Formal Methods and Requirements Engineering: Challenges and Synergies

Sara Jones
University of Hertfordshire, Hertfordshire, UK

David Till
City University, London, UK

Ann M. Wrightson
University of Huddersfield, Huddersfield, UK

Formal methods (FM) are already making important contributions to both theory and practice in the field of requirements engineering (RE). This article presents an overview of work in this area drawing on discussions and presentations which took place at an international workshop held in December 1996. Our aim is first to outline the current demonstrated capability of FM to support RE, and then to present some of the recent developments which seem most likely to result in practical methods in the near future. Some of the significant themes identified are: the use of theorem-provers and model-checking tools, the application of formally-based requirements acquisition strategies, development of reusable software architectures which can anchor requirements elaboration, and formal treatment of inconsistency and the use of multiple notations. © 1998 Elsevier Science Inc.

1. INTRODUCTION

This paper stems from a workshop held in December 1996, organized jointly by two special interest groups of the British Computer Society: RESG (Requirements Engineering Specialist Group) and FACS (Formal Aspects of Computer Science). The aim of the workshop was to encourage communication and understanding between academics and practitioners in the fields of requirements engineering and formal methods, and to identify areas where work in each field might reinforce, complement or enlighten work in the other.

Our aim in this paper is to outline, using selected examples, the current demonstrated capability of FM to support RE, then against that background, to present the agenda for further research which emerged from the workshop. We do not pretend to provide within the scope of this article a complete coverage of the wide span of work which is under way in the general area of formal methods in requirements engineering; however, the intention when constructing the programme for the Workshop was to select a number of key topics in order to bring to the participants' attention the range of issues which arise and the efforts which are being made to address them. The best sources for further information are the proceedings of recent conferences and workshops in requirements engineering and software engineering, such as (Fickas and Finkelstein, 1993; Harrison and Zave, 1995; Ilcicmeyer and Mylopoulos, 1997; Proceedings, 1994; Proceedings, 1996; Proceedings, 1995; Garlan, 1996).

The paper falls into two main parts: firstly, we describe briefly a selection of recent successes in the application of formal techniques in requirements engineering; secondly, we present the main ques-
tions and suggestions for further research which emerged from the workshop, under the following headings:

- Change in Software and System Requirements and Specifications
- Requirements Traceability
- Requirements Elicitation and Validation
- Non-functional Requirements
- Inconsistency in Software/System Requirements and Specifications
- Use of Multiple Notations
- Architecture
- Domain Knowledge

2. SUCCESSFUL USES OF FM IN RE

The workshop which gave rise to this article took place against a background of considerable success in the use of FM both in RE and in analogous requirements-led activities over the last few years. We first outline the case for using FM in RE, with reference to a cluster of related work in one application area—high-integrity systems in aerospace—where FM has had several acknowledged successes. This is followed by brief discussions of FM in two other application areas. We close this section with an example which is partly acknowledged success and partly ongoing research, and which illustrates the subtle technical and research issues which can arise in apparently straightforward formal modelling.

2.1 The Case for FM in RE

A few years ago, John Rushby of SRI International took on the challenging task of providing the United States Federal Aviation Administration with a reasonably concise view of “strengths, weaknesses and technical issues in formal methods that should be considered in certification [of high-integrity digital systems used in aircraft]” (Rushby, 1993). (His report, especially chapters 1, 2 and the appendix, is recommended to any readers new to FM who want an accessible yet wide-ranging analysis of the nature, benefits and fallibilities of formal methods.) Interestingly from the perspective of this paper, he concludes that formal methods find their most effective application early in the lifecycle, where conventional methods are apparently weakest. There are two main aspects to this. Firstly, faults that have their origin early in the lifecycle are often both difficult to detect, and very significant in that they cause serious failures or very costly later corrections. Any technique which can locate difficult-to-detect faults (and formal methods is one such) is therefore more cost-effective in RE than later in the lifecycle. Secondly, FM can improve both the process and the content of the RE effort, which has a pervasive beneficial effect throughout the lifecycle.

Perhaps the most widely recognized of these more general benefits is that FM can provide an effective and precise means of communication between software engineers and stakeholders (Miller and Sripas, 1995). As this new mode of communication becomes more familiar, a further benefit appears: FM provide a repertoire of mental building blocks, encouraging precise yet abstract specification, and enabling reuse of difficult concepts in the domain in an already debugged precise form. These benefits have been realized in pilot projects facilitated by the NASA Langley Formal Methods Program.

2.2 The NASA Langley Formal Methods Program

The Program lies within the wider NASA Langley mission of developing techniques for the design and validation of flight critical systems. They have initiated a series of studies and pilot projects aimed at transferring formal methods technology into the US aerospace industry (Butler et al., 1996). One of these is a trial use on what is probably the biggest and most difficult planned change to the software which controls the space shuttle in flight.

The change concerns the addition of a Global Positioning System (GPS) capability for the shuttle in-flight software. The processes in the parent shuttle on-board software project are very well developed (Carnegie Mellon, 1995); it is interesting that they have turned to formal methods very late in a sustained and intensive concern with improving their processes. The reasons for this are discussed in detail in (Carnegie Mellon, 1995); briefly, they perceive formal methods as providing a worthwhile benefit only after many other measures to improve project processes, especially in relation to the flow and control of requirement information, have been thoroughly evaluated and adopted as appropriate.

The NASA report on the GPS formal methods pilot (Di Vito, 1996; Di Vito and Roberts, 1996) states:

Portions of the GPS Change Request have been modelled using the language of SRI’s Prototype Verification System. Developing the formal specifications, even with only limited analysis conducted on them, resulted in 86 requirements issues being discovered.
Experience with the GPS effort showed that the outlook for formal methods in this requirements analysis domain is quite promising. The approach detailed here produced specifications that requirements analysts and others from the Shuttle community can and did learn to read and interpret, without having to become PVS practitioners. Moreover, the mere construction of formal specifications using this method... led to the discovery of flaws in the requirements. There are good prospects for the continuation of the effort by Shuttle personnel.

From a very cautious organization, this is praise indeed.

Some of the most obviously useful applications of FM in RE have been in providing a detailed analysis of requirements which are generic (relating to a domain rather than a specific system), and also fall into the area often characterized as non-functional. Fault-tolerance, in particular, is sufficiently difficult to characterize precisely that it has given rise to lively R & D collaborations involving leading researchers in computer science. Formal methods have been an essential part of the successes in this area; the NASA Langley technical report (Butler et al., 1996) outlines how formal methods enabled three related strands of this work to be coordinated. This involved repeated use of reusable verified formal models of these difficult concepts; the formal analysis has been a crucial enabling factor in bringing theory into practice in a complex domain where high integrity is essential. (This work also uses PVS, along with the Murϕ tool developed at Stanford [Dill et al., 1992].)

2.3 GEC/SafeFM Flap Case Study

Another tricky generic property is safety. In the GEC/SafeFM Flap case study, a more heavyweight FM tool was used to automate second-order analysis of a specification developed in a formalism incapable of modelling explicitly the safety concepts involved (though of course it was capable of modelling the system functionality exemplifying those concepts).

The case study project considered a coherent fragment of a requirements specification for an Air Data Computer (the control of a flap in the context of variable wing sweep). The B toolkit (Abrial, 1996: Wordsworth, 1996) was used to develop a specification describing the functional requirements, and that specification was then analysed for various safety properties. Some of these could be stated as invariants of the B specification; some could not. The latter were analysed by translating a subset of the B specification into the PVS language. A detailed account of the work is available as a technical report from GEC-Marconi Avionics (Ormsby and Draper, 1996).

2.4 Feature Interaction

Feature interaction in telecommunications services is another area where formal specification has proved useful (see, for example, [Guerra et al., 1996]). It has contributed substantially to a clear understanding of the problem within the joint community of researchers and industry providers who are working to resolve the practical problems of unintended interactions between services. Interestingly, one conclusion mediated by formal analysis has been that feature interaction itself is easy to describe and understand: it is the system in which it is to be embedded which provides the intractable complexity. Pamela Zave and Michael Jackson (1997) discuss several ways to attack the complexity of requirements for telecommunication services.

2.5 The FAROAS and IMPRESS Projects

The work of the FAROAS and IMPRESS projects raises several questions for future research.

Funded by the UK Civil Aviation Authority, the FAROAS project (McCluskey et al., 1995) developed a formal requirements specification for the control of aircraft flying in the eastern half of the North Atlantic. The specification is a domain model capturing the concept of conflict, as defined by air traffic control rules and practice, between aircraft flying in the Shanwick Oceanic Control Area. The requirements are written in an automated subset of many-sorted first order logic, and are supporting by custom-built tools supporting diverse validation methods, including hand checking, simulation, and automated analyses. This project was successful not only in building the specification, but also in identifying and resolving a number of previously unrecognized issues regarding the concept of conflict in the air traffic control context.

The successor project, IMPRESS (McCluskey et al., 1996; West et al., 1996), is responding to concerns raised in the first project. Although it was evident that each of the validation strategies used had improved the specification, it was difficult to see how one could measure the correctness of the validated specification; furthermore, the formal analyses, as well as the informal methods, might well have succeeded for the wrong reason. They had indeed often failed, thus indicating a problem, without giving sufficiently clear diagnostics.
Within the project, improved verification and diagnostic techniques are being developed in order to provide traceable explanations and candidate fixes, as well as results from formal analyses of the specification. Current work centres on providing full analyses of the roles played by negated clauses in results obtained from the specification—a difficult problem, but probably tractable for a closed world in the logic they are using. Initial results from this work were reported at a recent conference (West et al., 1997).

For a sobering account (for logic technicians only) of the wider ramifications of negation, see Dov Gabbay’s analysis in (Gabbay, 1994) (Chapter 7, p. 177). The issues surrounding negation are just one aspect of the strong tension evident at times between the restricted (and sometimes restrictive) repertoire of logics whose use is well understood by FM practitioners, and which are supported by powerful and usable tools, and the seductive subtlety and variety of logics which can be found in the logic literature.

3. THE CHALLENGES

In this section we follow roughly the structure of the Workshop in presenting a range of issues which are the subject of ongoing research.

3.1 The Fundamental Issue

Michael Jackson began by setting the scene. His main concern was to point out the relationship between formality and informality in the requirements context. The essence of formality is reasoning with symbols without considering what the symbols stand for, using mechanical inference rules. The essence of informality is the unboundedness of the set of considerations that could turn out to be significant. This unboundedness can invalidate otherwise sound logic applied to an informal domain. Machines are formal whilst requirements are informal. To establish the requirements we must explore the problem context far from the machine; we need to know something about the intrinsic properties of the artifacts and organisations involved and we need to understand the reasons for things and the objectives to be satisfied. We formalise requirements—necessarily imperfectly—in order to understand how a formal machine behaviour at the interface affects the world. We must endeavour to minimise the approximation error by appropriate choices.

We will not go into more detail about these ideas here because there are several recently published papers by Michael Jackson and Pamela Zave discussing these issues; see for example (Jackson, 1995; Jackson and Zave, 1993; Zave and Jackson, 1997) and also the article by Michael Jackson in the present Special Issue.

We would however like to bring out one important point. A significant benefit which derives from the formalisation of properties of the real world is that we thereby provide documentation of just what is assumed about the behaviour of the real world which is relevant to the system to be built. We can only expect to demonstrate that the artefacts we build work together correctly given certain assumptions about the behaviour of autonomous (non-automated) agents in the system. It is unrealistic, in general, to attempt to show that a system will work in all circumstances; thus it is essential that we document what has been taken into consideration (for example what behaviour is expected of any human agents in the system) and by default what has not been considered.

3.2 Changes in Requirements

The requirements for a particular system or piece of software typically undergo a considerable amount of change during the course of a project. Changes to requirements may be necessary as a result of changes in the environment in which a system is to operate, or may simply arise as client, user or developer understanding of a project grows and further possibilities emerge for solving relevant problems or benefitting the client organisation in previously unforeseen ways. Some of the different causes of change are identified and discussed by Harker et al. (1993).

The question of change has particular implications for projects in which requirements are formally stated. It generally costs more to produce a precise and formal specification of system requirements than it does to develop an informal statement. This cost is felt both at an organisational level, where more financial and human resources are consumed, and at an individual level, where a requirements engineer must make a considerable intellectual investment in the development of a formal specification. Once such investment has been made, both individuals and organisations may be unwilling to consider the need for change.

The investment involved in developing a formal specification of requirements is, of course, not wasted. It can be argued that having requirements formally specified makes it easier to assess both the risks and costs involved in making particular changes. The process of formal specification itself forces many potential areas of confusion to be confronted early on: those involved in discussing requirements are
encouraged to develop and refine their ideas before requirements are specified, so that the likelihood of later requests for change as a result of developments in user or client understanding is reduced. However, the possible need for changes to requirements resulting from changes in the environment remains.

The need to keep track of changing requirements through the course of a project has been at least partially tackled by approaches to requirements traceability such as those described below. However, other work in this area has taken a slightly broader perspective. Some of this work was discussed in a panel session at a recent Requirements Engineering conference (Heitmeyer and Mylopoulos, 1997; Madhavji, 1997) where it was proposed that Lehman’s ideas on program classification and laws of software evolution might make an important contribution to solving the problem of change in the context of requirements engineering.

A logical framework for modelling and reasoning about requirements evolution has recently been proposed by Zowghi and Offen (1997). Zowghi and Offen take the view that requirements modelling can be seen as the construction of logical theories, and requirements evolution can therefore be seen as the mapping of one theory onto another. Such mappings can be formally defined and non-monotonic reasoning can then be applied to demonstrate that changes made to requirements form part of a process of rational belief revision. Such mappings can be formally defined and non-monotonic reasoning can then be applied to demonstrate that changes made to requirements form part of a process of rational belief revision. Zowghi and Offen are currently investigating the applicability of their approach to real problems of requirements volatility in industrial software development projects, and are building a prototype toolset which they hope to validate in realistic industrial contexts. Developments such as these, and ones discussed in the following section on traceability may eventually both reduce resistance to changing formal specifications and increase the reliability of change procedures.

3.3 Requirements Traceability

Work on requirements traceability can be seen, at least in part, as a response to the need for keeping track of changing requirements as discussed in the previous section. However, other concerns are considered here too. An overview of relevant issues is provided by Gotel and Finkelstein (1993) who define requirements traceability as “the ability to describe and follow the life of a requirement, in both a forwards and backwards direction,” or in other words, to see both the impact and origins of any requirement in a particular project. Traceability has been recognised as an important issue by organisations such as Nortel (formerly Bell-Northern Research Ltd.—see Macfarlane and Reilly [1995]) and the US DoD’s Weapon System Technology Support Branch (Stubbs et al., 1995).

Some support for tackling the problem of requirements traceability can be provided by the formal methods community in the shape of formally defined traceability tools. One such tool, TOOR, is described by Pinheiro and Goguen (1996). TOOR adopts an object-oriented view of the traceability problem, permitting all artefacts of the development process, formal or informal, to be traced through the course of a project. The TOOR system provides facilities similar to those of a database, but is based on the formally defined FOOPS language (Goguen and Messeguer, 1987). A number of case studies are currently being developed which aim to demonstrate how real-world problems may be handled effectively using TOOR.

Laurence James of Marconi Systems (the developers of RTM, a requirements traceability and management tool), has identified what he calls the “four principles of requirements traceability.” These are:

- closure: the need to make connections between requirements all the way through to delivered systems
- continuity: consistency of information flow between different project viewpoints
- logic: the need to ensure that solutions developed are correct
- emotion: the idea that traceability is emotion backed up by logic, or in other words, that requirements engineers want not just to know that requirements can in principle be reliably traced, but also to feel confident that this is really possible in practice

Traceability tools can make the necessary links between informal requirements and formal specifications of components and their properties: formal languages can be used to specify precisely and unambiguously what is needed, while traces help to answer questions about why such precisely described components are needed and how they relate to the informally expressed requirements. Formally defined tools such as TOOR can thus make an important contribution in the first three of the areas identified above, though supporting the fourth may require something more. We will only address the fourth point—emotion, or confidence—when the detailed ergonomics and the reliability of the tools reaches a high level and when there is sufficient experience of
successful use of the tools. This is beginning to happen.

3.4 Requirements Elicitation and Validation
The requirements elicitation process involves a gradual increase in the precision and/or formality of requirements. An important attempt to support this process of moving from initially vague or abstract ideas about required functionality towards a precise formal specification has been made in the KAOS project (van Lamsweerde et al., 1993; Massonet and van Lamsweerde, 1997) as described in the section below on Domain Knowledge. In a related development arising from the ICARUS project, DuBois et al. (1997) have developed the Albert II language which, they claim, allows a natural mapping of customers' informal needs onto formal statements in the domain of real-time distributed systems. The KAOS methodology has already been applied in two industrial projects (Massonet and van Lamsweerde, 1997), and tool support for the Albert II language is under development.

The need for validation and verification of requirements and specifications is familiar to practitioners both in formal methods and in requirements engineering. A discussion of whether formal specifications are, or should be, executable has been underway for some time (see, for example, Hayes and Jones (1989), Fuchs (1992) and Gravel and Henderson (1996)). Animation of formal specifications of requirements has been seen as a way of harnessing the benefits of both formal methods and prototyping approaches to the development of interactive systems. A number of authors have considered the animation of specifications written in Prolog, some implementations of which provide for easy development of graphical interfaces and simulations (see, for example, Kramer and Ng (1988), Johnson and Harrison (1990) and McCluskey et al. (1995); also see section 2.3 of the present article) Nobe and Warner (1996) also describe the way in which statechart modelling was used in developing and validating specifications of system behaviour in the Boeing Commercial Airplane Group. The use of statecharts is supported by a number of commercially available tools such as Statemate by i-Logix, Inc. and Better-State by R-Active Concepts, Inc., and Nobe and Warner state that:

the most powerful experience for the users—[who were specialists in areas such as air conditioning, electrical power generation and flight controls, rather than software engineering]—came from their ability to see the effect of their models via the tool's dynamic simulation and interactive animation features (Nobe and Warner, page 86, 1996).

In a slightly different vein, Easterbrook and Callahan have investigated the role of lightweight formal methods in uncovering major errors in incomplete specifications of large safety-critical systems (Easterbrook and Callahan, 1997). They conclude that such methods show significant promise in that context, though they identify a number of problems in applying formal methods in this way. A further note of caution is sounded by Loomes and Vinter (1997) who have demonstrated that software engineers are prone to making a range of errors when reasoning about Z specifications.

While validation of requirements specifications—formal and otherwise—is still so difficult, no single method is likely to be sufficient on its own, particularly in the development of safety-critical applications where reliable validation procedures are crucial. Solutions considered in the FARAOS and IMPRESS projects were to use a range of different validation methods and to apply artificial intelligence techniques to automate some formal analysis of specifications, as described in Section 2.

3.5 Non-Functional Requirements
There is some debate as to how the term “non-functional requirements” can be defined. Various other terms including “nonbehavioural requirements” and “quality requirements” have also been used to denote a similar category of requirements which we may describe roughly as “requirements that define the overall qualities or attributes to be exhibited by the resulting software system” (Davis, 1993).

David Robertson suggested at our workshop that the use of formal methods in dealing with non-functional requirements is still at a pre-engineering stage, so that examples of good practice are scarce, and reliable guidelines for their use in this context are nonexistent. The view taken by Ian Sommerville at the workshop was that non-functional requirements cannot satisfactorily be identified as a kind of requirement fundamentally different from functional requirements. In fact, he claimed, it is the mistaken assumption that such a distinction can be made that has been largely responsible for the poor uptake of academic formal methods in industry: formal methods research has focused almost exclusively on functional requirements, while industry has had to be
very concerned with so-called non-functional aspects of systems such as performance, reliability and usability.

Some work on developing notations with formal semantics which are appropriate for representing non-functional requirements is underway: for example, Ben-Abdallah et al. (1997) discuss the formal specification of temporal and resource requirements for real-time systems. This work is still at a relatively early stage. Other interesting work in this area is that of Suzanne Robertson of the Atlantic Systems Guild on making requirements measurable. Her approach to non-functional requirements is that we must make any such requirements sufficiently specific as to be measurable, and that the analyst must continue to question the stakeholders until appropriate metrics have been established. So, for example, it is not sufficient to ask for a system which is “pleasant to use”: how much training will be needed? how long does it take a user to find relevant on-line help information? by how much will the users’ productivity increase? how often will they choose to use the system rather than doing the task manually? If we can obtain answers to questions such as these at least they can be expressed precisely and even formally, though we may not usually be able to prove by logical inference that the delivered system meets such requirements. The validation and verification of such requirements will often involve field trials of prototypes, but this is of course a normal part of engineering. In other cases, where the non-functional requirements relate, for example, to automated sub-system performance, if we make them measurable we may indeed be able to demonstrate that the system will, under predicted load profiles, meet the specified performance criteria.

3.6 Inconsistency

We first need to consider what we might mean by inconsistency, because it turns out that it can mean a range of different things. If we are using standard notations, or diagram-based systems—for example because we are using an object-oriented analysis method—there may be inconsistency in the sense that the well-formedness rules for the notation are not satisfied. This is of course easily checked mechanically, assuming the rules are computationally effective. On the other hand, if we try to compose by conjunction two collections of logical formulae, then the conjunction may be unsatisfiable, and we have logical inconsistency. However it is important to realise that we will often fail to detect such logical inconsistency, not because there is no conflict but because our formal descriptions do not take into account enough significant properties of the domain to reveal the conflicts. Another possibility is that inconsistency will be detected as a result of inconsistent usage of terminology and vocabulary by different stakeholders, when in fact there is no real underlying conflict. Thus a more general term has been suggested, namely interference, to mean that two collections of information refer to some of the same concepts and issues, with the possibility that there could be latent conflict and there could be confusion of terminology.

There is currently considerable research effort directed towards the clarification of the important issues here, and towards providing support for the management of interference, including the detection and resolution of the conflicts involved. For example, see (Cugola et al., 1996; Finkelstein et al., 1996). There is also to be a Special Issue of the IEEE Transactions on Software Engineering on the management of inconsistency, scheduled to appear in Spring 1998.

At the Workshop, Alfonso Fuggetta pointed out that though we must ultimately have a consistent and complete specification of the behaviour of a system which will meet agreed and consistent objectives, the process of reaching such a happy state of affairs will inevitably pass through intermediate states in which our knowledge is partial and even inconsistent. This is not only inevitable but indeed necessary: the existence and detection of inconsistency is one of the major driving forces in the discovery of further salient information about the domain and the problem to be solved within that domain, and in the identification of points where compromises have to be found perhaps reducing the expectations of certain stakeholders.

Tony Hunter gave examples of how (logical) inconsistency can be a spur to action; in each case, it turned out that upon further investigation there were unstated assumptions. In effect, the inconsistencies arose in exceptional circumstances, and the unstated assumptions were that the general rules asserted did not apply under those conditions. He also described how we can reason in the presence of inconsistency by using quasi-classical logic (Hunter and Nuseibeh, 1997). Again, he stressed that we do not necessarily want to eliminate inconsistency; rather, we want to manage it. In particular, we may want to investigate the sources of inconsistency, we may want to ignore it or delay its resolution. If we do in the end resolve it, it is important to keep a
trace of it, since this will remind us of what conflicts had to be addressed, and indeed who was involved in the decisions which led to its resolution.

3.7 Multiple Notations

It is generally accepted that it can be helpful to try to understand the views of different stakeholders in isolation before we try to construct a specification which takes account of their different views but which resolves in generally acceptable ways the inconsistencies and conflicts of interest which emerge. It is natural that different stakeholders will use (perhaps with the help of a requirements engineer) different languages and notations to express their views, and in many cases these notations will be formally based. At the Workshop, Tom Maibaum expressed the challenge which arises here in the following way:

How can we relate descriptions in different formalisms with possibly different structuring principles and almost certainly different structures?

Jose Fiadeiro recast the problem:

How can we relate and integrate different notations so that we can understand the whole, detect interferences and support incremental development and evolution?

Some authors, see for example (Zave and Jackson, 1993), have explored the idea of translating notations into a common style of predicate logic and then integrating on the basis of a common semantic domain. Fiadeiro and Maibaum (1995) have proposed as an alternative the use of category theory as a convenient mathematical framework which allows the representation of explicit linkages of entities in different views and translation between structuring principles. So, rather than using a common semantic domain, the different notations are formalised within a common mathematical framework, namely category theory, and the relationships are established by functors.

Clearly this is interesting and important work addressing a central problem in RE; we believe it is fair to say that currently this work is going on at the level of theoretical underpinning and it will take some time to reach a point where practical tools and methods can be provided for the working requirements engineer.

3.8 Software Architecture

Addressing the Software Architecture theme, Jeff Kramer suggested that software architecture is in the solution domain, while requirements are in the problem domain, though requirements themselves often do describe an architecture of sorts, perhaps best represented by Michael Jackson’s problem frames (Jackson, 1995). He also suggested that it can be very difficult to state or to understand requirements clearly until some feasible architecture has been contemplated. Thus, in many cases, the cycle of activities most likely to produce better understanding of how high-level system goals may be operationalised is to push forward towards design of a feasible implementation, based on well-tried software architectures, and then to step back again, having achieved greater insight into what is possible and the kind of behavioural characteristics to be expected from such an implementation. Of course, this assumes that for the software architectures concerned we can infer certain important behavioural characteristics—for example, performance measures—without knowing the details of the plug-in, application specific modules which will be needed.

The software architecture field is a very active research area at the moment. As we have said, it can be argued that its relevance is greater in the design phases of system development, but the interested reader is referred to the proceedings of recent international research meetings (Garlan and Perry, 1995; Garlan, 1996; Vidal and Wolf, 1996) and also to the following further publications (Garlan and Shaw, 1993; Shaw and Garlan, 1996).

3.9 Domain Knowledge

Once again, the idea of domain knowledge is open to different interpretations, although its importance in the field of requirements engineering has long been acknowledged (Bolton et al., 1992). For Michael Jackson, page 9 (1995), the term “application domain” means:

... something entirely specific to the problem at hand; not a generic domain, denoting a class of applications, as in the phrases the banking domain, or the telephone switching domain, or the command and control domain, but rather this particular company's employees, this particular aeroplane's control surfaces, these particular suppliers and products and orders.

According to this definition, a proper understanding and explicit description of the application domain is an essential precursor to system development, and facilitates early identification of reusable requirements patterns or “problem frames”.

Other authors considering the notion of generic domain knowledge have discussed the extent to which formal representations of such knowledge can assist the developer both in reasoning about new
developments and in reusing knowledge or requirements gathered in previous projects. For example Holland et al. (1994) describe how a general mathematical theory of the operation of clinical directorates was used to reason about the requirements for a new system, and to help detect inaccuracies in the analysis phase of a project to develop a clinical information system. Work on providing support for requirements reuse through the more or less formal specification of domain knowledge has been carried out by the NATURE (Jarke et al., 1993) and KAOS (Massonet and van Lamsweerde, 1997) projects.

We now look at the KAOS project in some detail because we feel that it makes a significant contribution to the synthesis of FM and RE by attempting to provide a smooth transition from the informal requirements gathering phase through to the formal specification of system components. KAOS proposes the use of a meta-model, embracing meta-concepts such as Goal, Object, Agent, Constraint and Relationship, together with meta-relations such as Agents-are-Capable-of-Actions, Goals-Conflict-with-Goals, Goals-Concern-Objects, and so on. There are also meta-attributes and meta-constraints (for example, an Agent-Performs-an-Action only if that Agent-is-Capable-of-that-Action). The intention is that the meta-model should provide a set of interrelated concepts which may be used to describe the requirements for a large spectrum of systems, and that these meta-concepts and their meta-relations should be comprehensible and acceptable to stakeholders. A requirements model is constructed as an instance of the meta-model: so there will be domain-specific instances of meta-concepts, related by domain-specific instances of meta-relations, and so on.

Two levels of formalisation are involved in KAOS requirements models; an outer level corresponding to the meta-level entity-relationship structure and an inner level whereby at later stages appropriate formal assertions—for example, Pre-Condition for Actions, Invariants for Objects—can be attached to instances of meta-concepts. In the early stages of constructing the requirements model, the focus is on the high-level Goals and their inter-relationships; so AND/OR Goal-reduction trees are constructed, and conflicts between Goals as well as other important Objects, Agents and Actions are identified. Thus, very early on there is a formal model which relates to the objectives of the stakeholders and can provide a starting point for a process of progressive formalisation. The reader is referred to (van Lamsweerde et al., 1993) for a more comprehensive presentation of these ideas.

4. SUMMING UP

In conclusion we highlight the work which seems most likely to promote the productive synergy of FM and RE, and give some indications of where there are methods which can be put into practice in the shorter rather than the longer term. We shall not add anything here to our earlier review of recent successful applications of FM in RE in Section 2.

Firstly then, the most significant strands are, we feel:

- the use of theorem-provers and model-checkers, especially where the requirements are for control systems for safety-critical applications
- the work on software architecture which promises to provide reusable frameworks, with known properties, for distributed systems
- the work on inconsistency and use of multiple notations
- where the requirements are intimately bound up with the high-level objectives of an enterprise, goal-directed acquisition strategies such as that of the KAOS project seem very promising

Secondly, for shorter term applicability, with tool support:

- there are a number of well-engineered immediately usable theorem-proving and model-checking systems, such as PVS (Owre et al., 1993a, 1993b, 1993c), B (Abrial, 1996; Wordsworth, 1996) and the SCR tools (Heitmeyer et al., 1995)
- tools are currently being constructed for the KAOS (van Lamsweerde et al., 1993; Darimont et al., 1997) approach; many larger-scale case studies have been undertaken and literature will be produced over the next year or so intended to make the method accessible to the RE community at large
- the software architecture work is under very active investigation by various research groups and there are experimental systems available

There is still a great deal of resistance to the use of formal methods at all stages of the systems development lifecycle. However, we feel that in the not too remote future it will more generally be seen as an indispensible part of the training of professional systems engineers to be able to use appropriate mathematical techniques and their supporting tools in order to model the behaviour of systems to be built and to prove that they have the properties required of them. We have tried to show here the range of work which is currently underway which could form the basis for such techniques and tools.
REFERENCES


Darimont, R., Delor, E., Massonct, P., and van Laarhoven, P., GRAIL/KAOS: an environment for goal-directed requirements analysis integration and layout. In [Heitmeyer and Mylopoulos, 1997].


Hunter, Anthony, and Nuseibeh, Bashar, Analysing inconsistent specifications. In [Heitmeyer and Mylopoulos, 1997].


Jackson, Michael, Problems and requirements. In [Harri-son and Zave, 1995].


Macfarlane, I. A., and Reilly, I., Requirements traceability in an integrated project environment. In [Harrison and Zave, 1995].

Madhavji, N. H., Impact of environmental evolution on requirements change. In [Heitmeyer and Mylopoulos, 1997].

Massonet, P., and van Lamsweerde, A., Analogical reuse of requirements frameworks. In [Heitmeyer and Mylopoulos, 1997].


Stubbs, C., Ramesh, B., Powers, T., and Edwards, M., Implementing requirements traceability: A case study. In [Harrison and Zave, 1995].


Zave, Pamela, and Jackson, Michael, Requirements for telecommunication services: an attack on complexity. In [Heitmeyer and Mylopoulos, 1997].


Zowghi, D., and Offen, R. A logical framework for modeling and reasoning about the evolution of requirements. In [Heitmeyer and Mylopoulos, 1997].