Requirements Engineering Through Viewpoints

Anthony Finkelstein, Steve Easterbrook[1], Jeff Kramer & Bashar Nuseibeh
Imperial College
Department of Computing
180 Queen's Gate, London SW7 2BZ
acwf@doc.ic.ac.uk

0 Abstract

This paper provides a short review of contributions to a better understanding of requirements engineering arising from research at Imperial College. These contributions share a common theme - a focus on "multiple perspectives" or viewpoints.

1 Theme

The development of most large and complex systems necessarily involves many people - each with their own perspective on the system defined by their skills, responsibilities, knowledge and expertise. This is particularly true where the system is a composite system, that is one which deploys a variety of different technologies (software, hardware, mechanical and so on). Inevitably, the different perspectives of those involved in the process intersect and overlap, giving rise to a requirement for coordination. The intersections are, however, far from obvious because the knowledge within each perspective is represented in different ways. Further, because development may be carried out concurrently by those involved, different perspectives may be at different stages of elaboration and may each be subject to different development strategies.

The problem of how to guide and organise development in this setting - many actors, sundry representation schemes, diverse domain knowledge, differing development strategies - we term "the multiple perspective problem". The multiple perspective problem is central to both requirements expression and elicitation.

System specification from multiple perspectives using many different specification languages has become an area of considerable interest. The integration of methods, notations and tools has generally been addressed by the use of a common data model, usually supported by a common, centralised database (Wasserman & Pircher 1987, Alderson 1991). Recent work by Zave & Jackson (1992) proposes the composition of partial specifications as a conjunction of their assertions in a form of classical logic. A set of partial specifications is then consistent if and only if the conjunction of their assertions is satisfiable.

Other authors have also considered multi-perspective or multi-language specifications, in Wileden et al. (1991) specification level interoperability between specifications or programs written in different languages or running on different kinds of processors is described. The interoperability described relies on remote procedure calls and ways that interoperating programs manipulate shared typed data. Wile (1991) on the other hand uses a common syntactic framework defined in terms of grammars and transformations between these grammars. He highlights the difficulties of consistency checking in a multi-language framework.

[1] Steve Easterbrook is now at the School of Cognitive and Computing Sciences, University of Sussex

Traditionally, multiparadigm languages, which deploy a common multiparadigm base language, have been used to combine many partial program fragments (Hailpern 1986), while more recently the use of a single, common canonical representation for integrating so-called “multi-view” systems has been proposed (Meyers & Reiss 1991).

Approaches based on facilitating negotiation and cooperation between participants in the development process are another angle of attack on the multiple perspectives problem, these can be combined with more conventional techniques for determining inconsistencies.

We have sought to address the multiple perspective problem in a number of research projects aimed more generally at improving the state-of-the-art in requirements engineering.

These projects have:

> developed formal representation schemes for requirements specification and built development methods targeted on those schemes;
>
> extended tool support for CORE, an established requirements engineering method;
>
> outlined a formal model of the process of specification from multiple perspectives based on an account of how “commitments” are established during requirements elicitation;
>
> provided a scheme for organising the process of review and correction of complex requirements specifications;
>
> developed a framework and environment which supports the use of heterogeneous representation schemes and partial views of complex domains;
>
> developed a method for using collaborative exploration of multiple perspectives as a means of resolving conflicts between views.

Below we briefly outline each project, give an account of its particular contribution to addressing the multiple perspective problem and relate it to the other projects.

2 FOREST

2.1 Outline

The FOREST (Formal Requirements Specification Techniques) project has addressed the problems of specifying the requirements for large scale real time embedded systems. In particular it has developed a formal specification language, Modal Action Logic (M[A]L), and a method, Structured Common Sense (SCS), to guide and organise the process by which a M[A]L specification is constructed. Other contributions of the FOREST project in the area of automated reasoning and tool support have been described in the literature. For a description of the project reference should be made in the first instance to Goldsack & Finkelstein (1991).

M[A]L is based on the familiar (many sorted) first order predicate logic, extended by a series of layers, which are needed in order to describe the services provided by, and the constraints on the behaviour of, a reactive system. The layers are: agent action modalities; deontic formulae; action combinators; temporal interval logic.

SCS supports the construction of a formal requirements specification in M[A]L, guiding the processes of elicitation and formalisation. It identifies the basic concepts underlying M[A]L which must be elicited and the best order in which to elicit them. Techniques from a variety of existing methods and some specially developed techniques targeted on M[A]L have been assembled into an integrated method. The structure of SCS is similar to that of conventional development methods. It consists of a number of distinct steps some of which are performed in parallel and some sequentially. Progress through SCS is driven by a work plan. Each step has associated with it intermediate graphic representations and heuristics.

FOREST was funded by the DTI under the Alvey Initiative and was a collaboration between Imperial College, GEC & AEA Technology.

2.2 Multiple Perspectives

FOREST did not directly address the multiple perspective problem but had two features relevant to it. M[A]L is based on agents which perform actions. Using M[A]L for requirements specification led us to consider how we would go about identifying these agents. Simply considering these as functional blocks in some envisaged system was inappropriate - it leads to premature design decisions. We therefore chose to interpret M[A]L agents as “knowledge sources” in elicitation. This interpretation is very similar to that adopted by the requirements expression method CORE, also referred to below, and leads to requirements specifications for which traceability back to originating statements and elicitation records are clear.

In building SCS we treated each intermediate representation as a “view” in the partial specification sense of that term. Each representation was designed to elicit one of the elements of M[A]L. Thus, for example, we used a data-flow table to help identify the actions performed by agents. The transforms in the data-flow table were mapped to actions in a “rough-cut” M[A]L specification and the data-flows themselves discarded. In certain cases the extra information was held as a prompt to further steps in the method.

The work on FOREST lead us to examine how requirements specifications ought to be structured and how methods should be constructed.

3 TARA

3.1 Outline

The particular objective of the TARA (Tool Assisted Requirements Analysis) project was to examine three important extensions to current CASE technology in the area of requirements analysis. We were interested in the role of automatically provided method guidance to support the use of requirements analysis methods, the ability to use software tools to help clients and analysts visualize the behaviour of the specified system by animation of the specification, and the possibility of supporting the reuse of specification fragments or parts of existing specifications in the composition of a new specification. The main application focus of our work was the large class of systems which can be classed as "real-time information systems"; that is systems which must satisfy temporal constraints and are also data rich.

The intention was not to construct another diagram editor and requirements specification technique. Hence we adopted as a base for this work an existing, widely used, requirements analysis method - CORE (Systems Designers 1986) - and a CASE tool for diagram construction and consistency checking - The Analyst (Stephens & Whitehead 1985).

The results of this work are detailed in Finkelstein & Kramer (1992). An earlier paper, Kramer et al. (1988), provides an incomplete and preliminary view of the project. These results include: a method guidance system for CORE integrated with The Analyst, giving normative and remedial advice to method users; tool support for animation of CORE transactions, The Animator, with full graphics support for the generation and manipulation of transaction diagrams, Kramer & Ng (1988); a prototype tool - TRUE - to support Transaction Reuse based on a model of reuse derived from artificial intelligence research on analogy, Finkelstein (1987).

TARA was funded by the USAF Rome Air Development Centre and was a collaboration between Imperial College and SD-Scicon.

3.2 Multiple Perspectives

The insights and experience derived from working on the TARA project have been considerable and has directly influenced our work on multiple perspectives. This is a result both of the specific contributions of the work and of the increased respect we have developed for CORE as a method. In particular it has lead us to favour support for software development by methods consisting of many, relatively simple, representations tightly coupled to each other by large numbers of consistency checks. In this setting an explicit and enactable work plan provides a means for both managing the enforcement of the consistency checks and managing the consequences of redundancy.

Given a method with a work plan and with a rich collection of heuristics the method advice must be delivered to the point at which the work - the construction of the specification - is actually being carried out. The granularity of this method advice must be appropriate to the tasks being performed. It is worth noting that we modelled CORE using M[A]L.

TARA also gave us a much better understanding of how people work with software development methods. In particular we have come to realise that work is often left incomplete and inconsistent, that users move rapidly between different representations changing their minds frequently, and that analysis and validation are tightly interleaved with the construction of the specification.

Support for reuse needs to be engineered into the representation schemes underlying a method from the start. Without such support taking advantage of existing specifications will always be difficult. The most sensible strategy in this setting is to provide a variety of powerful means of viewing and understanding such specifications.

As mentioned, TARA provided us with considerable experience of and respect for CORE and for CORE viewpoints as a means of domain decomposition. The CORE viewpoint, essentially an agent or role, combines a domain structure with the distribution of authority for making decisions about the specification. As such it provides a powerful means of structuring requirements specification and organising requirements elicitation. However, CORE viewpoints are required to be orthogonal, and the way in which information within a viewpoint is provided is restricted. These limitations severely constrain the extent to which CORE or similar methods support multiple perspectives.

4 IC-DC

4.1 Outline

The objective of the IC-DC project was to develop a formal understanding of specification from multiple perspectives in order to both support the construction of formal specifications and reason about the process of specification itself. To do so, it took what might be broadly termed an AI approach - modelling the mechanisms which underlie the way people carry out the complex task of specification.

We developed a model (IC-DC) based on how "commitments" are negotiated and established as a specification is constructed. The model, which develops concepts taken from dialogue and commitment logics, is described in full in Finkelstein & Fuks (1989).

In this model a specification consists of a set of statements. Each statement is an effective restriction on the freedom of action on the part of the developer who must build a system that satisfies the specification. By making a statement the specifier is, in effect, making a "commitment" that is, holding him or herself out as liable for the consequences of that statement. In this context specification is seen as a process by which commitments are negotiated and established by the parties having responsibility for the description and development of the system. This process takes the form of a dialogue or organised sequence of locutions.

Dialogues have an "etiquette" which governs the legitimate shape of the interaction, performing a locution will generally have an effect on the commitments of the participants, these can be defined in dialogue and commitment rules. The form of reasoning permissible within the dialogue can be defined syntactically.

We developed tools which animate, albeit in a simple minded way, the dialogue scheme. These tools allow the user to develop simple dialogues and then replay them in whole or in part.

4.2 Multiple Perspectives

The IC-DC work directly contributed to research on multiple perspectives. It examined in some detail the idea of independent agents (loosely coupled, locally managed objects) holding a perspective and interacting with other independent agents. This work mapped out the links between work on multiple perspectives in requirements engineering and the related fields of computer-supported cooperative work and distributed artificial intelligence. In particular the work established the importance of the high-level protocol that allow agents to interact. The work demonstrated the contribution that logic might make in this area.

5 Fixing Specifications

5.1 Outline

The Fixing Specifications project used the IC-DC model to underpin support for a practical problem in the review of large and complex specifications - marking errors and corrections. A simple graphical notation, similar in spirit to typographical correction marks, was developed. Errors and corrections could be marked on the specification text using this notation and the IC-DC model was used to define how these marks should be removed and replaced by the corrected or changed specification. This scheme provided a categorisation of common types of edits and queries and thus gave a simple vehicle for communication between reviewer and author which addresses corrections and annotations to specifications at the appropriate level and is (notationally) consistent and well (formally) defined. A full account is given in Finkelstein (1992).

5.2 Multiple Perspectives

The work on Fixing Specifications provided us with a better understanding of how a model such as IC-DC could be applied. We were able to develop a method of formally annotating specifications which was independent of the underlying representation scheme, and hence can be used in documents consisting of heterogeneous representations, and which marked errors and queries explicitly and by identifying potential conflicts support the resolution of disagreements.

6 ViewPoints

6.1 Outline

The ViewPoints framework directly addresses the use of multiple perspectives in system development. This work has been primarily, though not exclusively, carried out under the aegis of the Software Engineering and Engineering Design (SEED) project in conjunction with The City University. A good account of the work on ViewPoints is given in Finkelstein et al. 1992.

The primary building blocks of the framework are “ViewPoints”. A ViewPoint can be thought of as a combination of the idea of a “actor”, “knowledge source”, “role” or “agent” in the development process and the idea of a “view” or “perspective” which an actor maintains. In software terms it is a loosely coupled, locally managed, coarse-grained object which encapsulates partial knowledge about the system and domain, specified in a particular, suitable representation scheme, and partial knowledge of the process of development.

Each ViewPoint is composed of the following components, which we call slots:

> a representation style, the scheme and notation by which the ViewPoint expresses what it can see;
>
> a domain, which defines that part of the “world” delineated in the style;
>
> a specification, the statements expressed in the ViewPoint’s style describing particular domains;
>
> a work plan, describing the process by which the specification can be built;
>
> a work record, an account of the history and current state of the development.

The development participant associated with any particular ViewPoint is known as the ViewPoint “owner”. The owner is responsible for developing a ViewPoint specification using the notation defined in the style slot, following the strategy defined by the work plan, for a particular problem domain. A development history is maintained in the work record.

Many ViewPoints may employ the same development technique to produce different specifications for different domains. We have therefore defined a reusable ViewPoint Template in which only the style and work plan slots are elaborated. A single ViewPoint template may then be instantiated more than once to yield different ViewPoints.

In general, a method is composed of a number of different development techniques. Each technique has its own notation and rules about when and how to use that notation. Thus, in the context of the ViewPoints framework, a method is a configuration (structured collection) of ViewPoint templates, the templates corresponding to the method’s constituent development techniques. A further phase of the FOREST project has used the ViewPoint framework to build a version of SCS targeted on a substantially modified version of M[A]L which includes an object-oriented structuring scheme.

A development is viewed as a configuration of ViewPoints instantiated from a method's ViewPoint templates.

Integration is achieved by checks maintained locally within each ViewPoint and enforced, where it is required, by a model of the process of development. These checks define partial consistency relations between the different representation schemes. Consistency is checked incrementally between viewpoints at particular stages rather than being enforced as a matter of course. Checks may be used to determine whether ViewPoints are consistent with each other and as transformations to move information between viewpoints.

A prototype computer-based environment, called The Viewer, has been constructed to support the framework, and several sample tools supporting individual ViewPoint templates have been integrated into this environment (Nuseibeh & Finkelstein 1992).

6.2 Multiple Perspectives

The ViewPoint framework is the current focus of our work on multiple perspectives. It combines ideas from all of the projects summarised above and has particular implications for method and tool integration.

We are broadly satisfied that the work to date gives a coherent approach to the management of multiple perspectives. It has enabled us to see more clearly the major technical problems that must be tackled in order to provide appropriate automated support for deploying multiple perspectives. Our attention is now focused on two closely related issues which we view as of particular importance:

how the relations between different representation schemes should be expressed;

the mechanisms by which these relations are used as checks or transformations.

We have been able to distinguish and characterise these issues and have explored some alternatives particularly in the development of a model of inter-ViewPoint communication.

An area which we regard as important and which we hope to be able to consider in more detail is conflict resolution. Current progress in this area is described below.

7 Computer-Supported Negotiation

7.1 Outline

The Computer Supported Negotiation (CSN) model takes the ViewPoints framework as a basis, and defines a process for resolving conflicts between ViewPoints. The project was carried out as a PhD thesis, and is described fully in Easterbrook (1991). A shorter account of the conflict resolution model can be found in Easterbrook (1993).

CSN assumes during the elicitation process, it is convenient to represent separately contributions from different sources, ignoring conflicts between them. This offers two advantages: individual contributions remain available for purposes of tracing the origin of particular requirements; and it reduces the possibility that previously elicited, conflicting perspectives might restrict and interfere with the elicitation process.

As a collection of disparate ViewPoints is gathered, CSN offers a tool-supported approach to resolution of conflicts between them. No attempt has been made to automate the resolution process, as in most cases a resolution requires new information, in addition to that already represented in the ViewPoints. In fact, the process promises to be a cost-effective technique for eliciting the hidden assumptions and contextual information associated with a ViewPoint: by concentrating on the reasons underlying conflicts between ViewPoints, it prompts for most important pieces of missing information.

Essentially, the approach combines methods drawn from organisational psychology with cognitive modelling techniques, to produce a three-phase exploration process. The three phases are exploration, generation and evaluation. In the exploration phase, the conflicting ViewPoint specifications are displayed side-by-side, and their owners identify correspondences between elements or groups of elements. Areas where there is no correspondence, or only an approximate correspondence, are noted – these comprise the components of the conflict. The generative phase elicits suggestions for integrating parts of the conflicting ViewPoints, by considering the different ways in which they might be combined, and the extent to which they interfere. The evaluative phase considers how these suggestions might be combined, and the extent to which each of them addresses the original conflict.

7.2 Multiple Perspectives

This project addressed a crucial aspect of multiple perspectives: how to deal with incompatibilities. Earlier approaches, based on consistency checking, tackled the types of conflict that occur during incremental refinement of a single model. However, such approaches were unsatisfactory for a proper treatment of multiple perspectives. In this project, we concentrated on the types of conflict that were not amenable to automated resolution, typically where two perspectives have evolved independently and have no common basis.

In many cases, no single resolution is possible, as the ViewPoints represent orthogonal views. In such cases, the approach offers a means of exploring and mapping the conflict. The most important outcome, therefore, is often not the resolution, but the understanding of the conflict arrived at during the exploration process.

Note that the model does not address the detection of conflict, nor the connection between inconsistencies which are detectable at the syntactic level, and conflicts. It is likely that inconsistencies will often reveal the existence of conflicts, but this is certainly not always the case. In software design, some requirements conflicts remain undetected until after the software is implemented. Further work is needed on the detection of conflicts in the ViewPoints framework, beyond the syntactic approaches currently available.

8 Summary

This paper has briefly reviewed contributions to a better understanding of requirements engineering arising from research at Imperial College. These contributions share a common theme - a focus on “multiple perspectives”. Copies of reports and papers can be obtained from our ftp archive at doc.ic.ac.uk (directory: papers).

Acknowledgements

We would like to thank our colleagues and students some of whose work is reported above.

References

Alderson, A. (1991); Meta-CASE technology; European Symposium on Software Development Environments and CASE Technology, Königswinter, June 1991, LNCS 509 (Endres and Weber eds.), Springer-Verlag, pp 81-91.

Easterbrook, S. M. (1991) Elicitation of Requirements from Multiple Perspectives; PhD Thesis; University of London.

Easterbrook, S. M. (1993) Resolving Requirements Conflicts with Computer-Supported Negotiation; [To appear] Social and Technological Issues in Requirements Engineering; Bickerton, M. & Jirotka, M. (eds.); Academic Press.

Finkelstein, A. (1987); Reuse of Formatted Specifications; IEE Software Engineering Journal; 3, 5, pp186-197.

Finkelstein, A. & Fuks H. (1989); Multi-Party Specification; Proc 5th International Workshop on Software Specification & Design, pp 185-195; IEEE CS Press (also as Special Issue of ACM Software Engineering Notes).

Finkelstein, A. & Kramer, J. (1992); TARA: Tool Assisted Requirements Analysis; Loucopoulos P. & Zicari R; Conceptual Modelling, Databases & CASE: an integrated view of information systems development; Wiley.

Finkelstein, A. (1992); Reviewing and Correcting Specifications; [To Appear] Computers & Writing: issues and implementations ; Sharples, M. (Ed.); Kluwer.

Finkelstein, A.; Kramer, J.; Nuseibeh, B.; Finkelstein, L. & Goedicke, M. (1992); Viewpoints: a framework for integrating multiple perspectives in system development; International Journal of Software Engineering and Knowledge Engineering, 2, 1, pp31-58

Goldsack, S. & Finkelstein, A. (1991); Requirements Engineering for Real-Time Systems; Software Engineering Journal; 6, 3, pp101-105.

Hailpern B (ed.) (1986); Special issue on multiparadigm languages and environments; IEEE Software, 3(1), Special issue on multiparadigm languages and environments, 10-77, January 1986.

Kramer J. & Ng K. (1988); Animation of Requirements Specifications; Software - Practice and Experience; 18, 8, pp749-774.

Kramer J., Ng K., Potts C. & Whitehead, K. (1988); Tool support for Requirements Analysis; Software Engineering Journal; 3, 3, pp86-96.

Meyers S & Reiss S P (1991),A System for Multiparadigm Development of Software Systems", In Proceedings of Sixth International Workshop on Software Specification and Design, Como, Italy, 202-209, 25-26th October 1991.

Nuseibeh, B. & Finkelstein, A. (1992); ViewPoints: a vehicle for method and tool integration; Proceedings of International Workshop on Computer-Aided Software Engineering (CASE '92); Montreal, Canada, 6-10th July 1992.

Stephens M. & Whitehead K. (1985); The Analyst - a workstation for analysis and design; Proc. 8th Int. Conf. Software Engineering; pp364-369; IEEE CS Press.

Systems Designers (1986); CORE - the manual; Internal Publication, SD-Scicon.

Wasserman A I & Pircher P A (1987); A Graphical, Extensible Integrated Environment for Software Development; Proceedings of 2nd Symposium on Practical Software Development Environments; SIGPlan Notices, 22, 1, pp131-142.

Wile D S (1991); Integrating Syntaxes and their Associated Semantics; USC/Information Sciences Institute Technical Report, 1991.

Wileden J C, Wolf A L, Rosenblatt W R & Tarr P L (1991); Specification-level interoperability; Communications of the ACM; 34, 5, pp72-87.

Zave P & Jackson M (1992); Conjunction as Composition; draft paper (to appear), 1992.