Immaturity and potential of formal methods:
A personal view

Luigi Logrippo
School of Information Technology and Engineering
University of Ottawa, Ottawa, ON Canada K1N 6N5
www.site.uottawa.ca/~luigi

ABSTRACT. After some remarks on the use of different formal methods in the area of feature interaction detection, the question of the use of these methods in telecommunications systems development is addressed. Nowadays there does not seem to be (yet) a compelling or general need for formal methods. Also these methods are hindered by their fragmentation and discontinuity. Still, it is concluded that the success of formal methods is inevitable in the long run, although the time frame for this success and which methods will prevail remain open questions.

1. Formal methods and the IN conceptual model

I start with some remarks on the need for different formal methods in the area of feature interaction detection. In the next section, I will claim that different methods are needed at the various stages of development of new systems. It will be seen that these needs lead to discontinuity and fragmentation in this area. Throughout, the point of view will be the one of off-line methods, which are the main area of application of formal methods.

The Intelligent Network conceptual model identifies four planes, corresponding to different abstract views of the IN capabilities. These views can be thought to correspond to four different stages in the design of a system. Feature interactions can manifest themselves in each plane or stage.

In IN's Service Plane, potential interactions may show up as plain contradictions in the conjunction of the descriptions of the features. Here is a very elementary example:

- Call Number Delivery stipulates that the number of the calling party be displayed at the called end
- Unlisted Number stipulates that certain numbers cannot be displayed

Hence an unlisted number should be displayed according to CND and not displayed according to UN, a contradiction that implies feature interaction. This can be detected by a simple theorem prover, but may elude some other methods mentioned below. It appears therefore that the theorem-proving approach should be exploited for finding feature interactions in this plane, and at the corresponding stage of design.

In IN's Global Functional Plane, we encounter feature descriptions in the full-call model. This was the model used in 1998's FI contest. Techniques such as Chisel [1], Use Case Maps [7][13], ANISE [12], Tussilago [4], relational methods [12], are appropriate to describe in various ways this model. It is interesting to note how different these approaches are. Some (such as Chisel and Use Case Maps) are graphical in nature and some detection may be done directly by following graphical clues. Others are related to theorem-proving approaches. Others yet are related to state exploration models. Each has its own strengths and weaknesses.

In IN's Distributed Functional Plane and Physical Plane, we find the half-call model that is the basis of numerous state-oriented detection methods. This is the model that was used in this year's FI contest.

Testing methods are useful throughout, since tests can show concrete situations where a feature interaction occurs, or can show that there is no feature interaction in certain important cases. Testing, in many forms and at many levels, is turning out to be an interesting minimum common denominator [8].

We can conclude that feature interactions (potential or actual) can manifest themselves in many different ways and that detection methods should make it possible to catch them in just as many ways.

2. Formal methods and the development stages of new telecommunications systems

In our experiences with industry concerning methods for feature design and feature interaction detection, we have encountered different attitudes, that at first might appear to be contradictory, but which in fact are more likely complementary. We use here the three-stage model adopted by ITU-T and by other telecommunications standard bodies. According to this model, the development of a standard is done in the following sequence of stages: 1) service descriptions, 2) definition of functional entities and message sequences, and 3) protocol and procedure specification. In collaboration with industrial partners, we have tried to match these three phases to the description techniques that were best known to us [2]. Our conclusions are summarized in Fig. 1. Use Case Maps, a graphical notation that describes partial orders among abstract responsibilities, is proving to be successful at many levels and for many types of users. Message Sequence Charts are a less abstract technique, since they define specific message sequences between specific components. We found LOTOS to be useful for the specification and validation of Use Case Maps, which can be translated directly into it [3]. SDL is useful afterwards because it is at a level close to implementation. And TTCN is useful as a test specification language.

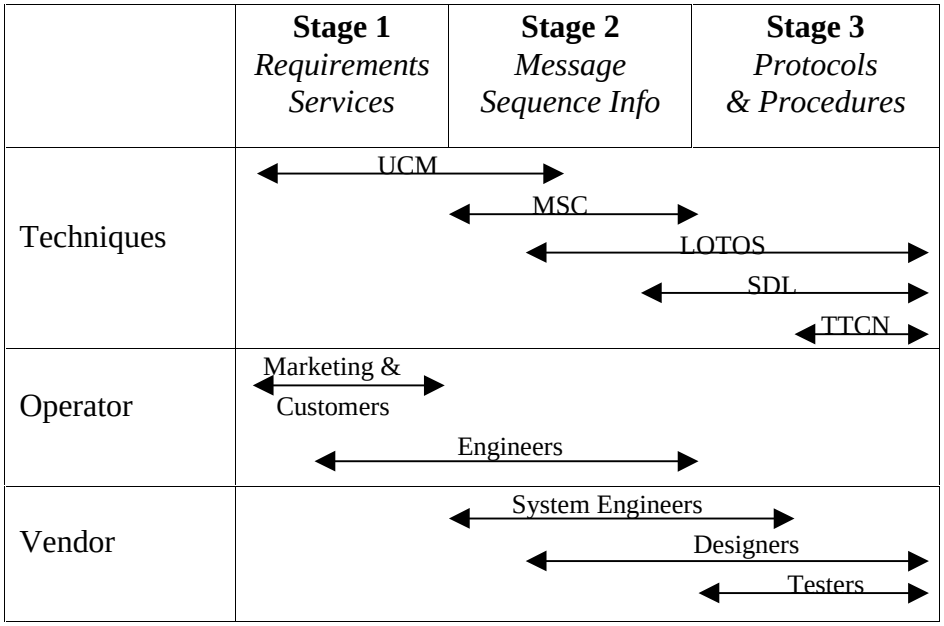

Figure 1: Relevant methods and teams involved in the three stages of the standard development process

This may be all well in principle, but too many methods and tools may be involved. Also although these techniques are partly complementary, unfortunately they do not always interwork well. They were developed by teams who were interested in developing one technique only, at the exclusions of others. This can be observed even for techniques that are used together, such as SDL, MSCs and TTCN: each of these techniques has its own semantic model, based on different concepts.

Every one of the techniques mentioned has easily recognizable strengths but also very definite weaknesses. For example, several designers have mentioned to me that they do not feel that SDL is a good language for design. Also, SDL has known semantic problems. LOTOS, on the other hand, lacks a clear path and helpful tools towards implementation. Can we ask a company to invest in learning so many languages, each of them with many peculiar aspects, for the reason that each can help part of the way, considering also the lack of 'bridging' methods and tools? Surely, variety of methods and instruments is common in engineering. However, such variety must be proven convincingly to be necessary and this takes much experimentation and time. And all methods must relate to a common set of basic concepts. Unfortunately, these have not yet been established in our area.

3. The use of formal methods in industry

Often I ask software designers and implementers, some of whom are familiar with formal methods (in fact, some of them are my former students who have written theses on the subject) why they are not using these methods. The most common answer cites a company view that the use of these methods would slow down the software development process, without sufficient advantages. It appears that the industrial environment on the whole is fairly satisfied with current software quality standards and lacks the motivation to try new methods in the area of validation and verification. This puts formal methods in a challenging position: at present, they are not considered to be sufficiently useful. They may, of course, have a better chance with the more complex software of tomorrow. However for this chance to be realized, their sophistication may have to increase more rapidly than the sophistication of the software they address, because only in this way a point may be reached in the future where the sophistication of the formal methods will be a good match for the sophistication of the software for which they will be needed at that time. Surely, the optimistic expectations that the formal methods community had some years ago have been frustrated...[8]

On the positive side, pockets of intensive use of formal methods exist in mission-critical systems, and very probably these pockets will increase in extent. They are the fine edge of penetration of formal methods [8].

Furthermore, we have to watch for certain possible developments. Piling up weak software on top of weak software may one day reach a breaking point. Will there be a widely recognized crisis in software quality, sufficiently serious to cause a change in attitudes? The telecommunications area is a prime candidate for the occurrence of such a crisis, with possible systematic failures in utilities, arising lawsuits, etc. De-regulation and unbridled competition have created a 'brave new world' in our area [9]. Many telecommunications products nowadays provide basic connectivity at reasonable prices. Hence the competition has shifted towards the provision of increasingly sophisticated services. Indeed, the integration of telecommunications with computing makes the creation of such services much easier (although risky). What can convince a user to subscribe to a new system? Better prices, of course, but also more sophisticated features. There is a race towards defining and implementing such features. Established standardization bodies such as ITU-T, ETSI and TIA, who have a slow process that allows time for careful scrutiny, are being pressured by industrial consortia to speed up. Old standards (produced in a very regulated telecommunications world) were far from perfect, newer ones might be much worse. The need for interconnectivity among operators in the provision of services will introduce further challenges.

Will some companies acquire an edge on the market by superior software reliability achieved by using formal methods? Will consumers challenge the current policy of the software industry: *'no warranty provided - use at your own risk - pay to get a better*

version ' ? We are all painfully aware of the fact that no other industry has such low standards.

4. A long-range view

I remain very optimistic about the eventual success of formal methods. And this for at least two reasons:

- Formal methods have already proven themselves, in the sense that, where they have been used, the quality of software has improved considerably. The only problem is that this improvement is not considered important at present, in most application areas. But this is sure to change. Mathematical methods have their role in all branches of engineering, so it is inevitable that they will earn a role in software engineering. Logic is to programming what calculus is to mechanics.
- The activity of interacting with a computer is formal by nature. One has to follow precise conventions whatever one wishes to do. Using conventions based on logic or algebraic methods has the benefit of making it possible to generate long chains of deductions, which provide much useful information on the characteristics of the system that is being built.

However, in order for formal methods to realize their potential, they will have to be closely integrated in the software development process. Their use cannot be a side trip. They must be included in a straight, inevitable line from design to implementation, to testing, to maintenance. One day, reading the warnings and error reports of the theorem provers will be just as inevitable as reading the warnings and errors of compilers is today. And in fact, syntax and static semantics analyzers are simple theorem provers.

At the design level, this straight line will include graphical methods, because they appeal to designers naturally. UML, Use Case Maps, ROOM, SDL, etc., include many excellent ideas, although unfortunately these too are different and not all well integrated. The graphical methods must allow for property-preserving transformations such as refinement, architectural rearrangements, and compilation into executable code. Protocols of the future will not be designed in detail, but they will be derived in a computer-assisted fashion from high-level designs, which probably will be expressed graphically.

Testing will be a type of theorem proving, and will be accompanied by proofs that the set of test cases is complete according to given criteria. Test cases will apply not only between the requirements and the implementation, but also between all the intermediate stages.

In order for this to be possible, formal methods will have to support powerful refinement and transformation rules. Process algebras include some theory to support refinements and transformations [5], but not to the extent necessary to make them truly useful. We must have a unified logic for data and control, so that there will be no barrier between them, and they can be transformed the one into the other. State exploration must find its place as a proof technique among others [10]. Also, as mentioned, users don't want to learn and use many different formalisms (consider the current situation: finite state machines, process algebras, inference rules, abstract data types, temporal logic, in all their dialects and implementations...). Some newer theories, such as linear logic, provide a unified logic for data and control. Will they support these processes better?

Building this straight line will not be a straight line itself... history has many twists and turns. The basic principle of steam machines was known 1700 years before such machines became of general use. Making a convincing case for industry, building and evolving all the necessary methods and tools, training software designers and implementors to use them, etc., may take a long time. Historically, new applications of mathematical methods have caused the development of new areas of mathematical thought, and this process is

continuing with applications in computing. New logic and algebraic paradigms, new language paradigms, new types of tools will have to be developed.

Information Technology is often said to be a fast-moving area. However, it is still promoting IP, C, Unix, and SDL, all invented in the seventies. Think of this: in 50 years we have managed to advance from machine code to C and C++. Many projects have perished during these years: many languages, operating systems, protocol families that cost untold efforts but were not successful. Some of them probably were too advanced for their time. Some of those ideas may be lost forever, made irrelevant by developments, others may resurface in various forms. It will be an interesting future...

References

[1] A. Aho, S. Gallagher, N. Griffeth, C. Schell and D. Swayne. SCF3TM/Scultor with Chisel: Requirements Engineering for Communications Services. In: K. Kimbler and L.G. Bouma (eds.) *Feature Interactions in Telecommunications and Software Systems V, IOS Press* (1998) 45-63.

[2] Amyot,D., Andrade, R., Logrippo, L., Sincennes,J., Yi, Z. Formal methods for mobility standards. Proc. of the 1999 IEEE Emerging Technologies Symposium on Wireless Communications and Systems. Richardson, TX, 1999

[3] D. Amyot, L. Charfi, N. Gorse, T. Gray, L. Logrippo, J. Sincennes, B. Stepien, and T. Ware, Feature Description and Feature Interaction Analysis with Use Case Maps and Lotos. This book.

[4] J. Blom. Formalisation of Requirements with Emphasis on Feature Interaction Detection. In: P.Dini, R.Boutaba, and L.Logrippo. *Feature Interactions in Telecommunication Network IV*. IOS Press (1997), 61-77.

[5] T. Bolognesi, J. v.d. Lagemaat, and C. Vissers, *LOTOSPhere: Software Development with LOTOS*, Kluwer, 1995.

[6] G. Bruns, P. Mataga, I. Sutherland, Features as Service Transformers. In: K. Kimbler and L.G. Bouma (eds.) *Feature Interactions in Telecommunications and Software Systems V*, IOS Press (1998) 85-97.

[7] R.J.A. Buhr, D. Amyot, M. Elammari, D. Quesnel, T. Gray and S. Mankovski, Feature Interaction Visualization and Resolution in an Agent Environment. In: K. Kimbler and L.G. Bouma (eds.) *Feature Interactions in Telecommunications and Software Systems V*, IOS Press (1998) 135-149.

[8] M. Calder, What use are design and analysis methods to telecommunications services? In: K. Kimbler and L.G. Bouma (eds.) *Feature Interactions in Telecommunications and Software Systems V*, IOS Press (1998), 25-31.

[9] J. Cameron, F. J. Lin, Feature interactions in the new world. In: K. Kimbler and L.G. Bouma (eds.) *Feature Interactions in Telecommunications and Software Systems V,* IOS Press (1998), 3-9.

[10] A.P. Felty and K. S. Namjoshi, Feature Specification and Automatic Conflict Detection. This book.

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

[12] M. Frappier, A. Mili, J. Desharnais, Detecting Feature Interactions on Relational Specifications. In: P.Dini, R.Boutaba, and L.Logrippo. Feature Interactions in Telecommunication Network IV. IOS Press (1997), 123-137.

[13] M.Nakamura, T.Kikuno, J.Hassine and L.Logrippo, Feature Interaction Filtering with Use Case Maps at Requirements Stage. This book.

[14] K.J.Turner. Validating Architectural Feature Descriptions Using LOTOS. In: K. Kimbler and L.G. Bouma (eds.) Feature Interactions in Telecommunications and Software Systems V, IOS Press (1998) 247-261.