12] Proceedings of International Workshop on *Feature Interactions in Telecommunications Systems II*, St. Petersburg, U.S.A., IEEE Communications Society, 1992.

[13] W. Bouma and H.Velthuijsen (eds.). *Feature Interactions in Telecommunications Systems II.* Proceedings of International Workshop, Amsterdam, IOS Press, 1994.

[14] K.E. Cheng and T. Ohta (eds.). *Feature Interactions in Telecommunications Systems III.* Tokyo, IOS Press, 1995.

[15] P. Dini, R. Boutaba, and L. Logrippo, (eds.) *Feature Interactions in Telecommunications Systems IV.* Montreal, IOS Press, 1997.

[16] M. Faci, L. Logrippo, and B. Stepien. Structural Modals for Specifying Telephone Systems. To appear in *Computer Networks and ISDN Systems.*

[17] N.D. Griffeth and H. Velthuijsen. The Negotiating Agents Approach to Runtime Feature Interaction Resolution. In [13].

[18] G. Holzmann. Design and Validation of protocols: a tutorial. *Computer Networks and ISDN Systems*, No. 25, pp. 981-1017, 1993.

[19] J. Jacky. Specifying a safety-critical control system in Z. *IEEE Trans. on Software Engineering*, volume 21, number 2, 1995.

[20] K. Kimbler. Addressing the Feature Interaction Problem at the Enterprise Level. In [15].

[21] F. Kristoffersen, L. Verhaard, and M. Zeeberg. Test derivation for SDL based on ACTs. *Formal Description Techniques*, pp. 381-396, V. M. Diaz and R. Groz (eds.), Elsevier Science Publishers, B.V. (North-Holland) 1993.

[22] F.J. Lin and Y-J. Lin. *A Building Block Approach to Detecting and Resolving Feature Interactions*, in [13].

[23] G. Lowe. Breaking and fixing the Needham-Schroder public-key protocol in CSP and FDR. In Proceeding of TACAS '96, *Lecture Notes in Computer Science*, volume 1055, pp. 147-166, 1996

[24] *Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour.* International Organisation for Standardisation. 1988.

[25] D. J. Marples, E.H. Magill, and D.G. Smith. An infrastructure for Feature interaction resolution in a multiple service environment - The application of transaction Processing techniques to the Feature Interaction Problem. In *TINA 95 Telecommunications Information Network Architecture Conference*, Melbourne, Australia, February 1995.

[26] M. Morley. Safety-level communication in railway interlockings. *Science of Computer Programming*, volume 29, number 1-2, July 1997.

[27] L. Paulson. Proving Properties of Security Protocols by Induction. pp. 70-83, Proceedings of the 10th Computer Security Foundations Workshop, IEEE, 1997.

[28] M. Stickel et al. The deductive composition of astronomical software from sub-routing libraries. In Proceedings of CADE 12, *Lecture Notes in Computer Science*, pp. 341-355, volume 814, Springer Verlag.

[29] M. Thomas. Modelling and Analysing User Views of Telecommunications Services. In [15].

[30] H. Velthuijsen. Issues of Non-monotonicity in Feature-Interaction Detection. In [14].