

Elicitation of Requirements from Multiple Perspectives

Steve Easterbrook

June 1991

Department of Computing,
Imperial College of Science, Technology and Medicine,
University of London,
London SW7 2BZ.

A thesis submitted for the degree of
Doctor of Philosophy of the University of London

Abstract

The success of large software engineering projects depends critically on the specification, which must represent the requirements of a large number of people with widely differing perspectives. Conventional approaches to software engineering do not address the process of identifying and integrating these perspectives, but instead concentrate on the maintenance of a single consistent description. This results in a specification which represents only one point of view, often the analyst's, excluding suggestions which do not fit with this view. The processes which led to the adoption of this point of view will go unrecorded, making any rationale attached to such a specification incomplete. Other participants will not be able to validate it properly, as it does not relate to their requirements.

This thesis integrates ideas drawn from the study of knowledge acquisition, computer-supported co-operative work and negotiation into a model of the specification activity which allows the capture of multiple perspectives, and resolution of conflicts between them. Perspectives are elicited separately, and each develops as an independent description of the requirements. If a description becomes inconsistent, it is split into separate perspectives which represent each side of the argument. Comparisons between perspectives can be made, and when conflicts are discovered, a process of computer-supported negotiation is invoked. This allows participants to teach each other about their perspectives, and elicit the issues on which those perspectives are based. Where resolution is required, new suggestions are evaluated using these issues. A set of tools has been developed to support the model, and the thesis illustrates how they might be used. Example analyses of case studies drawn from the software engineering literature are used to show how the tools can reconcile differing requirements descriptions.

The model and associated tools form an environment in which a collection of perspectives can be maintained, and relationships between them explored. This environment facilitates the communication needed to resolve conflicts. Validation is supported by allowing items to be traced back, through the resolution process, to their originators.

Statement of Contribution

This thesis describes a model for requirements elicitation from many sources. The model itself is the major research contribution of the thesis: it represents a novel approach to requirements engineering. There are two important innovations. The first of these is the idea of separately describing different perspectives. The second is the resolution of conflicts via a process of computer-supported negotiation. Parts of the model are drawn from other fields, including knowledge acquisition, computer-supported co-operative Work, organisational psychology, management science and decision theory. The way in which these fields are drawn together is novel, as is their application in the area of requirements engineering.

Various aspects of the model have been described in papers published elsewhere. The model was first outlined at the third European workshop on knowledge acquisition [Easterbrook 1989]. Chapter 6 will appear in a forthcoming issue of the journal *Knowledge Acquisition* [Easterbrook 1991]. This thesis should be regarded as the definitive treatment of the model.

Acknowledgements

This thesis wouldn't have happened if various people hadn't kept me going. First and foremost, Anthony Finkelstein supervised the whole thing, providing inspiration and guidance whenever it was needed. I owe a great deal to him. Various colleagues helped out along the way with feedback, discussion and/or distraction. Jeff Kramer, Jeff Magee, Tom Maibaum, Martin Sadler, and Manny Lehman all gave me good advice. Bharat Patel, Sati Sian, Stuart Kent, and many others at Imperial shared their ideas with me. Bill Robinson at Oregon helped me shape up my thoughts on conflict resolution. Mike Sharples and others at Sussex read and commented on various drafts.

I am very grateful to Brian Gaines for making a stay at the University of Calgary possible. I also owe many thanks to Maurice and Lois Sharp, Debbie and Richard Leishman, Dan Freedman, Rosanna Heise, Mildred Shaw, Ian Witten, Brian Woodward, Bruce MacDonald, John Boose, and all the others at KSI who looked after me and entertained me.

I was funded throughout by the SERC, under studentship number 87311891. The department at Imperial and the AISB provided me with various travel funds. Most importantly, my mum loaned me the money to buy a Macintosh, which proved vital (thanks mum!).

Finally, there is one person who was there all the time, to understand and support me. This thesis is dedicated to my wife, Sarah.

Contents

Abstract	2
Statement of Contribution	3
Acknowledgements	4
1 Introduction	8
1.1 Requirements Engineering.....	8
1.2 Multiple Perspectives.....	9
1.3 Conflict and Negotiation.....	9
1.4 Preview	10
2 Requirements Engineering	12
2.1 The Role of Requirements Engineering.....	12
2.1.1 Specifications.....	12
2.1.2 Constructing Specifications.....	13
2.1.3 Validating Specifications.....	14
2.1.4 Design Capture and Rationale	14
2.1.5 Exploration and Replay	15
2.2 Difficulties.....	15
2.2.1 Requirements Formulation.....	16
2.2.2 Nature of the Knowledge Sources.....	16
2.2.3 Nature of the Knowledge Involved.....	17
2.2.4 Negotiation	18
2.2.5 Conflict.....	19
2.2.6 Uncertainty	19
2.3 Objectives.....	20
2.3.1 Framework	20
2.3.2 Support Environment.....	21
2.3.3 Tools	22
2.4 Summary	22
3 Analytical Review	24
3.1 Requirements Engineering.....	24
3.1.1 Software Life-Cycle	24
3.1.2 Specification Languages	25
3.1.3 Specification Processes	26
3.2 Knowledge Acquisition.....	31
3.2.1 Building Knowledge-Based Systems	31
3.2.2 Machine Learning.....	32
3.2.3 Eliciting Conceptual Models.....	32
3.2.4 Specification as Knowledge Acquisition.....	33
3.3 Critical Analysis	34
3.3.1 The Single Viewpoint Bias	34
3.3.2 Conflict between Experts	35
3.3.3 Many viewpoints.....	36
3.4 Conflict Resolution.....	38
3.4.1 Terminology.....	38
3.4.2 Mathematical and Economic Models.....	39
3.4.3 Behavioural Models	44
3.4.4 Computational Models	48

3.5 Summary	51
4 Specification from Multiple Perspectives	53
4.1 Key Concepts	53
4.1.1 Perspectives and Viewpoints	53
4.1.2 Conversation	55
4.1.3 Representations	56
4.1.4 Evolution.....	56
4.2 Rationale.....	58
4.2.1 Separating Perspectives	58
4.2.2 Combining Knowledge Sources	59
4.2.3 Delaying Decisions	59
4.2.4 Constructing Specifications.....	60
4.2.5 An Example Problem.....	61
4.2.6 An Example Solution.....	63
4.3 Outline of the Model.....	64
4.3.1 Identifying Perspectives.....	65
4.3.2 Developing Perspectives	65
4.3.3 Comparing Perspectives	66
4.3.4 Resolving Differences.....	66
4.4 Method.....	67
4.4.1 Tools	67
4.4.2 Case Studies.....	68
4.5 Summary	69
5 Modelling Separate Perspectives	70
5.1 Identifying Perspectives	70
5.1.1 Evolving Perspectives.....	70
5.1.2 Viewpoint Hierarchies.....	73
5.1.3 Placing Statements.....	75
5.1.4 Functionality of Viewpoint Creation Tools	78
5.2 Reasoning Within Viewpoints	78
5.2.1 Viewpoints and Commitments	78
5.2.2 Inference Rules and Conflict Detection.....	79
5.2.3 Commitment Reasoning Scheme.....	79
5.2.4 Functionality of Viewpoint Manipulation Tools.....	81
5.3 Deriving Viewpoint Descriptions	81
5.3.1 Interpreting Interview Transcripts	82
5.3.2 Functionality of Annotation Tools	83
5.4 Integrating Viewpoints.....	84
5.4.1 Blackboard.....	84
5.4.2 Exploration	85
5.4.3 Agenda.....	86
5.4.4 Functionality of Integration Tools	86
5.5 Conclusions	87
5.5.1 Implementation.....	87
5.5.2 Advantages	87
5.5.3 Problems.....	88
5.6 Summary	89
6 Computer-Supported Negotiation	90
6.1 Conflict Resolution.....	90
6.1.1 Sources of Conflict	90
6.1.2 Consequences of Suppressing Conflict	91
6.1.3 Role of Communication.....	92
6.2 Conflict Resolution Model.....	92
6.2.1 Resolution Context	93
6.2.2 Detection of Conflicts	94

6.2.3	Example Conflicts	94
6.3	Exploration Phase	95
6.3.1	Establishing Correspondences	96
6.3.2	Identifying Conflict Issues.....	98
6.3.3	Agreeing Resolution Criteria	99
6.3.4	Functionality of the Exploration Tools	100
6.4	Generative Phase	102
6.4.1	Types of Conflict	102
6.4.2	Generating Resolution Options.....	103
6.4.3	Support for the Generative Phase.....	104
6.5	Evaluation Phase.....	105
6.5.1	Relating Options to Issues	105
6.5.2	Relating Options to One Another.....	106
6.5.3	Choosing a Resolution	106
6.5.4	Support for the Evaluation Phase	106
6.6	Summary	107
7	Conclusions and Further Work	109
7.1	Summary	109
7.1.1	Problem Domain	109
7.1.2	Objectives.....	110
7.1.3	Solution	110
7.2	Critical Review.....	111
7.2.1	Rationale	111
7.2.2	Advantages	111
7.2.3	Remaining Problems	112
7.3	Future Work.....	113
7.4	Summary	114
8	References	115
9	Appendix	125
1	Algorithm for splitting agents.....	125
2	Rules for creating new sub-viewpoints.....	125

1 Introduction

Matching software to needs is a problem. Describing the needs as a set of precise requirements is one step towards solving that problem. This thesis describes a framework for the elicitation of differing requirements from many sources, and the subsequent process of accounting for and resolving discrepancies in the gathered knowledge.

1.1 Requirements Engineering

The success of the design process depends critically on the designers' understanding of the problem. In software engineering, this understanding is encapsulated in the requirements specification, which describes, precisely and unambiguously, both requirements for the delivered system and any restrictions on the development process. Any formal verification of the software is based on this specification, and so is tied to the quality of the specification. In effect, the specification acts as an anchor and a communication channel throughout the software development process. The elicitation and formulation of requirements to produce a specification can be termed *requirements engineering*.

For large software development projects, the requirements must be communicated to all members of the development team. The difficulties involved in communication and co-ordination distinguish *programming-in-the-large* from *programming-in-the-small*, where the former involves teams rather than individuals. The field of software engineering has produced various methodologies in an attempt to facilitate management of large software projects, and to encourage more rigorous approaches.

A fundamental goal of research in this area is automation: it is widely accepted that development of software will become easier and cheaper if more of it can be formalised and automated. In effect, this has meant that the boundary of automation has crept steadily upstream, to the earlier stages of the development process [Balzer 1985]. Complimenting this trend is the introduction of formal methods which ensure that each step in the development process is formally justified, allowing the products of the process to be formally verified.

Precise definition is vital for the requirements specification. There is always a degree of uncertainty and informality in original needs, and these needs will themselves be changed by the development and introduction of a software system. However, large software projects cannot be co-ordinated if the requirements are woolly or ambiguous. Hence, requirements engineering involves the translation of informal and incomplete needs into a precise description of what a software system will be expected to do. This inevitably involves deciding exactly which needs will be met, and resolving conflicts between competing needs. All later stages of the project rely on this precise description as the prime source of knowledge about the requirements.

As requirements engineering is concerned with the interpretation and translation of the initial informal needs, it can never be completely formalised. Rather, the major goal of requirements engineering is to provide better support for the construction and validation of specifications. This support should include guidance for the elicitation and formulation of requirements and the subsequent validation process. This thesis concentrates on how these activities might be supported.

1.2 Multiple Perspectives

A specification must be representative of the various needs of different people. For example, users will have different needs from maintainers or administrators, and even from other users. Their individual needs will form the basis of their view of what is required. The validation of a specification tests that all these views have been addressed and are adequately represented. Ideally, all the people concerned should be able to identify with the specification and agree it as representative. The need to ensure that the specification reflects different views suggests that these views need to be identified and represented.

The model of requirements engineering developed in this thesis allows the diverse perspectives to be elicited separately. We use the term *perspective*, to emphasise the fact that there is no simple relationship between requirements and people, or even between requirements and rôles. A perspective is simply an area of knowledge which is internally consistent. Each perspective will normally have an identifiable focus of attention, being the motivating concern of the requirements it represents.

The model allows the analyst to maintain a set of separate descriptions to represent perspectives. These perspective descriptions are developed independently, so that disagreements between them are ignored while the perspectives are elicited, even where the disagreements are terminological. Because the analysis does not rely on fitting elicited knowledge into a single consistent model, all elicited information is represented, even that which might otherwise have been discarded as inconsistent. This forestalls competition between perspectives in the early stages of elicitation, allowing analyst to concentrate on information gathering.

The elicited perspectives are regarded as knowledge bases which together represent the current state of the evolving requirements. Taken together, the perspectives may be inconsistent and incomplete. However, individually each perspective is consistent, and so can be interrogated and manipulated by the analyst. The processes by which perspectives are identified and developed are recorded so that the development history can be traced.

The combination of perspectives to form a specification inevitably involves negotiation. In order to record the course of this negotiation, and allow it to be controlled properly, it needs to be made an explicit process. If the negotiation were carried out as the perspectives are elicited, with only a single resulting description maintained, there would be no way of checking that the result is representative, or that the negotiation was acceptable. On the other hand, the negotiation process involves a degree of elicitation, as additional supporting information may be needed. Hence, while perspectives are elicited separately, comparisons between them will need to be made, which will in turn assist with the elicitation process. Both the perspectives and the comparison process need to be recorded for validation purposes.

1.3 Conflict and Negotiation

In seeking to represent the perspectives of the many participants in the specification, the problem of conflict inevitably arises. In the computing literature, mention of the need to handle conflict is rare, which is perhaps surprising given the importance attached to it in the social sciences. For example, management studies and sociology recognise that conflict plays an important role in group interaction [Robbins 1989]. Some recent software engineering research has identified conflict as an issue (e.g. [Curtis, Krasner & Iscoe 1988]), although as yet little progress has been made towards understanding how conflict might be handled.

Examples of many different types of conflict abound in the literature on organisational psychology. For example, Robbins [1974] describes a newly elected city manager who has promised an immediate improvement in rubbish collection. After several months the citizens complained that

there was no improvement. On investigation it turned out that the citizens regarded “improved service” to mean more frequent collection, whereas the city manager had meant earlier, quieter and more economical collection. Thomas [1976] illustrates a different type of conflict, in an example of a manager who wanted his staff to adopt a new form for their weekly reports, whilst they preferred the old form. Again, investigation revealed the underlying cause: the staff did not feel they had the time required to complete the new form, while the manager needed the extra information which was not on the old form. Both these examples are typical of organisational conflict, and it is not difficult to see how similar conflicts arise during the introduction of a new software system. Similarly, the group developing the software may be subject to such conflicts during the development process.

In this thesis, the term conflict is used in its widest sense, to cover any interference in one party’s activities, needs or goals, caused by the activities of another party. Conflict can be characterised as disagreement among the originators of the requirements and this disagreement may lead to inconsistencies in the specification. However, disagreements do not always lead to inconsistency, and inconsistencies do not always indicate the presence of conflict.

Typical software engineering methodologies do not explicitly address the resolution of conflict. Such methodologies are geared towards maintaining consistency and so do not allow conflicts to be expressed, let alone constructively resolved. Indeed, we could characterise existing software methodologies as conflict avoidance, in that they prescribe particular approaches, which assist software practitioners in breaking problems down and resolving design decisions in particular ways. In this way uncertainty is reduced by the provision of the collective wisdom embodied in the methodology [Lehman 1990].

While this approach helps to avoid conflicts during development, it does not help to deal with conflicting requirements. Most methodologies require a single consistent specification as a basis for a coherent design, which means that conflicts are suppressed when the specification is written. Formal methods do not necessarily help, even though formal languages are intended to prevent ambiguity and inconsistency [Finkelstein, Finkelstein & Maibaum 1990]. The ability to formally reason with specifications is a huge step towards detecting the presence of conflict, but carries the implication that inconsistencies are errors which must be eliminated. Methods developed to support formal specification reflect this philosophy, and miss the chance to explore conflict.

If much of software engineering is geared towards conflict avoidance, this in itself may not be a problem, for two reasons. Firstly, conflict may not be as extensive in requirements engineering as has been suggested, so that studying it might only help in exceptional cases. Secondly, avoidance is a valid way of tackling conflict, especially where it prevents energy being wasted on fruitless confrontation.

In this thesis, we argue that conflict is extensive in most real domains, and that avoidance is not a satisfactory approach to handling conflicting requirements. Examples are given which show that conflict is important enough not to be suppressed, and that a specification can be enhanced by handling conflict in more direct ways.

1.4 Preview

The central argument of this thesis is that modelling multiple perspectives separately and then examining and explicitly resolving conflicts between them will result in a more precise and more representative specification. Participants can identify more easily with a specification in which their contribution can be traced. Furthermore, the process of comparison of perspectives elicits issues and assumptions that might otherwise remain hidden.

The thesis presents a model of the requirements process which provides guidance for identifying and developing descriptions of the perspectives, and resolution of conflicts between them. A

perspective can be thought of as a consistent view of the world arising from the context of a particular role. Perspectives do not necessarily correspond to people, as one person may use several perspectives, and a perspective might be shared by several people. Perspectives are represented using *viewpoints*, which are formatted descriptions in some appropriate representation scheme.

No restriction is placed on the form of these descriptions, nor on the degree of formality. Hence the model may be used in conjunction with existing specification languages and knowledge representation schemes. The novelty of this work lies in the explicit support for alternative and competing descriptions, and in the provision of a framework in which conflicts can be explored fully.

The next chapter introduces requirements engineering more fully, describing some of the difficulties of the task. Objectives for a framework for requirements engineering are set out. Chapter 3 reviews existing work in software engineering, and in knowledge elicitation, concluding that most existing techniques produce descriptions which represent a single perspective. This observation is used as a pretext for an analytical review of fields from the social sciences which cover aspects of multiple perspectives and conflict resolution.

The remaining chapters describe the model. Chapter 4 explains the ideas underlying the model and discusses the rationale. Four main problem areas within the model are discussed. Elicitation, based on the capture of perspectives, introduces two of these problems: how to identify perspectives and how to construct the viewpoint descriptions. The other two problems arise when integrating (possibly conflicting) perspectives: how can viewpoints be compared and how can differences be resolved.

The processes of identifying and recording the perspectives is described in chapter 5, together with a system called *Analyser*, which was developed to support those activities. The model approaches the problem of identifying perspectives by splitting viewpoints when distinctions between perspectives are discovered. The split viewpoints form an inheritance hierarchy, with shared knowledge inherited from a common ancestor. Descendant viewpoints represent specific areas over which perspectives disagree. The viewpoints represent only that which their originators have explicitly stated, ensuring that they remain accurate models of elicited knowledge.

Chapter 6 presents a model for resolving conflicts between perspectives through a process of computer-supported negotiation. As conflicting viewpoints may co-exist in the model, there is no compulsion to resolve conflicts until a resolution becomes necessary or desirable. The resolution process itself involves three phases, of which the initial, exploratory phase is the most important. In this phase, participants compare the conflicting viewpoints, and identify points of correspondence and disparity between them. Issues underlying these correspondences and conflicts are elicited, in order to come to a better understanding of the conflict. In the second phase, a number of resolution options are generated, according to the types of conflict involved. The final phase involves comparing these options to one another and to the underlying issues, in order to choose the combination which best satisfies the participants.

Chapter 7 presents conclusions. It discusses the applicability of the multiple perspective model, and the advantages it offers. Remaining problem areas are discussed, along with plans for future research.

2 Requirements Engineering

The term *requirements engineering* describes the processes leading to the production of a requirements specification. Much of software engineering research takes the existence of this document for granted, concentrating instead on the downstream areas of software development. In this chapter, we argue that the problems of requirements engineering deserve greater study. To understand why this is so, we consider the role of the specification in the software engineering process, and describe the issues which must be addressed during specification construction. The difficulties of requirements engineering come from many directions, including the sheer quantity of knowledge involved, the inherent uncertainty, and the need for negotiation between conflicting requirements. We conclude that a prescriptive framework for requirements engineering is highly desirable, and describe a number of objectives for such a framework.

2.1 The Role of Requirements Engineering

The phase of the software engineering process that begins with an informal statement of need and produces a requirements specification is generally referred to as *requirements analysis*. Ideally, the resulting document should contain all the information about the requirements that might be needed during the design stage, although in practice, the clients' perceived needs will change during the lifecycle of the system. If the requirements aren't clearly defined, the result is uncertainty throughout the software life-cycle.

The importance of the requirements specification implies that a great deal of effort should be invested in the creation of such a document [Boehm 1981]. Creating the specification is far more than just analysis: it involves eliciting relevant knowledge; understanding the task and its social and organisational context; negotiating with the client over the scope, contents and language; resolving conflicting requirements; and synthesizing appropriate structures for describing the result. Use of the term *Requirements Engineering* has been proposed to indicate the complexity of this process, and to convey the message that specifications need to be carefully constructed.

This section examines the importance of requirements engineering in the software process, concentrating particularly on the demands made of the specification. We examine the roles the specification must play, pointing out that it is both a contract and a communication channel. We also consider how specifications are constructed and validated. Finally, sections 2.1.4 and 2.1.5 look at the exploratory nature of requirements engineering, and the importance of recording the development history of the specification, and its underlying rationale.

2.1.1 Specifications

Specifications have a vital role to play in the software engineering process, as the only precise description of needs. The specification provides a way to verify the correctness of the eventual design and implementation. If the specification is inappropriate, verification will be pointless. Information omitted from the specification will not be taken into account in the design process. Ambiguities in the specification lead to uncertainty throughout the process. Misunderstandings and errors in the specification will lead to designs which, while complying with the specification, do not properly satisfy the needs of the users.

Typically, the specification acts as a contract between the clients and the software developers, and as such is the main channel of communication between them. Balzer & Goldman [1979] describe

three criteria by which a specification should be judged, namely: it must be clearly and unambiguously understandable by both parties; it must be testable that any implementation satisfies the specification, and that the specification meets the needs it is designed for; and it must be easy to modify, as the requirements will change over time. These criteria reflect the contractual nature of specifications.

The specification also acts as a channel of communication amongst the software team. As the main source of information about the clients' needs, it defines what will be common knowledge among the developers. Too often, the specification does not adequately fill this role, and a recent field study of behavioural aspects of software developers [Curtis, Krasner & Iscoe 1988] concluded that many software teams depend upon a single *exceptional designer*. Such designers are characterised as having a deep understanding of both the application domain and the design process. In such cases, this designer is a better source of information than the specification. However, the development team will not all have equal access to such person, and so will be working with different amounts of knowledge. Clearly, it is preferable to express as much of this knowledge as possible in the specification, in order to ensure dissemination.

Because of its accessibility, the specification ought to facilitate co-operation between the various parties. However, there is evidence that it frequently fails to do this. The study described above suggested that exceptional designers are able "to integrate different, sometimes competing perspectives on the development process". In other words, the specification is only providing one perspective, and there are other important points of view which it has excluded. This leads to two major problems, namely: whole sets of knowledge and ideas are ignored by the specification; and the people concerned will lose faith in the specification. The software team can become fractured, as such people attempt to exert influence in other ways, and will become dependent on the existence of a team member with the background knowledge and communication skills required to resolve the problem.

We have identified two major roles of the specification, as a communication medium, and as a yardstick by which design and implementation are judged. In order to fulfil these roles, the specification must be unambiguously understandable, testable, and modifiable. The trend towards formal specification languages is an attempt to satisfy at least the first two of these criteria.

In addition to these criteria, it has been suggested that the specification should be representative of the many people that contribute to it [Zave 1982], as each contributor may have some unique insight or perspective that might otherwise be ignored. More importantly, potential users are unlikely to co-operate in the development, nor accept the final system, if they feel their views have not been taken into account. In the worst case, participants would have to circumvent the specification in order to get an issue raised.

We have referred throughout this discussion to specifications as though they are of a uniform type. In fact, several types of specification can be distinguished, which are derived in different ways and have different uses. For example, it is common to distinguish between requirements specifications and design specifications, where the former describes needs, and the latter describes how those needs are to be met. While this discussion is directed at requirements specifications, other types of specification also have a communicative role within the portion of the lifecycle in which they are used.

2.1.2 Constructing Specifications

As the specification has several important roles to play, it needs to be carefully constructed. It will be read by a number of different people, with widely differing backgrounds, and so must be accessible to them. Presentation is therefore important. It is essential that the specification should answer the types of question that various groups of people are likely to ask of it. In other words, it

should be easy to interrogate. Above all the specification should be regarded as a designed artefact, itself created to fill certain needs.

Throughout the life of a project, pressures will arise for changes to the specification. These have many causes, from understanding gained in attempting to satisfy the current specification, to changes in the environment of the system. Where such changes are incorporated, the specification should be modified accordingly, in order that it remain up-to-date. If the specification is not up-to-date it will fail in its communicative role. Specifications, therefore, must be easy to annotate and modify. Although some would argue that the contractual nature of specifications implies that they should not change, it is clearly preferable to negotiate a change in the specification than to deliver software which does not meet the specification.

The specification's role as a source of information suggests it should include some form of database or knowledge base. Work on knowledge-based systems have provided some useful ideas both for modelling requirements (e.g. Borgida, Greenspan & Mylopoulos [1985] – see §3.1.2) and for reasoning with the specification (e.g. Reubenstein [1990] – see §3.1.3.3). A knowledge based component can also facilitate access to and management of the body of information gathered during requirements engineering. Such a component can be used to interrogate the specification throughout the lifecycle.

2.1.3 Validating Specifications

The specification must be sufficiently precise to determine whether subsequent designs and implementations meet it, a process known as *verification*. If the specification is in a formal language, the verification process can be mathematically rigorous [Bjorner 1987], and to a certain degree, automated.

However, verification does not ensure that the specification is correct. The specification must be *validated* with regard to the original needs of the client. This process is particularly difficult, as there is no definitive statement of those needs: the requirements specification *is* the first precise description of those needs. Because of this, validation cannot be formalised, and must remain a subjective human activity, requiring input and discussion by the originators of the needs [Blum 1985].

Validation can only proceed if the participants can relate the specification to their needs, and is only successful if the specification is relevant to those needs. An important facility for validation is traceability [Alford 1977]. If components of the specification can be traced back to the original statements that inspired them, then the participants can assess the relevance more readily. Also, if a statement has been misinterpreted, the parts of the specification which are based on it can be identified.

Validation is an important part of requirements engineering. As it requires the originators' participation, it is likely to be considerably smoother if those people have participated throughout the requirements process. Such an involvement means that the requirements can be validated as they evolve, rather than when they have been refined into a specification.

2.1.4 Design Capture and Rationale

Often the ability to trace a component of the specification back to an originating statement is not enough to understand its purpose. For this, the process that led to the current specification needs to be recorded. Given that specifications are designed artefacts, then recording the derivation is a form of design capture. The design history must record the decisions made and their rationales in order to be of use.

Decision capture can be problematic, as rationales tend to be idiosyncratic [Kaplan 1989]. Analysts are experts in their jobs, and may have difficulty explaining their actions to others who need to understand the specification. Furthermore, explanations are usually tailored to a particular audience. If the analyst is recording a rationale, it is not clear who it should be aimed at. Also, understanding decisions involves making the goal of the participants clear, which is difficult as these goals are often unconscious, and involve many implicit assumptions. There needs to be a way to prompt for these goals, and to encourage all participants to think about the decisions involved.

While it is unlikely that requirements engineering can be automated, some degree of automation might be introduced for recording the process. Where interactive tools are used, the operations used can be recorded, automatically, as a basis for the attachment of rationales, using a *machine-in-the-loop* [Green *et. al.* 1983]. To a certain extent, such tools can elicit the rationales used for the operations being carried out, and automatically record these. The entire process history represents a documentation of the requirements engineering process, and should be stored with the specification as a supplementary source of information. However, storing and manipulating this documentation is a huge knowledge management task, from which the analyst should be freed.

2.1.5 Exploration and Replay

While software engineering aims to produce higher quality software, it is not always possible to get it right first time [Brooks 1975]. All models of the software lifecycle allow for a degree of feedback and revision (see §3.1.1). The causes are well known: no-one has perfect foresight to predict what they will want in the future; clients are not even certain about what they want now; and the consequences of particular requirements cannot be foreseen [Swartout & Balzer 1982]. Furthermore, the introduction of a new system may itself generate new requirements.

All these problems suggest that some form of exploration is desirable. For the later stages of software engineering, exploratory programming has been proposed. The specification process, on the other hand is naturally an exploratory process, in which the participants explore the requirements. Such an exploration is essential, as analysts will be unsure of the clients needs, and the clients will be unsure of what is possible. Once an initial specification is produced, clients will want to explore how it relates to their needs before accepting it.

A particularly useful tool for exploration is the ability to re-trace steps, undoing previous actions. This allows the participants to explore the consequences of a particular development without having to commit themselves to it. In a large project, however, this can be problematic as the action to be undone may have been originated by someone else, on a previous date. The facility therefore depends on the accurate recording of rationales and the ability to trace dependencies.

A related facility to the undo is the ability to replay parts of the process. This can allow re-use of previous, similar systems, by replaying their development, making changes where necessary. It can also simplify program evolution, as alterations can be made to the specification, and the development process replayed to generate a new implementation. Again, this depends on the capture of rationales and tracing of dependencies.

2.2 Difficulties

Specification construction is a difficult task. It is of a type of problem that has been termed *wicked*: it is ill-structured and open-ended, and the knowledge available is incomplete [Partridge 1978]. Most importantly, there is no notion of a finished specification, and the only criteria for stopping is some form of satisfaction.

Five major areas of difficulty can be identified as contributing to the problems associated with the formulation of requirements. The knowledge sources are diverse and vary greatly in the quality of information and exposition; the actual knowledge takes many different forms; the contents of the specification need to be negotiated; conflicts need to be resolved; and the knowledge itself is uncertain and unstable. This section examines each of these problems in turn. The problems are compounded by the nature of the software engineering process itself, in that the sheer size of the process makes effective communication vital, for which a solid common understanding of the problem is needed. Furthermore, the resulting specification must be (to some degree) consistent and complete.

2.2.1 Requirements Formulation

Before the specification can be constructed, the necessary groundwork must be laid, which involves knowledge elicitation and the formulation of the requirements. Both of these are difficult tasks. Elicitation of knowledge forms a major part of the process, whether or not it will eventually form a knowledge base. There are a number of well known problems in elicitation [Gaines 1987], caused by the tenuous nature of knowledge, the difficulties people have in articulating it, and the element of irrelevant or misleading statements. Like knowledge engineering, requirements engineering involves the extraction and representation of information through some form of interaction with the experts, in this case the clients. Both have the same set of techniques available for extracting the information, including various types of interview, observing people in action (and subsequent debriefing), tutoring, and case analysis. Most of the information gathered in this way is needed throughout the lifecycle of the software.

Finkelstein & Finkelstein [1983] describe the processes involved in requirements formulation. There are three basic systematic methods: the use of check lists; lateral (or divergent) idea generation; and formal specification languages. Decomposition and abstraction are important parts of the process, removing barriers to innovation. The primitive concepts used in design come from a number of sources, including: existing designs, analogy, convergent deduction, and divergent thinking. (See also Carter *et. al.* [1984] for a particularly graphic and entertaining account of requirements formulation).

Specification construction can be seen as an evolutionary process. The incremental steps either add more detail, or clarify existing parts by introducing exceptional case behaviour or retracting incorrect or over-simplified statements [Feather 1987]. We have also noted that specifications are designed artefacts, and hence the specification process involves design. Dubois & Hagelstein [1987] point out that requirements engineering differs from typical software design tasks in that the latter involve artefacts such as programs, and are done exclusively by specialists, while requirements engineering involves real world concepts, and requires extensive communication with non-experts.

2.2.2 Nature of the Knowledge Sources

There are many people involved with the development of a large software system, both in its design and use. Whilst these people can be broadly classed into groups, such as users (often of several types), management and software developers, these groupings will not enforce conformity of the individual members. Each person is a potential source of knowledge, and might contribute a unique insight. Additionally, people are not the only sources of knowledge, and valuable information may be found in manuals, memos, and other documents.

Much of the knowledge is difficult to elicit as the subjects need to untangle it, and the analyst needs to know how to organise it. For example, it is hard to disentangle the requirements from the goals and perspectives (and associated opinions and biases) of the people involved. It is usual for example to restrict the specification to what is desired rather than how it is to be achieved, to distinguish between functional and non-functional requirements, and to distinguish the system

model from its environment. People who are not conversant with analysis techniques will be unable to make such distinctions, and so cannot untangle the various contributions they are making. They will certainly be unlikely to offer information in a form that neatly fits with the organisation the analyst is developing.

Each person has a specific idea about what they want from the resulting system and how it will help (or hinder) them, and so are most concerned to influence its design. These individual goals colour their perspective of the requirements, and cause them to introduce systematic bias into their responses. Furthermore, organizations have goals and policies which may inhibit its members, again biasing their responses. This contrasts with knowledge elicitation for expert systems, where the expert is unlikely to have any particular requirements concerning the resulting system, and so can provide more objective responses. Unfortunately, knowledge acquisition research has shown that even disinterested experts are not free from bias [Meyer & Booker 1989], and it is likely that most of these forms of bias also crop up in requirements elicitation.

The analysts themselves are also sources of knowledge, providing a great deal of experience about requirements for similar systems, and knowledge about the likely effects of particular decisions. There is inevitably a temptation for the analyst to impose his or her own preferences in the specification, and studies have shown that convincing the clients that a chosen solution is suitable is the largest part of the analyst's job [Fickas Collins & Olivier 1987]. Even where the analyst has experience of the domain in question, his knowledge will be of a subtly different flavour to that of the client, and there will be subtle differences in the detailed requirements of the particular situation. The analyst must be careful, therefore, to blend his or her knowledge with others' contributions.

It is often difficult to compare the knowledge offered by different subjects, as their experience varies, both in type and level. Kolodner [1984] points out that experience changes the way a person reasons, and that the key difference between experts and novices lies not so much in the level of knowledge, but the ability to apply knowledge more effectively. Dreyfus & Dreyfus [1986] describe the process of evolution from novice to expert, and suggest that novices use simple heuristic rules, while experts internalise their knowledge. This implies that experts are less able to explain their behaviour than novices. Also, people have a tendency to tailor their explanations to the hearer, so for example a layman would receive a superficial answer, while an expert would receive a detailed response [Compton & Jansen 1989]. All these factors combine to make it very difficult to compare knowledge elicited from people with different levels of experience.

Another reason knowledge is difficult to compare is that subjects use different assumptions, and these assumptions are not always obvious. While some sources will articulate what others have left as assumptions, it is not always clear when this has happened, nor is it always obvious when people have made incorrect assumptions about areas of knowledge they are not expert in. As there are a large number of assumptions accompanying any communication act, ranging from assumptions that the hearer understands the language, to very specific assumptions about the domain, it is impossible identify all of them.

2.2.3 Nature of the Knowledge Involved

Many areas of knowledge need to be taken into account. In a large-scale project, the information needed includes knowledge about software engineering methodologies and the target machine's architecture, as well as application domain knowledge. Furthermore, for maintenance purposes, knowledge about the target software and the design history are also needed. Much of this knowledge is specific to a particular project; for example, knowledge about the needs which gave rise to the current study, knowledge about typical user behaviour, knowledge about performance requirements of the various components of the system, and knowledge about the software development process (its goals, its progress, its restrictions, etc). In his excellent survey of the use

of AI in software engineering, Barstow [1987] describes the roles played by these types of knowledge and argues that AI techniques could greatly assist with the handling of this knowledge.

The analyst will use many representations to capture the variety of information as a matter of course [Burton & Shadbolt 1987], as no one representation has sufficient expressive power for the many different types of knowledge [Sloman 1985]. Even natural language is frequently supplemented with diagrams and other devices. When a single specification language is to be used ultimately, during elicitation the analysts will still have to handle other representation schemes, as people attempt to explain their contributions in idiosyncratic ways. Indexing and cross-referencing multiple representations is a difficult knowledge management task in itself, for which extensive support is desirable.

As well as facts about the domain, relevant policies and preferences need to be considered, and these are problematic to represent. They cannot be represented as specific facts, nor as verifiable goals. Rather they are heuristic-like guidelines that restrict and channel the design process [Anderson & Fickas 1989]. The need to choose between the diversity of methodologies available for software engineering strengthens the role of institutional policy and individual preference in the decision process.

Finally, the sheer volume of knowledge compounds these problems, and makes the management of the knowledge difficult. Whilst consistency checking can be handled in a small knowledge base, as the size increases such checking rapidly succumbs to the combinatorial explosion. When a huge amount of information is involved any consistency checking is prohibitively difficult. In requirements elicitation, inconsistencies occur frequently, usually indicating a conflict between the interested parties. In such cases, the conflict represents the need for an explicit decision by the analyst, which should not be taken until all the appropriate information has been gathered. Hence all alternatives must be captured and accommodated to allow the analyst to delay these decisions as necessary.

There is currently a paucity of computer support available to the analyst to manage this knowledge. Most existing software tools are geared to producing manual representations for the information. This means that most of the knowledge needed is held in the software practitioners' heads: only a small proportion is explicitly stored in the documentation. This increases problems of communication across the software team. If the analyst never explicitly states such knowledge, it can only be passed on by word of mouth to others involved with the software life-cycle.

2.2.4 Negotiation

Because of the diversity of sources and types of knowledge, there will be many differences of opinion. Requirements engineering can be seen as the resolution of the various constraints and goals of the people involved, and their integration into a single consistent specification. Conventional analysis techniques do not address the resolution process directly, and so it usually begins in the analyst's head. Part of the analyst's skill involves the juggling of competing requirements and negotiating with the client.

Each participant brings a number of preconceptions or biases into the specification process, which are adapted as the process proceeds. One of the reasons for this adaptability is that clients are not always sure of what they want or what is feasible. The analyst's expertise can be used to guide them into understanding their needs better, as part of the analysis process.

However, too often the analyst takes this as an opportunity to impose his or her own solution. This arises from the view of analysis as a process of translating user intent into formal or semi-formal documents. The usual approach is to learn about the client, analyse the data, and then make suggestions. Presenting these suggestions to the client takes most of the time, as the client needs to be convinced that the analyst's interpretation is right [Fickas, Collins & Olivier, 1987]. As analysts

naturally have preconceptions of the requirements, these will be used as a basis of their understanding of the problem. Information gathered during elicitation is used to construct a description which reflects these preconceptions. Unfortunately, there is a danger that this will commit the specification to one particular understanding of the requirements, and information which conflicts with the current description will be discarded.

A better approach would be for the resolution to take the form of negotiation amongst clients and analysts, where the analyst and clients share their knowledge, and enter into a mutual process of making suggestions, and critiquing each other's suggestions. Unfortunately little computer support is available for synthesizing a solution from the various suggestions, nor for resolving the conflict inherent in the process.

2.2.5 Conflict

Even if negotiation is used as a basis for requirements engineering, there will be points at which explicit resolution of conflict becomes necessary. Such conflicts arise for a number of reasons, varying from misunderstandings to differences of values. In many cases it is not obvious what the cause of the conflict is, nor how to resolve it.

Traditional software engineering methods do not address the resolution of conflict, but try to avoid it. There are a number of problems with any model which avoids conflict. As there is no means of expressing conflict within the model, where it does exist it is suppressed. This can cause problems: the resolution must be carried out outside the framework of the model and consequently is likely to be carried out at an inappropriate time, using an undesirable method. In addition, resolution thus achieved is untraceable, making rationales invalid, and the process irreproducible.

Failure to recognise conflict between the perspectives of the participants will lead to confusion during the analysis phase, which will continue throughout the lifecycle. Often, they will lead to a single perspective being adopted as the basis for specification at the cost of any alternative perspectives. In this process useful ideas associated with the rejected perspectives will be discarded, along with the goodwill of their originators. Communication between participants will suffer, and will often breakdown altogether if there is an "injured" party.

To handle conflict properly, the specification needs to model the perspectives separately, so that their interaction can be studied. We can draw on many fields which have addressed some or other aspect of conflict resolution, in order to understand the processes involved. The next chapter surveys research on conflict in relevant fields, discussing in each case how the work might apply to software engineering.

2.2.6 Uncertainty

One of the main aims of software engineering has been to formalise as much of the development process as possible, in order to reduce the arbitrary nature of the software process, and to introduce automation. This formalization facilitates verification, which has been advocated as a means of improving reliability. However, a formal verification can only be used to demonstrate that an implementation fulfils the formal requirements specification; in other words that no errors creep in during the design process.

Because of the informal nature of the business environment in which the eventual system must operate, there will always be a degree of uncertainty in the requirements [Balzer, Goldman & Wile 1978], [Lehman 1990]. In particular, the initial step of requirements engineering begins with an informal statement of need, and so there can never be a guarantee that a formal specification describes exactly what is required.

One symptom of this uncertainty is that the specification gets altered once implementation is underway. The fact that the process of specification cannot be fully separated from the implementation has already been noted. Swartout & Balzer [1982] identify two main reasons for alterations to the specification once the implementation is underway. The first of these involves physical limitations arising from implementation decisions. The second reason is the most interesting, and is put down to lack of foresight in the specification. The implementation may yield new insights into the requirements of a task, and as such the entire software engineering process can be seen as one of prototype refinement [Giddings 1984].

Another barrier to formalisation of requirements engineering is the need for negotiation. Studies of conflict resolution show that the most successful methods require a degree of creative input [Fisher & Ury 1981]. Formal approaches to conflict resolution have a tendency to produce compromise solutions which do not properly satisfy any participant's needs [Luce & Raiffa 1957].

The uncertainty and the need for creativity means that requirements engineering can never be fully formalised. However, there is plenty of scope for prescriptive methods and tools to support the process. Cunningham *et. al.* [1985] give a number of dangers that proposed specification models have suffered from, which can be translated into recommendations. These include: lack of a method; difficulty of grafting methods onto existing procedures; stultification of creativity; and univocality, as few methods support elicitation from many sources and their consolidation into a consistent specification. Formal methods which do address these problems could provide a powerful framework for requirements engineering.

2.3 Objectives

We have discussed the importance of requirements engineering, and the particular problems relating to the process. Clearly, a model of requirements engineering is needed, which allows the analyst to overcome the difficulties. In this section we present a set of objectives for such a model, and for tools to support it. Note that we are not proposing automation. Case studies of analysts at work (e.g. Fickas, Collins & Olivier [1987], Adelson & Soloway [1985]) have revealed that a broad range of skills are employed by analysts. It is unlikely that the full range of these skills can be automated.

2.3.1 Framework

Rather than a rigid formal process, the analyst needs a framework which can guide his or her expertise. This framework must support the creative input and interpretive skills of the analyst. Finkelstein & Fuks [1989] suggest that such a framework be: flexible; empirical (in that the model maps onto the results of observational studies); enactable; co-operative; and that it should be able to handle conflict.

However, in order to facilitate some automation, a degree of structuring must be introduced. An approach which supports an incremental, evolutionary process is needed. The individual steps which build the descriptions arise out of the dialogues between participants, and a model that is overly-prescriptive will severely limit the scope of these dialogues, possibly causing vital steps to be missed. Therefore, any automated support or formalization must account for, and indeed encourage dialogue as exploration of the current state of the specification. One of the results of this requirement is that specification comes to be seen as a conversational activity.

In order to allow the participants to control the process, the model must allow any order of discussion. In other words, it cannot be guaranteed that needed information will be provided immediately. Rich, Waters & Reubenstein [1987] discuss the inevitable informality of human communication, listing abbreviation, ambiguity, poor ordering, incompleteness, and contradiction as key features. These features represent an essential part of the human thought process, as a

means of dealing with complexity. People present ideas in the order they occur, not in an order which is convenient to the hearer. In particular, the human mind is adept at ignoring inconvenient consequences of particular statements with the intention of clarifying them later. Studies of human designers have shown that they frequently make notes to themselves to return later to a particular item [Littman 1987]

Finally, the model must allow the participants to delay the resolution of conflicts and the making of decisions. Requirements engineering is primarily an exploratory process, involving the gathering and formulation of knowledge. It is vital therefore, that it does not become overly-restricted by premature decisions. A framework for the process should encourage participants to gather all the relevant knowledge and explore all the issues before making a decision.

2.3.2 Support Environment

In addition to a model for the specification process, we need to consider what kind of support is needed. The automated tools should form an environment in which the knowledge collected can be organised, manipulated and interrogated. We have characterised the specification as a knowledge base, which implies techniques from knowledge-based systems research can be applied [Barstow 1987]. If all the knowledge is collected into an on-line knowledge base, it can remain accessible for the remainder of the lifecycle [Harrison 1987]. This includes not just the knowledge about the domain, but the documentation of the process itself.

We therefore envisage an environment which comprises a knowledge base containing all the gathered information, an inference engine which defines the operations which might be carried out on the knowledge base, and a set of tools which assist in the formulation, refinement and presentation of the knowledge. This organisation is based on the typical architecture of a knowledge based system [Boose 1986], and has been applied to requirements engineering elsewhere (e.g. Reubenstein & Waters [1989]). The form of the inference engine will depend on the representation(s) used within the knowledge base. As the knowledge base represents at any point the current state of the specification, reasoning within the knowledge base allows participants to test the specification.

The knowledge base will be continually added to, and hence any reasoning is non-monotonic in that new knowledge may invalidate previous conclusions. The incremental refinement of descriptions inevitably involves adding details such as exceptional case behaviour, to fix problems which occur when descriptions are tested. Detailed tracing and recording of dependencies throughout the knowledge base is therefore desirable.

As a specification evolves, it will frequently become inconsistent, and at all times will be (to some degree) incomplete [Yue 1987]. At times there will be temporary inconsistencies, over-generalisations, and over-simplifications. However, participants will need to manipulate the specification as it evolves, as part of the exploratory process, and so the reasoning mechanisms must cope with inconsistency and incompleteness. Areas of conflict, and places where more details are needed can often be detected automatically, but the need to allow commitments to be delayed means that participants might choose not to resolve these immediately.

Finally, the entire process should be documented automatically. We have stressed the importance of capturing the design history, together with rationales. However, to do so requires a lot of extra effort from the participants. They are unlikely to be persuaded to make this effort unless a great deal of the recording process is automatic. If the series of actions made using automatic tools is recorded, this can form a framework to which rationales can be attached.

2.3.3 Tools

The last section described the general nature of a support environment needed to support requirements engineering. There are a number of areas in which tool support can be of particular help.

One major task in which tool support can help is in guiding participants to areas which need more discussion. We noted above that automatic detection of conflicts and missing information should be possible. Given enough background knowledge about the domain, it should also be possible to provide a degree of knowledge based critiquing [Fickas & Nagarajan 1988], to supplement the manual critiquing process. Maarek & Berry [1989] note that automating the clerical work of detailed checking of specifications is an ideal way to supplement the human activity.

The critiquing process will lead to more knowledge being gathered. Tools can assist with the incremental integration of this new information with the existing knowledge base. The translation of natural language utterances into the appropriate representations is unlikely to be automated, but again, clerical assistance can be given. Feedback can be given regarding the effect of the new knowledge, for example by tracing the effects of new cases. Where the new information was prompted by problems in the existing knowledge base, the system can keep track of which parts have been resolved, how they were resolved, and what problems still remain.

One advantage of automatically tracing the process is that the context of statements can be more readily accessed. Contextual information provides important clues for interpretation and again for validation. If the dialogues are recorded and held as a part of the knowledge base, then tools to access these transcripts can be provided.

Conflict is an important part of the specification process, and tools to help identify and resolve conflicts are needed. Whilst resolving conflict is essentially a human activity, a range of options needs to be created and explored. Assistance with developing and reasoning with the options can be provided. The system should also ensure that all relevant views are represented in the resolution process.

Finally, tools are needed for organising the knowledge for presentation back to the participants. This includes assistance with building initial descriptions from the participant's comments, and assistance with demonstrating to the participants the current state of the specification. Several techniques are useful for this, including animation of the specification [Kramer *et. al.* 1987] and summarization [Fickas 1987a].

2.4 Summary

This chapter has examined the importance of requirements engineering, and the difficulties, concluding that a model is needed to support the process. Requirements engineering is important because it is concerned with the production of specifications, which play a pivotal role in a software engineering project. The specification acts as a communication medium amongst the software team, and as a yardstick by which results of the later stages of development will be judged. The specification should contain all the information about the requirements needed during the remainder of the software lifecycle. It must be unambiguous, testable, and modifiable. It must be representative of the many people whose needs it refers to. Above all, it should be a precise description of the requirements.

Specifications need to be carefully constructed so that they are useful and usable. In particular, we have suggested that they be treated as designed artefacts, and careful consideration given to the role they must fill. The design process that creates specifications needs to be recorded for validation, to

allow traceability, and this design history must include rationale. As there is an inevitable amount of uncertainty in requirements, exploratory approaches must be supported.

There are a number of difficulties in requirements engineering. A very large amount of knowledge needs to be captured, covering a range of areas. A range of sources need to be consulted to elicit this knowledge, including people with different backgrounds and different perspectives, together with various texts and similar media. The knowledge might be represented in a number of different ways, and may be entangled with personal preferences and biases. Specification involves negotiation between the many participants, and conflict resolution, where there are competing needs and constraints.

All these issues point toward the need for a model of the requirements engineering process. Section 2.3 presented a number of objectives for such a model. The model should provide support for the interactions between analyst and client, and encourage them as explorations of the current state of the specification. It is vital that during this process the participants should be in control: the method must only guide, rather than force, the order of discussion.

Computerized support can assist this process in two main ways: documenting the information already gained, and guiding the discussions to areas which need more exploration. The support should form a knowledge management environment capable of accommodating knowledge from many conflicting sources, which is able to reason when inconsistencies aren't resolved immediately, and which can guide and document the process throughout.

The remainder of this thesis describes a model which meets these objectives. The next chapter explores the literature describing areas from which the elements of this model are drawn, and the following chapter introduces the model itself.

3 Analytical Review

This chapter analyses the literature in a number of fields. We begin by surveying existing work in requirements engineering (§3.1) and knowledge acquisition (§3.2). Section 3.3 reviews these two fields and concludes that most existing techniques concentrate on representing a single perspective. Some recent work that recognises this shortcoming is presented. Part of the problem is that techniques are needed to compare perspectives and resolve conflicts between them. The remainder of the chapter surveys work from related fields that study aspects of conflict resolution.

3.1 Requirements Engineering

In general, software engineering has two main focuses: representation schemes and development frameworks [Finkelstein *et. al.* 1987]. Research in requirements engineering has also fallen into one or other of these areas. The former generally involves developing specification languages, together with some investigation into the desirable features of such languages. Research on development frameworks has been more varied, and a number of paradigms have been proposed for modelling the requirements process. This section discusses the lifecycle models used in software engineering, and then surveys research on specification languages and on modelling the requirements process.

3.1.1 Software Life-Cycle

Several paradigms for the development of software have been proposed, of which the most commonly used for large systems is the waterfall model, which breaks the process up into a series of phases. This aids management of the project as each phase can be “signed off” as complete with the production of one or more artefacts, such as specification, design, code, etc.

The influence of the waterfall model has encouraged practitioners to think of the requirements phase as a separable part of the process, which can be treated in isolation from the remainder of the software development process. The interface between this and the later phases is the requirements specification, which should be a complete and precise description of the requirements. The requirements phase is then restricted to a consideration of *what* the system should do, while the later, design stage considers *how* it should be done.

In fact, this division is not realistic, and even the waterfall model recognises there is a process of feedback from the later phases. It has been pointed out that without perfect foresight, the specification itself is unlikely to remain unchanged by the design process [Swartout & Balzer 1982]. Such foresight is unlikely as the introduction of a software system actually changes both the domain and human perception of it.

Lehman [1990] suggests that uncertainty is a direct consequence of nature, and that software engineering is really an attempt to manage that uncertainty. To illustrate the point, he introduces a classification of three types of program: S-type (specifiable) in which the specification is the sole, definitive determinant of correctness; P-type, which are created to solve some stated problem, and success is judged according to the problem statement; and E-type (embedded) which solve a problem or implement an application in some real world domain, and are judged according to acceptability, value and level of satisfaction. Large scale systems are nearly always E-type, and therefore the notion of correctness does not apply [Lehman 1980].

The waterfall model is inadequate for real world applications as it doesn't make allowance for program evolution. Giddings [1984] proposes an alternative lifecycle modelled on a cycle of experimentation. He introduces a distinction similar to Lehman's, between software which is independent of its domain and so does not need to be validated, and software which is dependant on the domain, and which may change that domain. He introduces a lifecycle which is really a loop of observation, abstraction, design, implementation and experimentation. The results of the experimentation feed back into the next iteration. Products would be treated as spin-offs from this cycle, rather than an end in themselves, allowing a distinction between error correction (for a product) and evolution (for the continuing cycle).

A number of other paradigms for software development have been investigated, including exploratory programming, rapid prototyping, and formal transformation. However, in all these models, an understanding of the requirements is an essential initial step, whether or not these are eventually expressed in a specification. An examination of the various attempts to support the requirements process is therefore of value no matter which lifecycle paradigm is adopted.

3.1.2 Specification Languages

Research on specification languages has emphasised the distinction between *requirements* (what is to be done) and *design* (how it is to be done). Typically, specification languages are designed to assist with modelling the requirements without prejudicing future design decisions. Balzer & Goldman [1979] describe a number of principles for specifications, and require that a specification be unambiguous, testable and modifiable. Their principles include: that the functionality should be separated from the implementation; that a specification language should be process-oriented rather than mathematical; that a specification should encompass not only the system of which the software is a component, but the environment of that system too; and that specifications should be operational. Their requirement for modifiability leads them to suggest that specifications should be loosely structured, and localised.

Borgida, Greenspan & Mylopoulos [1985] also give a number of principles for specification languages. A specification language should be able to: model entities and events in the domain, together with constraints and assumptions; handle real world concepts (natural kinds) without causing vagueness or contradiction; and represent the passing of time. Furthermore, it should support the process of abstraction, and provide the facility to detect inconsistency, preferably by allowing redundancy. Finally specification languages ought to be easy to learn and read, and convenient to use.

Whilst there is broad agreement on the general thrust of these principles, there is disagreement over how they might be met. For example, there is a conflict between the desirability of localization and the suggestion that redundancy should be encouraged to assist with consistency testing. Some authors recommend that repetition of information be eliminated as it inevitably leads to inconsistency, while others maintain that it helps to detect errors and reduce univocality.

Specification languages have to provide both formality and expressive power, which tend to be complimentary features [Dubois 1989]. The expressive power supports the modelling of requirements, while the formality provides guidance for the process of this modelling. Current research is dedicated to developing specification languages that can maximise both these properties, and there is some disagreement between those that advocate purely formal specification (e.g. Cunningham *et. al.* [1985]), and those that argue for a degree of informality in specifications (e.g. Balzer, Goldman & Wile [1978]).

Plenty of reasons have been suggested for adopting formal specification languages, of which the most important is the level of precision they offer. Finkelstein, Finkelstein & Maibaum [1990] discuss the advantages, suggesting that with a formal specification it should be possible: to eliminate ambiguity using the formal semantics; to test the specification for incompleteness and

inconsistency; to formally verify a subsequent design or implementation; to trace back components of the specification to originating needs for validation purposes; and to evolve and modify the specification systematically in later stages of development.

On the other hand there are a number of drawbacks with formal specification languages, not least in the unnatural notations they employ. Cunningham *et. al.* [1985] give some reasons why the current generation of formal specification languages have not been adopted in industry. These include unnaturalness, unreadability, fluidity (the languages are still being developed), lack of structure, and contentious semantics. Above all, formal specification languages need to be supported with formal methods for guiding and organising the specification activity.

Early specification languages were intended primarily for program specifications, and were designed to model the things programmers are concerned with. Of these, the algebraic language CLEAR is a typical example [Burstall & Goguen 1981]. Such languages permit the specification of data structures and operations upon them, in terms of functionality, rather than implementation. This allows the designers to reason about the properties of various programming concepts, and provides a basis for proving that particular implementations satisfy the specified behaviour.

Addis [1985] notes that although representation schemes are a first step towards symbol manipulation, the representation must not be tied too closely to the physical features of the symbol manipulation system, or the descriptions will assume properties that have nothing to do with the knowledge domain. The initial analysis should use representations that closely match the mental models used by the participants. Recent specification languages aimed at modeling the requirements recognise this. A number of such languages have been developed, based on a number of different paradigms.

The object-oriented language called RML draws heavily on AI work on knowledge representation [Borgida, Greenspan & Mylopoulos 1985]. All concepts in the world are represented as objects, which are grouped together in class hierarchies, and related to one another by their properties. The class hierarchies provide inheritance, and are the key mechanism for handling abstraction. Activities and assertions are also treated as objects, allowing a uniform treatment of all concepts. The language also represents units of time as objects, and evaluation of an object's properties is always with reference to a particular point in time.

The relational database model provides a different paradigm for requirements modeling, as demonstrated by the language GIST [London & Feather 1982]. GIST is based on the description of data objects and their relations to one another, allowing associative reference to objects, including historical reference. One of the motivations for GIST was to provide a formal semantics for the features normally found in natural language specifications, such as constraints, inference, and demons (which model a system's environment). It is also a wide spectrum language, designed to support incremental transformation from requirements specification to detailed design.

A third paradigm which has been used for specification languages is logic. Maibaum [1986] describes a modal action logic (MAL) for formal specification. MAL is based on a typed first order logic, with the addition of: a modal logic with agents and actions; deontic operators such as *permitted* and *obliged*; and an interval logic for reasoning about time. Hence it provides a framework for modeling real-world concepts using a formal logic.

3.1.3 Specification Processes

Rich specification languages alone are not enough to support the specification process [Kramer, Ng & Potts 1987]. Specifications of large scale systems are themselves large and unwieldy, and constructing and comprehending such specifications is difficult [Feather 1987]. Furthermore, many specification languages, particularly the more formal, require a great deal of expertise. The *process* of specification must be supported, with guidance given for the elicitation and formulation

of requirements. Where possible this guidance should be formalised, to allow a degree of automation.

Osterweil [1987] argues that software development should be treated in the same rigorous way that software itself is handled, and that software processes could be encoded in the same way as programs. The advantages of this are that the process becomes a material entity, allowing closer examination of it, and possible re-use. Certainly such a formalisation is superior to the provision of procedures manuals.

However, Lehman [1987] cautions that representing software development as a program has its limitations, particularly as any programming language severely limits how a problem can be solved. As many parts of software development involve some form of creativity, they are not amenable to being modelled algorithmically. However, this does not preclude a form of heuristic guidance, and an accurate record of the decision-making process is certainly of vital importance.

Empirical investigations of analysts at work reveal the range of behaviours which need to be supported. For example, Fickas, Collins & Olivier [1987] studied analysts interviewing clients, and observed a number of techniques including tutoring, concern mitigation, example generation, exploration of automation impacts, and summarization. The customer's needs must be questioned in detail rather than accepted at face value, and an important part of this process is the construction of preliminary models for demonstration back to the client.

There are a number of paradigms currently being investigated as models for the specification process. Some of these concentrate on the creation of specifications, while others concentrate on the manipulation of specifications and their subsequent development. However, no clear distinction can be drawn between these two focuses, as there is no clear distinction between creation and development of specifications. Often, paradigms chosen for their applicability to one area of the specification process can be usefully extended to other areas. In the remainder of this section, we survey some of these paradigms, beginning with a discussion of the use of formal methods, and ending with some comments on the difficulty of capturing and recording the process.

3.1.3.1 Formal Methods

Formal specification languages provide a way to remove ambiguity, by defining a precise semantics. However, the full potential of formal specification languages can only be exploited with formal methods. Bjorner [1987] defines a method as a set of guidelines for selecting and sequencing the use of techniques and tools, in order to construct an artefact. A formal method is a method in which all the tools and techniques are formal, and in which the use of all the techniques and tools can be formally justified. The second part of this definition provides the chief characteristic of formal methods: the process of development is treated as a formal object which can be reasoned about in the same way as the artefacts produced by that process. As yet no existing methods satisfy this definition, although Bjorner discusses how some existing methods such as VDM might be formalised.

Cunningham *et al.* [1985] argue for the use of formal methods, pointing out that formal languages require prescriptive guidance (methods), if they are to be applied by anyone other than highly skilled experts to significantly large problems. They also point out that while formal specification methods may be very different from existing, informal methods, they should at least be compatible with current practices for procurement and quality assurance, and should not stultify creativity. Finally, formal methods should support requirements elicitation from many sources and the consolidation of these perspectives into a consistent specification, something that few existing methods do.

Finkelstein & Potts [1987] describe a method designed to support the construction of formal specifications in the language MAL (See §3.1.2). The method consists of a set of steps, each of

which consists of a strategy and a set of heuristics to cover specific situations. While not strictly formal according to Bjorner's definition, the name, Structured Common Sense (SCS) was chosen to convey its nature as a formal framework with which to guide the analyst's skills. SCS draws on the experience of established methods such as CORE [Systems Designers, 1985] in attempting to provide a prescriptive framework.

There is some controversy over the extent to which formal methods can contribute to software engineering. Most of the objections centre on the use of program proofs (e.g. [DeMillo, Lipton & Perlis 1977], [Fetzer 1988]), as these are probably the most widely advertised benefit of formal methods. The objections centre around the use of rigorous proofs to assure software reliability, and whether real-world software is actually amenable to proof of correctness. However, there is little doubt that formal methods have a lot to offer for the practice of software engineering, even if they are only ever used to supplement traditional informal methods [Gehani 1982].

3.1.3.2 Specification as Planning

Anderson & Fickas [1989] suggest that requirements can be regarded as goals, and the specification describes plans for achieving them. Work on planning from AI can then be applied as a model to guide the design of specifications. Using this paradigm, the expert knowledge used in the specification process is encoded as operators, and the objects and relations used by those operators, and a planning system is used to support the specification process. The planner produces plans using the given operators to show how the users' goals might be achieved. Plans can also be constructed to prove that prohibited states can be achieved, in order to debug and elaborate a partially developed specification.

The need to model and reason about the users' goals has been studied in a number of projects. Mostow & Voigt [1987] explore the use of planning in design, and although they studied algorithm design, some of their comments reflect a wider concern, especially of the specification process. They distinguish three types of goal: domain goals, which describe the task; performance goals, which roughly equate to non-functional requirements; and design process goals, which concern the resources available for design. All these forms of goals must be accounted for in the requirements specification.

Wile has developed a language called Paddle in which strategies and goal structures can be expressed [Wile 1982]. Although it is intended for transformations on programming languages, Wile points out that Paddle would be useful for such domains as theorem proving and specification design. The language encompasses individual transformations (operators), editors (which apply sequences of transformations), and strategies (which represent the intent, or plan, behind such sequences). In other words, it describes both goals *and* ways of achieving them.

One of the interesting features of Paddle is that it explicitly expresses relationships between goals, among which Wile includes: sequential dependency, where some goal must be achieved before another; goal independence, where goals can be achieved in parallel; choice, where one of a set of goals should be selected, all of which support the same overall goal; conditional goals, where some goal need only be achieved if another goal fails; and repetition. Paddle includes all these goal structures, together with variations, as built-in primitives. It allows the user to define *commands*, which might be single operations or entire strategies, using a mixture of primitive keywords and English phrases. The English phrases are not understood by the system, but rather act as stubs, which must be refined later by further definitions. A set of operators which describe particular ways of achieving goals is provided as a parameter to Paddle. Typically these would form an editing language, as when Paddle is used to structure sets of transformations on programs.

Treating specification as a planning process has certain advantages. It makes the goals (or purposes) of a specification explicit [Mostow 1985], and so aids the validation process. Yue [1987] points out that information about goals is vital to demonstrate completeness, and suggests

that *sufficiency* and *pertinence* can be used as formal measures of completeness with regard to a particular goal. However, while planning based approaches allow the analyst to model the goals of the users, it is not clear how easily these goals can be elicited.

3.1.3.3 Analogy and Reuse

It has long been recognised that one way of solving problems is by analogy with old ones [Leishman 1988]. In some cases an identical problem will have already been solved, and the solution can be re-used without alteration. Some effort in software engineering has been directed at re-using chunks of programs, and at building and indexing libraries of reusable software components (e.g. Neighbors [1984]). However, at anything other than the lowest level there is unlikely to be an identical past solution, and the best that can be hoped for is to find a similar solution which can be adapted.

Three problems need to be addressed for analogical problem solving to proceed: how experiences can be stored and indexed; how the closest past solution can be selected from a potentially huge list; and how the old solution can be transformed for the new problem. Finkelstein [1987] explores the possibility of re-using parts of specifications through a process of analogical mapping, and gives a number of strategies for selecting analogous solutions. Pattern matching on attributes can be used at both a high and low level of abstraction, and can be enhanced with the use of importance measures. Causal chain matching attempts to find similar causal links between objects. Where concepts can be generalised and so organised into classes, this provides another source of analogies. Finally the purposes of objects can also be used to provide information on possible analogies.

Carbonell [1985] criticises attempts to take a solution to an old problem and transform it into a solution to the new problem as being too simplistic, because it ignores information about strategy and decision-making. He proposes *derivational* analogy, which walks through the decisions used in an old problem and considers whether they are still valid for the new one. The derivation must capture the goal-structure of the problem, all the decisions made, pointers to knowledge used in the solution and the solution itself. Each decision must include all the alternatives that were considered, the reasons used for the decision, whether any false paths were followed and why (and if so, what the eventual cause of failure was), and any dependencies of later decisions on earlier ones.

A derivation can be re-used when the initial stages of problem analysis for a new problem are similar. At each step, the reasoning for the decision recorded in the derivation is examined to ensure it is still valid. Storing derivations is memory intensive, although several factors mitigate the problem. Firstly, all the dependency links used in a derivation are internal and so each derivation is self-contained. The size of a derivation is proportional to the search depth, rather than the number of decisions made, as only the final (successful) path need be recorded. Furthermore, problems can share portions of derivation, which also aids indexing.

Replay of design derivations has not yet been particularly successful. Mostow [1986] lists the problems that can arise. Missing preconditions can cause a step to be replayed when it shouldn't be: it doesn't work, it produces the wrong result, it doesn't achieve what it was originally supposed to, or an alternative step would be more appropriate. The rationale for the step must be re-examined, and the goal tested to see if it has been achieved correctly. There are also problems with references, as a reference to an object in the original derivation may not be correctly resolved in the new problem. Clearly, higher-level descriptions of the objects in question must be used to understand what the reference was intended to match.

Problems can arise in patching a derivation to meet a new problem. Although adding, deleting and modifying steps can allow the derivation to be used on a much larger set of problems, there is a problem in identifying exactly which steps need to be altered. Furthermore, altering steps may cause problems elsewhere in the derivation, as steps make implicit assumptions about the context

in which they are used. They rely on their predecessors to set up this context, and their successors to complete the process successfully.

Ultimately, the intention to allow reuse must be built in to the method, and extra information stored with previous cases to facilitate identification of analogical cases [Finkelstein 1987]. Some methods have been deliberately designed with this in mind. For example, the incentive for the strategy language Paddle was the observation that developments expressed formally could be automatically re-applied to a changed specification to produce an appropriate re-implementation. Analogy with existing specifications can provide an important resource in the construction of new specifications, while replay of developments can simplify the design process.

3.1.3.4 Transformation and Elaboration

A typical use of formal specifications is to enable transformation into an implementation. In order to reason about program development, it can be regarded as a series of correctness-preserving transformations. Transformational systems generally employ a wide spectrum language in which implementation becomes a process of optimization [Wile 1982]. Typically, the human designer selects appropriate transformations which can then be carried out automatically. As the implementation is derived directly from the specification, its compliance with the specification can be proved. However, the transformational approach usually assumes that libraries of valid transformations can be developed, which may not be possible, particularly as many of the transformations will be domain dependant.

The process of specification itself can also be regarded as a series of transformations, from an initial abstraction to a detailed specification. The individual transformations either add more detail, or clarify existing parts. In this model, however, the transformations are not correctness-preserving, and so cannot be verified. Rather, the emphasis is on consideration of the rationale that led to particular parts of the specification, and hence the validation of those parts with the original needs. Feather [1987] demonstrates this approach to specification development, showing how reconsideration of an over-simplification leads to a more detailed specification. There is evidence that this is how human designers work, beginning with a simplified model which ignores inaccuracies and over-simplifications, and then gradually elaborating the details [Adelson & Soloway 1986].

Goguen [1981] points out that there are a number of limitations on the use of transformational approaches. Firstly, if transformation is the only mechanism used to develop the specification, there must be a simple starting point which can be written down directly. There are a large class of problems for which there is no simple formulation. Also, transformations apply to parts of a specification, and so in a large complex specification many checks are needed for side-effects in other parts.

3.1.3.5 Capturing Rationale

One of the problems in software engineering is how to capture the development structure. A record of the development process would facilitate maintenance. More importantly, if programs are to be allowed to evolve, the rationale that led to the current design must be accessible. One way of managing program evolution without causing specification and design to diverge is to make the necessary alterations to the specification, and then replay the design process to derive the next version [Green *et. al.* 1983]. Like derivational analogy, replay depends critically on the ability to record and represent the development history in all its detail. This includes capturing the structure of any decisions and the goals which motivated them, together with the inter-relationships of those goals.

Conklin & Richter [1985] argue that current methodologies are artifact-oriented, while critical maintenance information, which once existed as process information is discarded. This includes

understanding, problem formulation, and rationales. For example, the waterfall model concentrates on distinct documents produced at each stage, while the processes are not described. Most specifications describe what must be done, but not why.

In order to represent rationales, a model is needed to structure them. Kaplan [1989] points out that what may be considered an important decision by one person might be a trivial assumption to another. Hence rationales tend to be idiosyncratic, which complicates the problem further. Kaplan's approach is to attempt to record decisions indirectly, by recording and labelling utterances. He points out that alternative approaches, such as the use of formalisms, or the attempt to model the design process with an expert system are unlikely to capture all the decisions.

By contrast, Conklin [1989] describes a more direct attempt at modelling decisions, as part of the Design Journal project (See §3.3.3). The early exploratory stage of the design process can be supported with an Issue-Based Information System (IBIS), in which participants suggest *issues* which need to be addressed. Other participants can then attach *positions* which describe ways of tackling an issue, and *arguments*, which support or refute a position. The result is a hypertext network recording the rationales being used by the participants.

3.2 Knowledge Acquisition

Knowledge acquisition and requirements engineering share many similarities. Both represent the initial stage of an engineering process, and both are concerned with the elicitation and formulation of knowledge which will be used to design and build a software system. This section reviews work in knowledge acquisition which is relevant to the requirements process.

3.2.1 Building Knowledge-Based Systems

The field of knowledge acquisition grew out of recognition that the hardest part of building knowledge-based systems is gathering and representing the knowledge. Boose [1986] defines a knowledge-based system as a system which solves problems using explicit symbolic knowledge which is kept separate from the reasoning mechanisms. Typically, such systems are built using a rapid prototyping paradigm, using an expert in the domain to help refine them iteratively.

Some interactive systems have been built to help with this refinement process. For example, TEIRESIAS [Davis 1979] engages the expert in a dialogue, accepting knowledge to add to a knowledge base and guiding the process by indicating how new rules affect the state of the system. The tool makes extensive use of meta-knowledge to reason about the knowledge contained in the system. However, because it is geared to a rule based system, and does not attempt to derive or refine its own rules, it requires immense co-operation from the expert. The ADVISE system [Michalski & Baskin 1983] performs a similar role, but makes use of multiple representations (three are implemented: a rule-base; a conceptual network; and a relational database). It also includes tools for inductive inference on sets of examples.

Rychener describes a number of projects which attempt to build what he terms an instructable production system, to acquire knowledge from a number of sources [Rychener, 1980]. Several functional components were coupled together to form a kernel capable of adding new production rules to the rule-base. The key feature of these experiments was interaction with experts via mixed initiative dialogue. In other words the system has to form a strong idea of what it needs to know, and plan how to obtain the information. This contrasts with the passive expert system tools which require more work from the expert. A number of ideas were used to guide the acquisition process, including simple means-ends analysis, analogy, search spaces, schemas and semantic nets, although no attempt was made to integrate these in a single system.

More recent interactive tools provide more structure to assist the expert. These are either based on a theory of the nature of the knowledge being acquired, (e.g. KSS0, which is based on repertory grids [Shaw & Gaines 1987]), or are domain dependant and designed to elicit particular types of knowledge, (e.g. SALT [Marcus 1987]). Boose [1989] gives a comprehensive overview of such tools.

3.2.2 Machine Learning

Knowledge acquisition is also generally taken to include learning systems, which derive their own knowledge, through a process of abstraction from cases or sets of examples. Shalin *et. al.* [1988] survey learning systems, pointing out that there are four ways in which learning systems can facilitate knowledge acquisition: deriving an initial set of rules for a system; refining existing knowledge bases; adapting an existing system to fit the user's expertise or style; and finally, the study of learning systems might lead to a principled method for constructing knowledge bases.

Research on machine learning generally falls into three types: *task oriented*, based on a particular application; *cognitive simulation*, which attempts to model human learning processes; and *theoretical analysis* which looks at the foundations of learning methods. A number of useful learning strategies have been shown to work well in limited areas [Dietterich & Michalski, 1983]. These include rote learning, instruction, deduction, genetic algorithms, analogy, induction, and observational discovery. Most effort in machine learning has been devoted to learning from examples, as this represents the most tractable area for empirical study. However, analogical learning is also producing some interesting results [Leishman 1989].

The importance of machine learning to knowledge acquisition has been emphasized in a number of experiments comparing learned knowledge with that elicited from domain experts. For instance, Michalski and Chilausky conducted a series of experiments on diagnosis rules, and found that rules derived inductively from a set of examples consistently outperformed those suggested by the experts, even when the experts' rules were put through extensive debugging procedures [Michalski and Chilausky 1980].

It is frequently claimed that machine learning can relieve the "knowledge acquisition bottle-neck" of AI systems [Simon 1983]. However, there have been few attempts to incorporate learning into larger AI systems (See Hayes Roth & Hewett [1985] for a notable exception). Learning systems have drawbacks in that changing systems become opaque and the learned information is conjectural [Michalski 1986]. Empirical studies have shown that any structure that is initially present in a system deteriorates as the system evolves, unless strenuous measures are taken to maintain the structure [Lehman 1980]. Where that evolution is due to an automatic learning element, the problem is compounded, as the evolution is invisible. Michie [1982] has also pointed out that the knowledge derived by a learning system (usually in the form of rules) is not easily understood by humans. If human experts do not find anything familiar in the rules produced by a learning system, then the use of machine learning to supplement knowledge acquisition can be counter-productive.

3.2.3 Eliciting Conceptual Models

The greatest contribution of knowledge acquisition is in the techniques it provides for eliciting and describing the mental models used by people. Norman [1986] describes the role of such models in the design process. The designer has a conceptual model of the system to be built, whilst users of a system build mental models of how that system works: if these conceptual models differ greatly, then the user is likely to have problems using the system. The primary job of the designer is therefore to build the system in such a way that her conceptual model is correctly conveyed to the user, and as Norman points out, the main way that the design model is conveyed is through the user interface. In order to ensure the image the system projects is an accurate portrayal of the design model, it must be clear what that design model is: in other words the mental models used in the design process must be made explicit.

Shaw & Woodward [1989] characterise the series of models constructed in the attempt to capture a person's knowledge. They distinguish between the expert's internalised model (the mental model); her first attempts to articulate that model (the conceptual model); and the received or developed model built through interaction with the knowledge engineer (the model of the conceptual model). To complicate matters further, the expert's mental model might not be a true reflection of the processes involved in applying her expertise.

There are a number of ways of gathering knowledge in the process of building these models. Johnson *et. al.* [1988] classify the main methods used as follows: structured interviews and questionnaires; observational techniques; concurrent and retrospective protocols; and experimental techniques, for example the use of repertory grids. There are also a host of other miscellaneous techniques that can be incorporated into those above, including group discussion, instruction, third party commentaries, and brainstorming [Welbank 1983].

Of these methods, the most commonly used are those which provide verbal data, which must then be processed in some way. Wielinga & Breuker [1984] list some of the problems with collecting verbal data: there is no way of detecting whether the expert has omitted anything; the knowledge may be hard to express verbally; the knowledge may not be accessible by introspection; the expert makes implicit assumptions about the level of knowledge of the analyst; experts may not be motivated to reveal their inner thoughts; and most experts have little experience in verbalising their thoughts.

An alternative to handling verbal data is to make use of "mediating representations", which mediate between verbal data and operational representations [Johnson, 1989]. Such representations are intended to assist in the early stages of conceptualization. Johnson lists the desirable features of such representations as: expressiveness, economy, and communicability. Furthermore, mediating representations provide a kind of high-level specification of the knowledge used in a system, and as such are analogous to specification languages.

In summary, many of the problems in knowledge acquisition remain to be solved, while the successful tools and techniques are still limited in scope. Shaw & Gaines [1989] provide a good overview of the field, and discuss some future trends. In particular, they predict a growing integration with related fields such as text analysis, hypermedia, and software engineering.

3.2.4 Specification as Knowledge Acquisition

Specification can be regarded as a knowledge acquisition task [Easterbrook 1989]. From this perspective, the requirements specification is treated as a knowledge base, which can then be used as a key resource to support the remainder of the software lifecycle. Requirements engineering is concerned with the creation and refinement of this knowledge base. Given this view of specification, support for the process can borrow heavily from work on knowledge acquisition.

Work on knowledge acquisition has provided a number of interactive tools for the elicitation process [Boose 1990], which can be adapted for use in requirements engineering. For example, Reubenstein & Waters [1989] describe a system called the Requirements Apprentice (RA), which assists the analyst in refining an initial, informal specification. RA is a knowledge-based system with three components: a cliché library which contains the domain knowledge; an inference engine called Cake, which handles propositional deduction and maintenance of dependencies [Rich 1985]; and an interface, which provides the main functionality of the RA. The system provides assistance in three main ways. The interactive output informs the analyst of any conclusions and inconsistencies of entered information. The knowledge base built up during the process can be used by other tools throughout the lifecycle. Finally, the system can produce various written documents summarising the state of the requirements knowledge base, for validation.

Another aspect of knowledge acquisition is the processing of verbal protocols and transcripts of interviews. Maarek & Barry [1989] describe a tool for identifying abstractions from interview transcripts using lexical affinities. They envisage an eventual organisation of the knowledge as a network of nodes containing both formalised abstractions and textual descriptions, with hypertextual links. The initial tool they have developed assists in the process of building this network by detecting repeated phrases in order to identify the key concepts. This tool shares many features with similar tools developed in knowledge acquisition (See for example Woodward [1988]). There have also been attempts to apply natural language understanding to the requirements phase [Balzer, Goldman & Wile 1978].

Johnson, Johnson & Russell [1988] use a knowledge based approach for task analysis. They treat tasks as concepts, using a frame-like representation to record associated information. A set of tasks has structure, in that certain tasks or elements of tasks are more likely to co-occur than others. Also, tasks differ in importance and representativeness. A framework for eliciting knowledge about tasks is described, which draws extensively on knowledge gathering techniques from knowledge acquisition.

3.3 Critical Analysis

We have surveyed a number of current methods for requirements engineering and knowledge acquisition. We divided work on requirements engineering into specification languages and specification processes, although clearly there is much interdependence and cross-over between the two areas. There is also much lively discussion over the utility of formal methods. Work on knowledge acquisition is closely related to requirements engineering, and has drawn on many other fields in the attempt to develop methodologies for the capture of knowledge.

3.3.1 The Single Viewpoint Bias

All the methods surveyed so far share a shortcoming: They all concentrate on the development of a single description of the item under study, whether it is a requirements specification, a system model, a domain model, or a cognitive model. This single description can only represent a single viewpoint, resulting in a “univocality” that has been criticised by several authors (see for instance Cunningham *et. al.* [1985], Shaw & Gaines [1989]). There is still little or no support available for developing and maintaining alternative descriptions from the various suggestions, nor for resolving the conflict inherent in the process.

Advice for those building expert systems usually includes that the problem being tackled should be solvable by a single expert. This is because no methodology yet exists for combining expertise from multiple experts [Boose 1986]. While there are some techniques available for eliciting a consensus opinion from a group of experts, most notably the Delphi technique, these are very limited in scope, and only produce very general responses. Psychological phenomena such as “group-think” indicate that consensus may not produce the highest quality decisions [Meyer & Booker 1989]. Furthermore, there are some fundamental conflicts which cannot be handled with consensus.

Backhouse [1988] makes more extreme criticisms of existing analysis techniques. He criticizes the preoccupation with procedure and information plumbing rather than the actual business activities saying that more attention should be paid to the agents involved in business and the social obligations entailed by their communication. He also criticizes the descriptions used (e.g. Entity-Attribute-Relation models) for being too arbitrary. He proposes a new approach which removes the assumption of a single object reality, and introduces the idea of business as a co-operative activity between agents. Winograd & Flores [1986] amplify this theme, and argue that communication is central to human activities, and that the role of computers should be to support co-ordination among people.

We discussed some of the consequences of suppressing or avoiding conflict in §2.2.4. However, we have not yet defined exactly what we mean by conflict. In its broadest sense, the term *conflict* covers any interference in one party's activities, needs or goals, caused by the activities of another party. The situation can be characterised as a disparity between the goals of the parties, although this would not be considered conflict until the effects are felt. This use is at odds with the more restricted use in political science as the opposite to co-operation as a mode of interaction. It corresponds more closely to its use in organisational behaviour, in which "conflict" is used to describe any situation in which the actions of one party interfere with those of another. Conflicts do not necessarily need to be resolved, but where they are, the possible resolution methods range from the co-operative to the non-cooperative.

The key to allowing multiple viewpoints to co-exist is to delay making a commitment to any one of them [Thimbleby 1988]. If one viewpoint or a particular combination of viewpoints is adopted as the primary one, and the specification based on that viewpoint, other useful viewpoints become neglected. A decision has been made (however subconsciously), and a commitment made to the chosen viewpoint. Such a commitment can overly restrict the future course of the design process. If that commitment can be avoided, all viewpoints can continue to develop, and the interaction between them can provide useful feedback on the consequences of those viewpoints.

To support the interaction of viewpoints, we need to model them explicitly. Each participant brings a number of preconceptions or biases into the software specification process, together with a certain amount of knowledge. The areas of knowledge do not fit together like a jigsaw, but instead overlap in some places, conflict in others, and often leave gaps. A set of descriptions representing the various viewpoints will not form a complete picture of the domain of interest, but can provide a fuller picture than a single description. The most important areas of the domain will become highlighted as those mentioned most often, and over which there will be most conflict.

3.3.2 Conflict between Experts

Conventional knowledge-based systems rely on consistent knowledge for their inference mechanisms to work. In many cases, this is achieved by only consulting a single expert. Where inconsistencies are detected, they are attributed to mistakes in the acquisition process, or the result of biases [Cleaves 1988], and are eliminated through a lengthy process of interactive debugging and refinement with the expert [Davis 1979]. The expert can be persuaded to participate in this process in order to appear consistent. This insistence that expertise must be consistent and rational imposes restrictions on the knowledge acquired. The knowledge acquisition process then becomes not so much the modelling of the expert's behaviour, but the synthesis of a domain model which need not resemble any mental model used by the expert [Shaw & Woodward 1989]. In this way conflicts in the expert's knowledge can be filtered out.

When further participants are involved, the problems of conflict cannot be avoided so conveniently. Although the same process of rationalisation can be undertaken with a group, there is little pressure on groups of experts to agree with one another, and the synthesis of a consistent domain model can be very difficult. One approach (see §3.4.4.1) is to keep the contributions separate, and apply them as separate reasoning systems [Nii 1986b]. Again, this is merely a form of conflict avoidance, with no means of combining the knowledge automatically. Instead, the problem is reduced to that of deciding which rule should be selected when several are applicable, which can be satisfactorily dealt with using a set of heuristics, as the possible combinations are foreseeable.

One notable attempt to study conflict between experts is that of Shaw & Gaines [1988]. In comparing the Entity-Attribute models of different experts, Shaw identifies four types of comparisons between conceptual systems:

Consensus - experts use the same terminology to describe the same concepts;

Correspondence - experts use different terminology to describe the same concepts;

Conflict - experts use the same terminology to describe different concepts;

Contrast - experts use different terminology to describe different concepts;

Each of these situations can be useful in capturing different perspectives, and in particular, the availability of alternative terminologies makes a knowledge-base more accessible.

Whilst Shaw's use of the term conflict is rather different from ours, it does highlight the way that terminological differences can obscure the deeper agreements and conflicts between people. Shaw describes a methodology for recognising each of the four situations using entity-attribute grids. However, this requires the participants to agree first on the definition of a common set of entities, and is concerned with differences in the way experts distinguish between the entities. It is not clear how this methodology might be extended to the types of representation used in systems analysis, where the establishment of an agreed set of entities is itself a problem of conflict resolution.

3.3.3 Many viewpoints

In requirements engineering there have been a number of attempts to handle the problems of multiple perspectives. For example, the distinguishing feature of the method CORE [Systems Designers 1985] is its use of *viewpoints* to give structure to the description. These represent the components of the system and its environment, and can be organizational, human, software, or hardware. In other words, every information processing entity with which the eventual system must interact, as well as those which will be partially or wholly subsumed by the system, are modelled as viewpoints. Furthermore, each viewpoint has a corresponding viewpoint authority, which is the person or machine responsible for carrying out the process described.

However, the viewpoints used in CORE provide a structure only for process knowledge. Declarative knowledge about the domain, and about the history of the analysis process itself, is excluded from this structure. Furthermore, the viewpoints are not allowed to overlap, and hence any conflict between them is avoided. The areas of authority for each viewpoint must be precisely defined during the process of identifying the viewpoints. There can be no redundancy, and hence no inconsistencies. Differences can remain in the expected interaction between viewpoints, which are ironed out in later steps of the method.

Conklin [1986] suggests that design needs many viewpoints, ideas, values, and concerns to be exchanged and argued, and proposes a structure called an ISAAC to represent design decisions. Each decision has an issue, a set of alternatives, an analysis of the alternatives, and a (possibly tentative) commitment to one of those alternatives. This work developed out of earlier studies of issue-based information systems [Conklin 1989], and forms part of a larger project on supporting design. The system is based on hypertext, and emphasises the capture of the design activities rather than just the final products of design.

Handling many viewpoints requires collaboration. Fickas [1987a] has reviewed some of the most promising approaches for supporting the co-operative process in requirements engineering. These include Gradual Elaboration [Goldman, 1982], in which a small number of types of step are available to incrementally build the specification, Parallel Development [Feather, 1987], in which partial specifications are developed separately according to different development concerns, and then merged at a later stage, and Knowledge-Based Critiquing [Fickas & Nagarajan, 1988] in which an intelligent model of the domain is used to debug a specification. Of these, the parallel development approach is the most successful at supporting co-operative work, as the merge process acts as a focus for conflict resolution, forcing the analyst to explicitly consider and document the interaction between different aspects of the specification. However, it is still not clear how the merge operations should best be carried out.

Finkelstein, Goedicke *et. al.* [1989] formalise the notion of a viewpoint. Each viewpoint represents some area of knowledge and a preferred representation for that knowledge. More specifically, a viewpoint has the following components:

- a style, which is the representation scheme used;
- an area of concern, or domain;
- a specification, which is the set of statements in the viewpoint's style describing the area of concern;
- a work plan, which describes how the specification can be changed, and any constraints on it;
- a work record, which describes how the specification developed, and its current status.

This definition of a viewpoint abstracts away from the people involved, allowing one person to have several viewpoints (as a person may have several areas of concern), and also for one viewpoint to represent several people (where people share an area of concern).

The notion of viewpoint provides a useful context in which to study conflict. As Robbins [1974] points out, many conflicts are communicational in nature. Finkelstein & Fuks [1989] use the viewpoint as a basis for a study of some of the communicational problems surrounding conflict. They describe a formal model of dialogue between two agents, which allows agents to share knowledge and detect inconsistencies between their knowledge. The dialogue takes the form of a game, in which moves consist of speech acts, such as statement, question, challenge, or withdrawal, and the rules constrain which acts are legal in which context. Agents can then query chains of reasoning made by other agents, and request information which they need to verify the conclusions. In this way, conflicts based on misunderstandings and incomplete knowledge can be detected and resolved.

Feather [1989a] proposes a specification model based on parallel elaboration of the various concerns. A basic specification is used as a point of departure for development along separate lines of concern. At some later stage the resulting specifications are merged to produce a single specification which will then reflect all the concerns. This model has the benefit of delaying the resolution of conflict between separate concerns until after the information gathering stage. While it is not yet entirely clear how best to merge the parallel elaborations, Feather has examined the different types of conflict that occur.

One approach to easing the integration of separate specification components is through tools which support negotiation. Robinson [1990] describes tools that allow a single arbitrator to evaluate the preferences expressed by various perspectives, and to guide the search for new solutions which satisfy all perspectives as much as possible. Taking a single domain model as a basis, in which needs are expressed as goals, where perspectives associate different values with these goals, integration involves searching for novel combinations of proposals, which increase the satisfaction of all perspectives. This is done using a joint outcome space on which an ideal, but probably unachievable combination of perspectives is used to stimulate consideration of other combinations that come close to this ideal.

These approaches to software specification all question the assumptions made by conventional software models which ignore conflict. Clearly, more work is needed to clarify how conflicts based on differing requirements can be resolved. One major issue is the need to establish common ground between viewpoints. No resolution can occur until participants have enough common ground to communicate; indeed, such common ground is needed before conflict can be expressed and recognised. In many of the above models, the common ground is assumed: in the parallel elaboration model the common ground is the initial specification from which the separate developments proceed, and in Robinson's negotiation model, it is the shared domain model. Only the viewpoints model does not make any assumptions about common ground, and even allows

different representation schemes to be used. However, it is not yet clear how correspondences can be found between viewpoints.

A second issue is how the resolution is devised. If all the possible conflicts are foreseeable, then the resolutions can be enumerated beforehand, either directly, or using a set of heuristics. Feather uses this approach in combining parallel elaborations, and has investigated the ways in which steps in the parallel process interfere with one another. Similarly, Robinson notes that there are some conflict situations which have conventional resolutions, such as the possibility of multiplexing for resource conflicts, and lists some heuristics which enhance his system.

However, in general the design process is not predictable, and sociological studies of negotiation indicates that good resolutions require creative input (see §2.3.5). This poses problems for automatic tools. Anderson & Fickas [1989] suggest that in well charted domains, the experts will be aware of typical conflicts and how to deal with them, which suggests that resolutions can simply be elicited from experts. This does not necessarily mean standard solutions are the best, nor will they always work in the environment introduced by automation. The best approach would seem to be to synthesize new solutions based on components of existing ones.

This section has surveyed recent work in software engineering that recognises the existence of alternative and possibly conflicting views in requirements analysis. The models developed in the work we surveyed allow for the capture of conflict by allowing the different views to be represented. In this way, they remove the conventional assumption that conflict can be avoided in software engineering. However, as yet no clear model exists of how conflict resolution might be modelled using these frameworks.

3.4 Conflict Resolution

In addition to the research described in the previous section, there are many related fields which have addressed the issues of multiple perspectives and conflict resolution. These fields fall roughly into two traditions – the mathematical and the behavioural – which approach conflict in very different ways, reflecting different disciplines. The former are concerned with numerical models of the processes of bargaining and decision making, while the latter are concerned with social responses to conflict and with how people approach negotiation, particularly in the context of organisational behaviour. Section 3.3.1 introduces some common terms with which to describe these fields, while sections 3.4.2 and 3.4.3 examine the two traditions in detail.

Finally, section 3.4.4 examines fields within computer science that have drawn on these two traditions. Distributed Artificial Intelligence (DAI) studies how knowledge bases can be partitioned to allow conflicting knowledge to co-exist, while Computer-Supported Co-operative Work (CSCW) examines the use of computers in human interactions. An important tool in CSCW is hypertext, which can allow groups of people to build and access an information base co-operatively.

3.4.1 Terminology

Before we survey other fields which address conflict resolution, it is useful to introduce a suitable terminology. We will talk about conflict between *parties*, as conflict may occur between individuals, groups, organisations, or even between different roles played by one person. Similarly, when discussing conflict resolution, we will refer to *participants* of the resolution process, to cover a similar diversity. Not all parties to a conflict need necessarily be participants in its resolution.

The approach used to settle a conflict is a *Resolution Method*. Methods include negotiation, competition, arbitration, coercion, and education [Strauss 1978]. Not all conflicts need a resolution

method, as not all conflicts need to be resolved. Three broad types of resolution method can be distinguished: *Co-operative* (or collaborative) methods, which include negotiation and education; *Competitive* methods, which include combat, coercion and competition; and *Third Party* methods, which include arbitration and appeals to authority.

Negotiation is a collaborative approach to resolving conflict by exploration of the range of possibilities. It is characterised by the participants attempting to find a settlement which satisfies all parties as much as possible. Such an approach has been variously termed *integrative behaviour* or *constructive negotiation* (to distinguish it from *distributive*, or *competitive* negotiation). This definition of negotiation is not universal. Authors such as De Bono [1985] restrict negotiation to its distributive variety, implying a process of bidding and concession-making, and so attack it as being inferior to an integrative approach. We prefer to give negotiation its broader definition, and call the concession-making process *Bargaining*.

There are other collaborative methods than negotiation. Some conflicts might be resolved by education, where the participants gain a better understanding of the problem, or simply learn about each other's viewpoint. Another important techniques is to reformulate the problem, so that it disappears, or becomes unimportant.

In contrast, *Competition* concentrates on achieving maximum satisfaction for a participant, without regard for the degree of satisfaction of other parties. However, a competitive approach is not necessarily hostile. An extreme form of competition is when all gains by one party are at the expense of others, which, in game theory, is termed a zero-sum game.

Third Party Resolution covers any situation where participants are unable to resolve a conflict between themselves, and so have to appeal to an outside source, whether this be the rule-book, a figure of authority, or the toss of a coin. Such a situation can occur with the breakdown of either negotiation or competition as resolution methods. There are two types of third party resolution: those in which the cases presented by each participant are taken into account, which we might term *judicial*; and those where a decision is determined arbitrarily (e.g. tossing a coin), or by factors other than the cases presented (e.g. by the relative status of the participants), which we might term *extra-judicial*.

Bidding and *Bargaining* are phases of the resolution process. Bidding is where participants state their desired terms for the settlement, often with an indication of the relative importance of them, as a basis for bargaining. Bidding takes place in some form or other in most resolution methods, as participants must present their side, although in methods such as coercion, the bidding might be one-sided and implicit. A *position* is the set of terms that a participant commits itself to by making a bid. In bargaining, participants search for a satisfactory integration of bids. In the simplest case this involves a converging sequence of bid and counter-bid, while at the other extreme, participants seek to blend complex bids together. Note that the description of the outcome as *satisfactory* depends on your viewpoint. However, bargaining usually results in a compromise, whereas true constructive negotiation seeks to develop a new solution which fully satisfies all participants.

3.4.2 Mathematical and Economic Models

3.4.2.1 Decision Theory

Decision theory is a prescriptive approach to help an individual make a choice among a set of pre-specified alternatives. While the generation of alternatives is considered of paramount importance, it is not generally addressed in decision theory, except where particular analyses of options can lead to the suggestion of new ones. The usual analysis has two components: an uncertainty analysis and a utility (preference) analysis [Keeney & Raiffa, 1976]. The interesting problems are concerned with resolving multiple conflicting objectives.

Decisions can be analysed in the following way. The alternative actions are listed (A_1, \dots, A_m). There are a set of attributes of concern (X_1, \dots, X_n) and each of the actions can be evaluated for each attribute, yielding a vector of values for each action. Comparison between two action involves the comparison of the two vectors, and weights can be used to give emphasis to attributes that contribute to particular objectives [Finkelstein & Finkelstein 1983]. There are, of course, a number of complications to this model. Firstly, the attributes are likely to be of incommensurable units, making direct comparisons difficult. Intangibles, especially psychological aspects, need to be taken into account, and these are notoriously difficult to measure and scale. Time also has an effect, as consequences of a decision will vary as they unfold. Finally the uncertainties involved require consideration of (multi-variate) probability distributions [Bell, Keeney & Raiffa, 1977].

A decision, as characterised by decision theory involves a single entity making a choice between a number of options based on some evaluation of the options. This may be contrasted with conflict, in which there is more than one entity, each with a different perspective, and the problem is not to choose between the bids favoured by those perspectives, but to find a solution that satisfactorily integrates the bids. In this sense, conflict resolution is more akin to problem solving than to decision making.

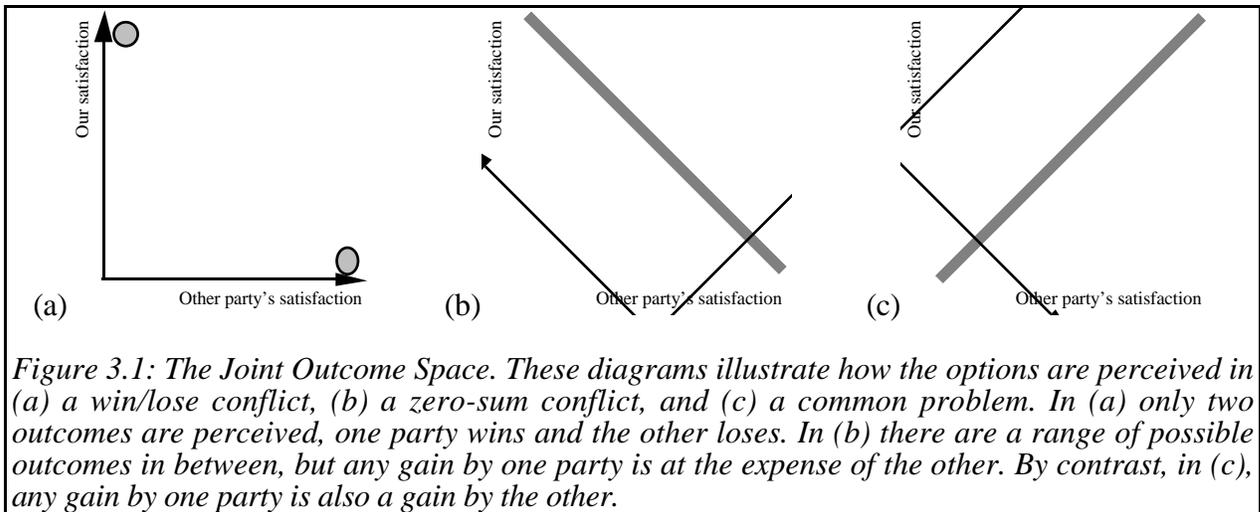
Even if we list a number of alternative possible integrations of bids as options for a decision, the usual assumptions of decision theory do not apply. Unless an arbitrator is brought in, the participants in the conflict will evaluate the options differently, obtaining different decisions, and so perpetuating the conflict [Galbraith 1977]. While decision theory does allow for alternative evaluations to be taken into account, this is to allow for uncertainty, and probability theory is used to analyse the alternatives. If the differences arise not from uncertainty but from different perspectives, no probabilistic analysis can be used. In the worst case, parties may not even agree over the measures to be used to evaluate the options.

This does not mean that decision theory has no role in conflict resolution. During the evaluation phase, there is ample opportunity for participants to use decision theory individually, to decide whether to accept a particular bid (from a number of alternatives), to justify such decisions, to decide what action to take next in the process, and to persuade the other participant(s) that a solution is satisfactory. Indeed, a participant who can use decision theory to evaluate a bid from the other person's point of view can have a distinct advantage.

As decision analysis is concerned with decisions by single entities, group interactive processes are ignored. Although decision analysis attempts to prescribe approaches for each individual, in which the actions of other members of a group can be regarded as uncertainties, there is no means to handle group decision making. Keeney & Raiffa [1976] use this observation to suggest uses of decision analysis for personal conviction, advocacy and reconciliation. In advocacy, decision theory can be used to provide a justification for a decision (often taken before the analysis) to convince others. Reconciliation is where an arbitrator needs to justify a decision to both parties to a conflict. Hence the principle role of decision analysis in conflict resolution might be that of providing supporting evidence for the various bids.

3.4.2.2 Bargaining Theory

Bargaining theory is an attempt to produce theoretical models of bargaining processes, in order to understand, for example, whether two parties will reach agreement, and what the terms of agreement might be. It is strongly rooted in economics, effectively an extension of decision theory, although attempts are sometimes made to draw wider social implications from the theories. Bargaining theory usually restricts itself to the process of what we term the bidding and bargaining phases, ignoring the context of the conflict. It is generally assumed that the parties both recognise the utility of an agreement, and enter the process of bargaining in an attempt to reach such an agreement. This reflects its concern with commercial deals and political agreements.



Patchen [1970] in his review of models of conflict resolution, defines bargaining theory (which he terms *models of negotiation*) as the study of “...the process by which two parties attempt, through a process of bid and counter-bid, to reach agreement on the terms of their future interaction ... and what the resulting rewards and costs to each will be”. He contrasts this to wider concerns of conflict resolution which include how the actions of each side influence the behaviour of the other, including the effects of coercion, co-operation, persuasion and threat. While early models restricted themselves to the process of accepting and rejecting bids, later models take such actions into consideration, allowing parties a range of actions at each point. The more sophisticated models take into account such factors as the cost of various actions, including the time element, and the cost of delaying agreement.

Bargaining theory frequently makes use of the *joint outcome space* as a tool for illustrating the utility of various bids to the parties involved. This is a graph, plotting the level of satisfaction of one party against another, and is applicable to two-party situations. Proposed solutions, or bids, can be plotted on the graph to show their relative strengths. The joint outcome space is also useful for analysing the type of conflict [Thomas 1976]. Figure 3.1 shows the graphs used by Thomas to illustrate how the options are perceived in an either/or conflict, a zero-sum conflict, and a common problem.

In contrast to decision theory, bargaining theory tends towards the descriptive, in attempting to build models that explain observed behaviour. Because of this nature, models developed in bargaining theory do not provide much guidance in practice [Strauss 1978]. For example, the graphs shown in figure 3.1 show how a party might perceive a conflict, but do not indicate how other resolutions might be found that satisfy both parties at once. In the zero-sum game, all gains by one party are perceived to be at the expense of the other. While this situation occurs frequently, particularly where the conflict is over the use of limited resources [Robbins 1974], it is rare that there are no other solutions to the problem than some form of division. Such solutions may include increasing the resource, decreasing dependence on a resource, or finding alternative resources. These possibilities are outside the scope of bargaining theory, which concentrates instead on the process of bidding and counter-bidding.

Bargaining theory is highly theoretical, and often the assumptions used do not apply to situations outside a very limited set of commercial and diplomatic bargaining. Furthermore, many of the theoretical models have not been empirically investigated, due to the difficulties in controlling the many parameters involved [Patchen 1970]. Models which use measurements of, for example,

levels of aspiration, rates of concession, costs of non-agreement, etc., do not give much indication how these might be interpreted in a practical study.

3.4.2.3 Game Theory

Rapoport [1974a] defines Game Theory to be a theory of rational decision in conflict situations. Participants are regarded as players, and game theory examines the strategies used by the players in the process of trying to achieve particular outcomes. It is usually assumed that the set of outcomes is known (though not necessarily finite), and that associated with each outcome is a calculable payoff for each player. All players are assumed to be *rational* in that each player's preference among the outcomes is determined solely by the sizes of the payoffs for that player. Also, players assume all the other players are rational in the same way, and so can use the size of other players' payoffs for information on their likely strategies.

Games can be classified according to whether they are two-player or n-player ($n > 2$), whether the player's choices of strategy are independent (non-cooperative games) or can be co-ordinated (cooperative games), and whether the sum of the payoffs for the players is constant or not. The latter distinction is perhaps the most interesting: zero-sum games include nearly all popular "parlour" games (chess, draughts, go, etc.) where all gains by one player are at the expense of the other (See figures 3.1a and 3.1b). On the other hand, non-zero sum games include outcomes where both players gain, and may require co-ordination between the players to arrive at a jointly optimal result.

		Prisoner B	
		Not Confess	Confess
Prisoner A	Not Confess	1 year each	10 years for A and 3 months for B
	Confess	3 months for A and 10 years for B	8 years each

Figure 3.2: The payoff matrix for the prisoner's dilemma. Each player must decide, in isolation from the other, whether to confess to a crime that the judge is sure they both committed. By confessing each will implicate the other, and their joint best strategy is for both to keep quiet.

The payoffs for a game are usually shown in a matrix. To illustrate, Figure 3.2 gives the payoff matrix for the prisoner's dilemma, a game which has received a lot of attention in the literature [Rapoport 1974b]. The prisoner's dilemma shows that a rational strategy might not always be the best one. Each player must choose whether to confess, and the choice will affect the sentence given to the prisoners: confession implicates the other prisoner. Whatever the other player does, the payoff for confessing is a smaller sentence, and so a rational strategy is always to confess. However, both players could improve their payoff by agreeing not to confess, but they need to be able to trust each other not to break the agreement, as the payoffs tempt them to do so. The prisoner's dilemma has been studied in tournaments of repeated games (with monetary payoffs, rather than sentences), which allow players to develop strategies [Axelrod 1984]. In this case players need to persuade each other, through their play, that they can be trusted. The repetitions allow players to use moves as punishments and rewards for previous actions.

There are a number of important limitations of game theory, which make the results less useful than they might otherwise seem. The biggest limitation is the restricted set of actions available in a game. For example, in the prisoner's dilemma, each player can choose only a co-operative or a non-cooperative action each move. While more sophisticated games introduce a larger set of actions, the rules still enumerate a bounded set of possibilities.

As game theory deals with pay-off matrices, it is usually assumed that the payoffs for any action are known with certainty by all players, in other words that all players have access to the same fixed payoff matrix. As it is rarely the case that payoffs are known exactly in real situations, this restricts the applicability of the results of game theory. If the participants do not know the payoffs with certainty, then it is likely that their perceptions of the likely payoffs will differ. Furthermore, game theory assumes that players are selfishly motivated and have to be induced into co-operation.

Despite these qualifications, game theory does produce some useful information about the kinds of strategy that can be used to induce co-operation and how various strategies pay off for the players. However, because of the limited scope for communication in the games, the games focus on how bidding strategies are developed over a series of games, rather than on single confrontations. Also, if we regard a move in a game as a bid, then negotiation in our sense is usually impossible, as a move, once made, cannot be modified. The exploration that participants to a conflict would enter into before bidding is shifted to a kind of experimental bidding, as the opponents test each other's strategies. An emphasis is placed on the learning process over a series of games, at the expense of deep understanding of a single conflict [Patchen 1970]. In a design setting, we are not concerned so much with strategy, but with integrative thinking.

3.4.2.4 Group Decision Making

An area related to both game theory and decision theory is the normative study of how individual preferences can be combined into a group decision. Much of this work was initiated by Arrow [1967], in his treatise on criteria for judging different methods of combining individual preference rankings. Arrow defined the problem as that of finding a "fair" method, or *welfare function* for combining individual preference rankings into a social preference. The conditions which Arrow proposed for a good welfare function are that it should be:

1. Defined for all possible combinations of individual preferences;
2. Representative, in that if just one individual changes his preference in some direction, the social preference should not change in the opposite direction;
3. Independent of irrelevant alternatives, in that the social preference between two options is not affected by the placement of another option in the individuals' preferences;
4. Not imposed, in that there is some combination of individual preferences that will achieve each possible social preference
5. Not dictated, in that there is no individual whose preferences dictate the social preference.

Unfortunately Arrow proved that these five conditions are inconsistent whenever there are three or more choices, so that no welfare function can satisfy them all. Subsequent work then, has examined the result of relaxing this set of conditions.

While no rule can satisfy all of Arrow's conditions, some are still more acceptable than others. Luce & Raiffa [1957] discuss majority rule as one of the most popular means of determining social preference. One of the biggest problems with majority rule is that it can lead to intransitive social preferences. For example, given three options, x, y, and z, society can end up preferring x to y; y to z; and z to x. Luce and Raiffa also point out that even where only one single societal choice is needed (as is usually the case in politics), and only two options are presented at a time for voting, the order of a series of such votes can affect the final outcome.

Zeleny [1982] questions Arrow's insistence on independence from irrelevant attributes, citing empirical studies which show that the inclusion of unavailable options can reveal more about the strength of individual's preferences, as well as altering their perceptions of what is desirable. Instead, Zeleny proposes his theory of the *displaced ideal*. Each person rates the options, and the

(infeasible) combination of each person's highest rated alternative, is used as a goal to guide the search for a feasible combination. The alternative which comes closest to this ideal will be chosen.

Whilst work on group decision making extends decision theory to cope with more than one decision maker, it still suffers from the assumptions used in game theory, that all the options are known. For example, while it can help decide which candidate is selected in an election, it does not consider whether there might have been other another candidate who did not stand, but who would have been more popular with everyone. Similarly, in design, we are often not concerned with deciding which of a number of options should be preferred, but in creating a new option that combines the best of each existing option.

3.4.3 Behavioural Models

3.4.3.1 Conflict Theory and Roles

The sociological view of conflict is concerned with social order and the evolution of social norms, i.e. how social order arises. Conflict theory states that society is in a permanent state of flux caused by the conflicting pressures of various groups. Co-ordination of society depends on the coercion of less powerful groups by the more powerful. Recent work recognises that there are many different groups in society with different goals, and that conflict is a frequent occurrence. However, Strauss [1978] points out that the majority of conflicts are resolved by co-operative means, often unconsciously, and yet despite this, very little attention is paid to these co-operative mechanisms. Strauss reviews the work of a number of sociologists in quite some detail, pointing out that much of their work assumes that some form of negotiation takes place, but that none of them actually address negotiation as such.

Viewing social orders as negotiated orders implies that a better understanding of negotiation is needed. This means studying not just the negotiation process (and indeed Strauss points out that far from being a single process there are many varieties of negotiation), but the context of the negotiation, including relating negotiation to other modes of interaction available, and consideration of the participants' own views of negotiation. Most importantly, the nature of the relationship between the participants can have a stronger bearing on the course of a conflict than the actual topic of the conflict. This is especially true where participants wish to ensure that a working relationship is maintained.

Deutsch [1973] begins to examine the nature of conflict, starting with a typology which distinguishes how the issues that the parties are arguing over are related to the objective sources of conflict. However, such a typology assumes there is a way to view the subject of the conflict objectively, which is impossible if the alternative perspectives are equally valid. More useful is the list given by Deutsch of issues involved in conflicts:

- Control over resources;

- Preferences and nuisances, where the tastes or activities of one party impinge upon another;

- Values ("what should be"), where there is a claim that a value, or set of values, should dominate;

- Beliefs ("what is"), when there is a dispute over facts, information, reality etc.;

- The nature of the relationship between the two parties.

This list closely follows those suggested by Robbins [1989], who adds that communication problems are a major cause of what he terms *pseudo-conflicts*, and De Bono [1985], who notes that conventional thinking styles encourage people to be contrary.

Social conflict is also studied in Role Theory, which attempts to explain conflicts in terms of the social roles which people play. Each person plays a number of roles, for example, parent, employee, spouse, union member, and conflict can arise between the roles played by a single person. The conflicts arise because of the different demands placed on different roles. These demands might arise from feelings of loyalty or responsibility, as well as from the expectations that other people hold of a particular role. Gross, McEachern & Mason [1958] describe a study of conflicting expectations of a school superintendent deciding the level of teacher's pay, and discuss how different superintendents resolved the issue. They found that the resolution often depended on how valid the subject considered the expectations to be, and whether the people that held them were likely to impose sanctions if the subject didn't conform to them.

3.4.3.2 Organisational Behaviour

Organisational behaviour is concerned with the functioning of organisations of various types with special emphasis on the business aspects and management. One of its major concerns is team-work within organisations, and how communication and co-ordination of teams can be effected. Early work tended to assume that all conflict was undesirable and should be eliminated from organisations. However, empirical work in the last few decades has demonstrated that conflict is an inevitable feature of group interaction, and the term conflict management has been used to describe means of resolving it.

Recently, Robbins [1974], among others, has advocated an expansion of conflict management to include not just resolution of conflict, but stimulation of conflict too. This is a result of observations that despite deeply ingrained social conventions that regard conflict (and disagreement in general) as destructive, conflict has a useful role in organisations. Conflict provides a stimulus to innovation as it involves questioning and evaluating received wisdom. It is also a major weapon against stagnation and resistance to change. Conflict management will then not just be concerned with peace-making, but with stimulation of conflict where appropriate. The difficulty lies in finding the right level of conflict, and ensuring it can be expressed in constructive ways. Robbins notes that if the conflict develops into open struggle, it is unlikely to remain constructive.

3.4.3.3 Conflict in Group Behaviour

Robbins [1989] distinguishes four stages in the development of conflict in group behaviour. The first stage is the potential for conflict in which, while unrecognised, the conditions occur that allow conflict to arise. These can be grouped according to type as:

- Communicational, including insufficient exchange of information, noise, and the semantic differences that arise from selective perception and difference of background;
- Structural, which includes the goal compatibility of members of the group, jurisdictional clarity and leadership style;
- Personal factors, including individual value systems and personality characteristics.

The second stage begins when the conflict is recognised, although the conflict will often not be acted upon until it is felt as well as perceived. When the conflict leads to action, the third stage, behaviour, is reached. The possible actions can vary enormously from the subtle to the overt. The nature of the action often dictates the course of the conflict, and in this stage the means for resolving the conflict will usually be initiated.

The final stage concerns the outcome of the conflict, and in particular whether the outcome is functional or dysfunctional for the group. An outcome that is functional to the group may not be considered functional by members of that group, especially where certain members feel they have lost. However, if the group's objectives are furthered, which usually also implies that the

relationships within the group have not deteriorated, then the conflict has been functional for the group.

3.4.3.4 Means of Handling Conflict

In §3.4.2.1 we mentioned a number of basic methods of conflict resolution, and categorized them as collaborative, competitive or third party. However, in organisational behaviour, researchers are more concerned with behavioural approaches to conflict, and whether it can be resolved or not. Studies have shown that different people are predisposed to tackle conflict in certain ways, according to their character rather than the context of the conflict. It is useful to identify these modes, and examine the utility of each.

Thomas [1976] describes five orientations towards conflict handling, based on areas of the joint outcome space (See figure 3.3). He defines them as follows:

Competitive - one participant seeks to dominate the process, achieving his or her goals without regard to others. It is useful for quick decisive action, or where unpopular actions are perceived as necessary for important issues.

Collaborative - participants seek to understand their differences and achieve a mutually beneficial solution. It is appropriate where participants' insights and commitment are important and need to be merged, for instance where their concerns are too important to be compromised.

Avoidant - the conflict is recognised to exist but is suppressed by one or more parties, or handled by withdrawal. It is useful where an issue is unimportant, where the potential disruption would outweigh the benefits of resolution, or where information gathering is more important.

Accommodative - a party becomes self-sacrificing to appease another, and places the other's interests above their own. It is useful when issues are more important to others than to you, where one party is losing and needs to minimise loss, or simply to build harmony and gain social credits.

Sharing - each party makes some concessions in order to reach a compromise. It is appropriate where temporary settlements or expedient solutions are needed especially under time pressure, or where goals are directly opposed.

Each of these modes is appropriate in some circumstances; the more aware people are of the possibilities the more likely a suitable mode will be used. It is useful to compare these modes with the methods available for conflict resolution. For example, collaborative methods such as negotiation and education, while most often used in the collaborative mode, can also be adopted in other modes. Education can be used to achieve conflict avoidance or

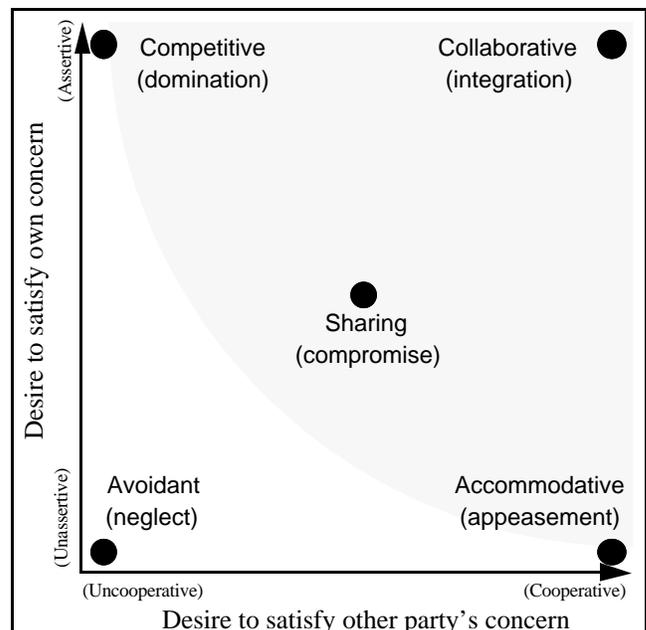
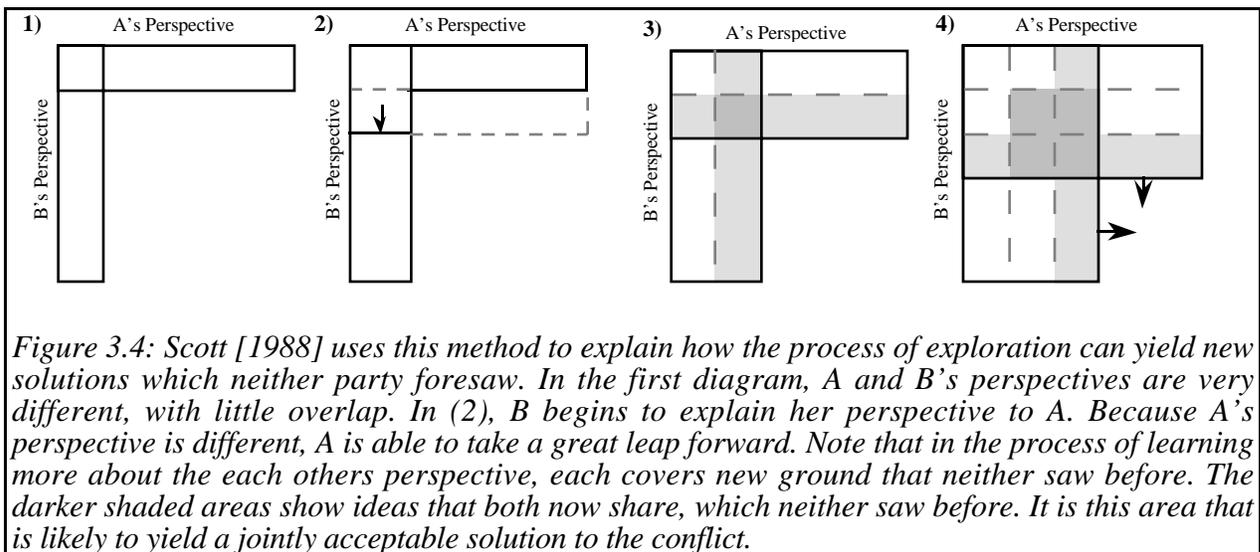


Figure 3.3: Behavioural modes of tackling conflict. Five distinct modes are identified, together with (in brackets) the outcome sought in each mode. The shaded area indicates the extent to which negotiation might be useful.



accommodation by enabling participants to understand their differences better. Similarly, negotiation can assist with achieving a compromise, seeking an accommodation, or regulating competition. It is likely that successful negotiation requires at least some assertiveness and at least some co-operation from each participant. This in turn implies that each participant must have some motivation to resolve the conflict rather than avoid it.

3.4.3.5 Negotiation Models

A number of models for conducting face-to-face negotiation in a commercial setting have been proposed, and are often popularised as management training material [Scott 1988], [Fisher & Ury 1981] [De Bono 1985]. Of these, Scott's book [1988] is fairly typical. He gives plenty of advice for preparation and the opening moments (setting the climate and procedure) of a negotiation, which are applicable to meetings in general. His model of the actual negotiation is of more interest here: he uses a four stage model to pace the negotiation. The stages are: exploration; bidding; bargaining; and settling.

Scott emphasises the exploration stage as the most crucial, taking up the most time. The exploration stage allows the participants to explore a range of possibilities before any confrontation takes place. In particular, it allows the participants to explain to each other their interests, which help to establish and maintain their long-term working relationship. The process is explained by Scott in a diagram (See figure 3.4), whereby at the start the two sides see only a tiny part of each other's perspective. As they explore each other's point of view, they discover shared goals that were previously obscured from both.

De Bono [1985] discusses the flaws in the argumentation process that render it ineffective as a means of negotiation. He shows how the assumption of a particular perspective (or theory) dictates how the world will be perceived, leading to a rejection of alternative theories. From this he argues that the Popperian method, of adopting a theory and attempting to disprove it, is misguided as the very adoption of the theory leads to a biased perception of the evidence. Instead, we need to hold several alternative theories at once.

More pertinent to the study of conflict resolution is his characterisation of argumentation as a polarising process. If an idea needs challenging, an opponent will set up an anti-thesis and initiate an argument. As the argument heats up, each side gets more rigid, and devotes less energy to exploring the alternatives. Each is certainly not encouraged to see any good in any part of the opponent's thesis. De Bono uses this observation to claim that a third party is needed to try an

build constructive solutions, breaking away from the divisive nature of argument. Unfortunately, De Bono himself gets carried away in arguing against argument, and does not consider any applications of argumentation useful.

The key to De Bono's method is to design solutions to conflict, as opposed to fighting, negotiating or problem solving. His complaints against negotiating and problem solving are based on very narrow definitions of these methods. For example he suggests that negotiation leads to compromise solutions, which are less than satisfactory all round. This assumes that negotiation is little more than a process of bidding, counter-bidding and concession making. Our definition of negotiation takes in integrative behaviour and so is more akin to the design approach advocated by De Bono. However, De Bono insists that such solutions can only be designed with the help of an independent perspective, i.e. by third party methods.

The bibliographic notes in Scott [1988], comment that many of the tactics De Bono suggests for third parties could just as easily be used by the negotiators themselves. Stefik *et. al.* [1987] suggest that removing the personal attachment to positions dissipates the problem. They observed that their computerised meeting room allowed participants to dispense with the feeling of ownership of ideas, and so reduce the associated emotions when ideas are discarded or adopted. This theme forms the basis to the negotiation model proposed by Fisher & Ury [1981], who recommend that rather than bargaining over positions, participants should focus on interests, and investigate options for mutual gain.

Fisher & Ury's model, which is the result of a large project on negotiation at Harvard, is really a set of recommendations, or tactics, rather than a rigid model. They too, characterise the usual bargaining process as polarizing, and recommend that the first step is to separate the people from the problem. This is to prevent emotions getting in the way, and to foster an attitude of working together on a problem. Then rather than making bids (i.e taking positions) the participants should focus on their interests. Their third recommendation reflects De Bono's insistence on multiple hypotheses, by developing a number of options that provide mutual gain, without concentrating on any single one. The final recommendation, to establish objective criteria, is designed to help come to an eventual decision.

These models provide some excellent strategies for dealing with commercial negotiations. However, while a co-operative mode is desirable for resolving design conflicts, it is not immediately clear that negotiation is the best method. However, some of the recommendations made in these negotiation models are clearly useful for design. To summarise the main points:

- Exploration of each other's perspective is essential to constructive negotiation;
- Participants should be separated from the bids in order to avoid polarization;
- The negotiation must begin by examining interests rather than positions so as to achieve a better understanding of the conflict;
- There is a need to generate many options without discarding any prematurely;
- The procedure (and criteria) for selecting a good solution must be agreed on.

3.4.4 Computational Models

3.4.4.1 Distributed Artificial Intelligence

Distributed Artificial Intelligence (DAI) studies how intelligence can be modelled in the co-operation of a set of agents [Huhns 1987], questioning the usual AI assumption that a single self-consistent entity (such as a conventional knowledge base) can demonstrate intelligence. Problem-solving activities can be divided up among agents, according to their specialist knowledge. As the system runs, the agents communicate partial solutions, and possibly control information (e.g. re-

allocation of tasks), amongst themselves, until they converge on a final agreed solution. The premise is that intelligence is an emergent feature of co-operative behaviour.

The paradigm has a natural ability to handle conflicting knowledge without the usual logical contortions arising from inconsistent conclusions. It allows agents to develop and maintain alternative hypotheses. Different agents will contain different knowledge, which may compliment or conflict with knowledge contained in other agents. This is demonstrated in the blackboard system [Nii 1986a, 1986b], which is typically used for recognition problems. Separate knowledge sources communicate partial hypotheses using a shared *blackboard*. The knowledge sources modify and extend existing hypotheses on the blackboard, until one of them adequately explains the phenomena being observed.

In the blackboard model the separate knowledge sources have no knowledge of each other [Erman & Lesser 1975]. Removing the global blackboard requires that agents communicate directly. In Lenat's BEINGS system they simply broadcast messages, usually in the form of requests for help, and hope that some other agent will reply [Lenat 1975]. Such systems still assume problems can be partitioned into totally independent sub-problems, and so the co-operation is reduced to that of trading tasks and sharing results. However, this is an unrealistic assumption, and is essentially conflict avoidance. There is a growing realisation that most problems cannot be partitioned in this way [Ginsberg 1987].

Various techniques have been proposed to allow epistemic reasoning, especially in multi-agent planning systems. Agents must be able to reason about what other agents know and are capable of in order to make full use of their existence, and to co-operate effectively [Konolige & Nilsson 1980]. Similarly, to communicate properly, agents must be sure their communications contain enough context to be understood, and that they are useful to the recipient [Appelt 1980].

Also, most DAI systems assume benevolent agents, all working towards the same goal. Rosenschein notes that in real world situations, perfect co-operation never happens, as the goals of any two agents will never coincide exactly [Rosenschein 1985]. He examines payoff matrices from game theory as a way of comparing goals, and discusses various situations in which conflict of goals can occur, and how they can be resolved [Rosenschein & Genesereth 1985]. Clearly it is these models that will have the most relevance to human conflict resolution, which also arises because of differences in the goals of the participants.

While DAI has contributed a variety of computational models of agent interaction, it has not progressed much beyond the game theoretical studies of conflict resolution [Sycara 1988]. Models which require resource conflicts to be resolved involve little more than bidding and bargaining strategies. Of most interest are those models which allow multiple hypotheses to be developed concurrently. At present these accept the first hypotheses to meet certain criteria, where the criteria used are built into the system. Work is needed to study how to combine elements of conflicting hypotheses as the basis for better hypotheses. In order to achieve this, the rationales behind each element of the hypotheses are needed.

3.4.4.2 Computer-Supported Co-operative Work

CSCW studies how computers can be used in collaborative activities. The field has only been recognised as such since the early eighties, although it has conceptual roots stretching back to the work of Engelbart [Engelbart 1963]. It takes a perspective of the computer not as an end in itself, but as an enabling technology, and studies how the computer might be used within the domain of interactions between people, and between people and information sources.

Greenberg [1989] divides the research on CSCW into two areas: Real-time collaboration and asynchronous collaboration. The former studies how computers are used in meetings both to provide electronic media for use in face-to-face meetings, and to facilitate remote conferencing.

Asynchronous systems are used to co-ordinate group working over longer periods, and to provide new communication channels between colleagues. These include electronic mail, bulletin boards and hypertext.

CSCW is relevant to conflict resolution in that it provides a number of tools which are intended to improve group collaboration and hence manage conflict. Most such tools are asynchronous, concerned more with the co-operative organisation of ideas over a period of time, although some meeting support tools, such as Xerox PARC's *CoLab* [Stefik *et. al.* 1987] address the same concerns. *CoLab* is a testbed meeting room which uses high resolution screens to replace the role of the whiteboard. Two of the software tools provided are of interest here: *Cognoter* and *Argnoter*. *Cognoter* is used to develop outlines for talks and papers collaboratively. It divides meetings into three phases: *brainstorming*, in which a shared window is used to jot down (unrelated) ideas; *organising*, where participants explore the relationships between ideas by linking them in some order and grouping them; and *evaluation* in which the ideas are tidied up, details filled in, and irrelevant ideas eliminated.

Argnoter is intended for use in evolving designs, where proposals have been partially worked out beforehand, and so the scope for conflict is greater. The intention is to overcome three major causes of dispute: personal attachment to positions, unstated assumptions, and unstated criteria. Again, three phases are used: *proposing*, *arguing* and *evaluating*. In the first phase, proposals are presented using webs of interconnected windows, often modifying existing proposals, or combining features of several. The arguing phase consists of noting reasons for and against each proposal, and linking these arguments to the proposal. The final phase is concerned with discovering the assumptions made by the arguments, and listing criteria for decision making.

Argnoter was designed to disassociate people from proposals so that they feel freer to critique. The assumptions associated with arguments are grouped together into *belief sets*, to characterise points of view, and to explore the consequences of those views. Stefik *et. al.* [1987] compare the tool to a spreadsheet in that it doesn't understand the proposals and arguments that it manipulates, but can compute the logical relationships between them. The resolution of conflict was one of the motivations in the development of the tool, with the hypothesis that making the structure of the arguments explicit reduces the disagreements caused by uncommunicated differences. In this way, the tool incorporates much of the methodology advocated by Fisher & Ury [1981]. However, the actual role of the tool in the conflict resolution process is not investigated, as the researchers were more concerned with how the enabling technology affects the meeting.

3.4.4.3 Hypertext

Hypertext systems [Conklin 1987] offer the ability to support the collaborative elicitation and organisation of ideas over longer durations. In particular, several hypertext-based software engineering environments have been developed, to help cope with the volume of information needed, and to support communication between a team. As Schuler [1988] points out, hypertext provides an excellent vehicle for supporting (but not supplanting) negotiation. Hypertext allows differing opinions and viewpoints to be represented and linked in the same system, encouraging plurality rather than stifling it.

The ability of hypertext to represent the associative structure of knowledge using typed links and nodes allows systems to reflect the structure of conflicts. Lowe has built a hypertext system directly based on a model for debate and reasoning, known as SYNVIEW [Lowe 1985]. By concentrating on debate, the system can contain all viewpoints rather than just a dominant one, with any group decision-making or voting based on access to a common body of material. The system is used for a group of experts to co-operatively compile information on a particular subject. It allows the experts to disagree, and provide evidence for their statements, by adding *backings* and *qualifiers* to chains of reasoning. Each statement of evidence can also be subject to disagreement. In this way, the system captures the style of a debate.

SYNVIEW appears to work well when the information has already been thought out by the experts, and is intended as a presentation tool. It does not assist the constructive thinking needed in design. This is because debating protocol concentrates on the adoption and defense of positions, encouraging an intransigence that inhibits creativity.

The dynamic nature of hypertext can be exploited to provide tools to explore problems in their early stages. While the ability to link together ideas associatively as they occur makes hypertext ideal for this kind of application, the ability to annotate and amend such links rapidly is important [Easterbrook 1990].

Software development hypertext systems allow designers to link together chunks of code, documentation, specification, etc, in order to record how the development progressed, and how the various parts of the system interact. For example, the Personal Information Environment (PIE) [Goldstein & Bobrow 1984], concentrates on perspectives. A program of research at MCC in Texas has produced gIBIS and ISAAC [Conklin 1989] which directly support group exploration of issues by allowing users to build up a network of issues, positions (or alternatives) and the arguments that support or refute them. While all these systems are designed with the software engineering application in mind, the principle could be adapted for other types of problem.

Several authors have used hypertextual linking in knowledge acquisition, for instance to link together (formal) interpretations and the original text (e.g. Woodward [1988], Regoczei & Hirst [1989], Jones [1989]). Gaines [1989] terms this *shadowing*, suggesting that items in a formal knowledge base can be linked to informal nodes of a hypermedia system, providing extra information for human digestion. This approach offers an excellent route to combining the power of a symbol manipulation system with the expressiveness of natural languages and images.

3.5 Summary

This chapter has surveyed work from a number of relevant fields. The first section discussed approaches to requirements engineering, which was divided into work on languages and work on specification processes. These two areas compliment one another: successful requirements engineering relies on both rich representations and suitable methods. Work on specification processes was divided into a number of paradigms, including formal methods, planning and transformation. Section 3.2 discussed related work in knowledge acquisition, and in particular techniques for elicitation of conceptual knowledge.

However, all these methods were found to share a shortcoming, in that they concentrate on the development of a single description, representing a single perspective. Part of the problem is that existing methods do not support the process of negotiation. Where a single consistent description is demanded, then all conflicts have to be resolved before the knowledge is formulated. These methods are therefore susceptible to the problems of conflict suppression described in chapter 2.

Section 3.3 describes recent work which recognises the need to handle conflict in knowledge acquisition and in requirements engineering. The former recognises the value of using more than one expert, and concentrates on how multiple experts knowledge can be encoded into a knowledge base. The latter centres on the need to consider the requirements of many different people, and the notion of viewpoints has been introduced, as a way of representing multiple perspectives.

Two major issues are raised concerning work on viewpoints. Firstly, common ground needs to be established between participants so that they can communicate: in most cases this common ground is taken for granted. Secondly, if conflicts are to be resolved, then there must be a way of devising solutions. Good resolutions are likely to require creative input, and so cannot be enumerated beforehand. Some conflicts might be handled using a heuristic approach; however it is not clear what types of conflict would be amenable to this approach.

Section 3.4 surveys a number of fields pertinent to the study of multiple viewpoints and conflict resolution, which were divided into three areas, *mathematical*, *behavioural*, and *computational*. The first of these includes areas of decision theory and game theory. These fields are highly theoretical, and make some restrictive assumptions about, for example, the state of knowledge of the participants, and their motivations. Nevertheless, research in decision theory has developed a number of tools which can be used to clarify the issues in decision making. Game theory and bargaining have laid the groundwork to evaluating and understanding the use of strategies in conflict situations.

The behavioural tradition, by which we mean much of the work on conflict from the social sciences, show more promise in channelling conflict situations into creative processes, but gives little indication how such processes might be supported with computational tools. The main results from research in these areas is that not only is conflict inevitable in society, both within and between individuals and organisations, but that conflict has a useful role in facilitating change and producing higher quality group decisions.

Of the fields within computing science, most are extremely young, some less than a decade old, and are only just recognising the role of conflict management. These fields seem to fall neatly into two camps: the AI approach, which attempts to automate completely the resolution of conflict between agents, and is based on mathematical approaches; and the supportive approach, in which computers are used as tools for supporting human conflict resolution, enabling greater collaboration.

There is a consensus of opinion in work on group behaviour that good solutions to conflict require creative input. If this is the case, and it certainly seems likely in design contexts, then it is unlikely that algorithmic conflict resolution methods can be provided. Rather, we can provide tools for analysing conflict situations and for encouraging co-operative and creative approaches to conflict resolution. The formal techniques developed in mathematical models of conflict can be used in these tools to supplement the creativity encouraged in behavioural models.

4 Specification from Multiple Perspectives

At the end of chapter 2, we described a number of objectives for a framework to support the requirements engineering process. The goal of this thesis is to develop a model which meets these objectives. The remainder of the thesis describes the development of a model of requirements engineering based on multiple perspectives. This chapter presents the underlying ideas, while the following two chapters describe in detail the two main parts of the model, together with the tools which have been developed to support them.

This chapter presents the rationale behind the multiple perspectives model. The first section presents the key ideas and assumptions underlying the model. The terms perspective and viewpoint are defined, and the issues of conversation support, representation schemes and changing requirements are discussed. Section 4.2 explores the advantages of modelling multiple perspectives, and demonstrates the operation of the model on an example drawn from the literature, while section 4.3 describes four key areas of difficulty in supporting the model. Finally, section 4.4 addresses the research methodology and gives a brief summary of how the model evolved.

4.1 Key Concepts

The model is based upon the development of a set of individual knowledge bases to represent the views, or *perspectives*, of the people with whom the analyst must interact. This allows the elicitation of knowledge from different sources to be separated from the integration of these sources' contributions. As the perspectives are captured and represented before the integration begins, the integration process is made explicit, allowing techniques of conflict resolution and negotiation to be applied to the collaborative process of building the specification. The way in which the specification is built can then be controlled by the analyst, rather than by the order in which the information arrives.

So far, the terms “perspective” and “viewpoint” have been used informally, to convey a rough idea. In section 4.1.1 we define exactly what these terms refer to in the model, and discuss their relationship. The remainder of the section describes the principles upon which the model is based: support for a conversational approach, the freedom to use many representations, and the accommodation of evolution in the specification.

4.1.1 Perspectives and Viewpoints

In order to reach an understanding of a business activity on which to base a requirements specification, some structure must be imposed. Traditionally, a description of the system being studied is constructed and incrementally elaborated. This description acts as a unifying theory with which to interpret the information about the problem as it is gathered. Typical analysis techniques include identifying processes and dataflows in order to structure the description. More recently, methods such as CORE have introduced the notion of viewpoints (see §3.3.3). Viewpoints are a logical development from processes and dataflows, which enable the processes to be more clearly identified with the agents that perform them, and hence capture the notion of responsibility [Finkelstein & Potts 1985].

Such techniques provide a way to structure and represent a system model, but do not necessarily capture any of the ephemeral or peripheral knowledge used by the participants during the process of requirements definition. For example, in CORE the viewpoints are used to structure a single,

consistent description of the system, leaving no room for alternative descriptions. In particular, CORE viewpoints are not allowed to overlap. Everything the analyst is told is either fitted into the description, by adjusting it where necessary, or remains unrepresented.

In this thesis we adopt a more abstract version of a viewpoint, which we term a *perspective*. Perspectives provide alternative descriptions of the system, which need not agree. Hence, the same concepts can be represented in different, conflicting ways. Each person can describe the system as they see it, rather than adding detail to someone else's description. For each perspective, the description may be structured using the most appropriate representation scheme. The perspectives are not restricted: they are descriptions of whatever each person considers to be relevant. In general, each person's description will be most detailed in the parts that concern them, and oversimplified elsewhere.

Once the perspectives have been elicited, the process of specification building then involves their comparison and integration. The resolution of conflicts between the perspectives becomes an explicit process, amenable to established techniques and computerised support.

Unfortunately, there is no one-to-one mapping between people and perspectives. Individuals are quite capable of holding several conflicting perspectives, and occasionally a group of people will share a particular perspective. One reason for this is that a person will often need to perform in more than one capacity, to carry out different tasks. Each task exists within a context, and may involve assumptions that are not valid for other tasks.

There is some correspondence between perspectives and roles – however, the perspective is not a description of a task or a role, but a description of the world from a particular angle. It would be more precise to identify a perspective with the context in which a role is performed. The relationship between people and their perspective descriptions is analogous to that between managers and their areas of authority (c.f. management domains [Sloman & Moffet 1988]). Authority can be shared and areas can overlap; conflicts can arise both between areas of authority held by a single person and between people who share an area of authority.

As an example, consider the job of a librarian. A librarian may have to perform several roles, and will describe the requirements of the library system differently, depending on which role she is considering them from. The intricacies of the task of cataloguing may conflict with the need for information at the issue desk. When considering the job of issuing books, the librarian might state a particular search facility is required. At some other time, she will be considering the cataloguing task, and might state that this same facility would be too complicated to provide. Such conflicts may disappear when pointed out to the person concerned, due to a conscious desire to appear consistent. However, in this case the conflict has not been resolved, but only suppressed: the search facility is still desirable, even if this particular librarian decides it can't be done.

Conflicts such as this often reveal when people are considering their requirements from different perspectives. As in the example above, suppression usually involves a decision which might not be appropriate to take until more information is available. In particular, it may become clear that there is a way of reconciling the conflicting pressures. Note that where the roles are performed by different people, there will be less pressure for the conflict to be suppressed.

Informally, a *perspective* can be defined as a description of an area of knowledge which has internal consistency and an identifiable focus of attention. Following Finkelstein *et. al.* [1989] (See §3.3.3) we use the term *viewpoint* more specifically, to denote the formatted representation of a perspective. A viewpoint consists of a set of statements which constitute a *description*. This description will be in a particular representation style, for example a particular type of diagram, or some specification language. A viewpoint will also have an *originator*, from whom the description was elicited. In most cases we can expect the originator to be a single person. However, as the

elicitation of viewpoints proceeds, coalitions of people will be seen to share a viewpoint, and so an originator may be a group of people, or even an entire organisation.

This distinction between perspectives and viewpoints allows us to regard a viewpoint as a partial description of a perspective. The viewpoint is restricted to what has actually been stated by the originator, in other words to the part of the perspective that has been elicited. This restriction of viewpoint descriptions is useful for validation purposes: the viewpoint does not contain any (possibly spurious) assumptions about what the originator might think – it is a verbatim description of what the originator actually said. In effect, statements are regarded as *commitments* for the speaker (See §5.2.1). The originator of a viewpoint can review her current set of commitments at any time, making modifications as necessary.

4.1.2 Conversation

One underlying assumption is that specification is a conversational activity. The definition of a viewpoint reflects this: a viewpoint is a record of the statements made by an originator. It is not a description of what a person might think, nor is it a reconstruction of what a person appears to do. Where such things need to be represented they form other viewpoints, originating from whoever is doing the hypothesizing or the reconstructing.

The elicitation of knowledge inevitably involves some form of interaction with the people who might have that knowledge. Such interaction can take many forms, and analysts typically use several techniques to gather information, some of which involve dialogue, while others have no verbal component (See §3.2.3). No assumptions are made about what form the elicitation takes, except that it involves some form of interaction with the originator, and presumably some degree of interpretation. The viewpoints capture the information arising from that interaction.

Whichever methods are used, at some point the analyst will need to discuss the gathered information with the originators, and it is these discussions that form the main vehicle for clarifying and validating the knowledge, and hence building the specification. The discussions between analyst and originator enable both parties to explore the current state of their viewpoint descriptions, and how those descriptions relate to the evolving specification.

Any framework for the requirements engineering process must not only support this conversational aspect, but must also encourage it, to ensure that the various contributors actually participate. In order to be effective, the discussions should not be restricted by the model, nor by the form of support provided by it: the discussants need to be free to raise any issues that concern them. If they are prevented from raising particular issues because of some imposition on the order of discussion, then such issues might never receive proper attention. In other words, the course of the discussions must be controlled by participants, rather than the model.

This requirement has an important ramification, in that where temporary inconsistencies and over-simplifications arise, there is no guarantee that the discussants will resolve them immediately. If the acquired information is to be represented as a knowledge base, then this knowledge base must allow participants to continue to interrogate it, to explore its current state, even in the presence of inconsistencies. This reflects the human ability to explore an idea, whilst ignoring any inconvenient or trivial details. The support tools must accommodate the inconsistencies until such time as the participants are ready to discuss them. They may turn out to be important conflicts, or they may be the result of errors or misunderstandings.

The use of perspectives supports the conversational aspect of requirements elicitation by capturing the information exchanged in conversation. It also facilitates and encourages discussion in several important ways. It allows participants to identify their contributions, and to see how those contributions have been represented – the analyst can present a viewpoint description back to its originator to initiate the process of discussion and clarification. Because inconsistencies are

assumed to be distinctions between perspectives rather than errors, there is no need to resolve them immediately they are detected. This allows the participants to explore particular ideas, by posing them as new perspectives. Finally, the conflict resolution process described in chapter 6 explicitly encourages the discussion of the issues underlying conflicts between perspectives.

4.1.3 Representations

The model provides a structuring for the collection of information from many sources. However, it does not impose any particular representation on that information. Many of the analysis techniques developed in the past are little more than representation schemes, where the form of representation used determines the way in which that information is gathered and analysed. In contrast, the model described in this thesis provides no representation scheme at all, in the expectation that existing representation schemes can be adopted as required.

The definition of a viewpoint (§4.1.2) restricted it to a description of the actual statements of the originator. However, as the intention is to enable some degree of interrogation and manipulation of the viewpoints, there is little to be gained by simply constructing them as natural language descriptions. Indeed, descriptions of this form are already available as the transcripts of conversations with a particular person. Rather, the statements need to be translated into a formatted description. This translation serves two purposes: it checks the analyst's understanding of the originators' statements, and it allows the description to be formally analysed in various ways. The formatted description can be presented back to the originator as a form of summarization, so that the originator can validate it.

It was noted in chapter 2 that requirements engineering involves many different types of knowledge, using many different representations. Typically, several representation schemes are used to supplement each other, whilst avoiding any duplication of information. It is often useful to describe the same information using different representations: the introduction of such redundancy can help reduce misunderstandings and discover conflicts. Different representations are more suited to particular areas of knowledge, in the same way that different programming languages suit particular problems [Petre & Winder 1988]. People have preferred ways of describing particular parts of their knowledge, and it is helpful if appropriate representation schemes can be used to structure the viewpoints.

The multiple perspectives model provides exactly this facility: the knowledge gathered for each perspective can be represented using any appropriate representation scheme. This allows the people with whom the analyst interacts to describe their contributions in a form that they feel comfortable with. The problem of comparing descriptions in different representations becomes part of the general problem of comparing information for which there might not be any prior common understanding.

4.1.4 Evolution

The process of knowledge acquisition can never be regarded as complete during the lifecycle of a software system. Although specification is an attempt to provide a complete definition of the requirements, some of those requirements will not become apparent until a new system is introduced. Also, the environment in which the system must operate will change, and these changes will be reflected in the perspectives of the people concerned with the system. The process of requirements elicitation therefore never really stops.

As elicitation proceeds even after parts of the specification have been constructed, there is a danger that new information will invalidate the decisions involved. This may reflect both changing requirements, and clarification of mistakes and misunderstandings in the original requirements. There are two ways in which evolution arises in the multiple perspectives model: the addition of new perspectives and the elaboration of existing ones. New perspectives might be added as the

result of new distinctions becoming apparent, additional participants joining the process, or new needs being identified. Existing perspectives will continue to be developed throughout the lifecycle: the originators will be encouraged to review their descriptions periodically, and changes are especially likely as a result of the validation process (§4.2.4).

The multiple perspectives model does not attempt to impose definite phases of elicitation and integration, as it is recognised that the integration process will entail the elicitation of additional knowledge. Rather, exploratory integrations are interleaved with the process of developing the individual perspectives. While the viewpoints provide the initial input to the resolution process, it is unlikely they will contain all the information needed, particularly if creative solutions are needed for major conflicts. The integration process will inevitably include a great deal of reconsultation with the originators of the viewpoints. The existing viewpoints provide indicators of where such consultation is necessary, and what type of additional information is needed.

Specification building is treated as an exploratory process, concerned with the comparison and combination of parts of existing viewpoints. The viewpoints themselves are not altered by this process, but remain as representations of the perspectives. Instead, the results of the comparisons are kept separate from the participating viewpoints. As such integrations always take place in the context of a particular set of viewpoints, the later addition of new viewpoints will not invalidate the reasoning behind any decisions taken. Similarly, because the participating viewpoints are preserved as a record of the information used in the decision-making process, any decisions can be re-examined. Hence, the effects of any further changes to the participating viewpoints, resulting from reconsultation with the originators, can be traced.

An inevitable result of this exploratory process is that decisions will frequently need to be re-examined. This is desirable in any situation where knowledge is incomplete. The extra effort involved is reduced by explicitly capturing the decisions, together with the knowledge on which they were based, and providing tools to allow full traceability: from viewpoints to the decisions they participated in, and from decisions to the viewpoints on which they were based. The problem is also reduced by encouraging analysts to delay decisions until they become necessary (See §4.2.3).

4.2 Rationale

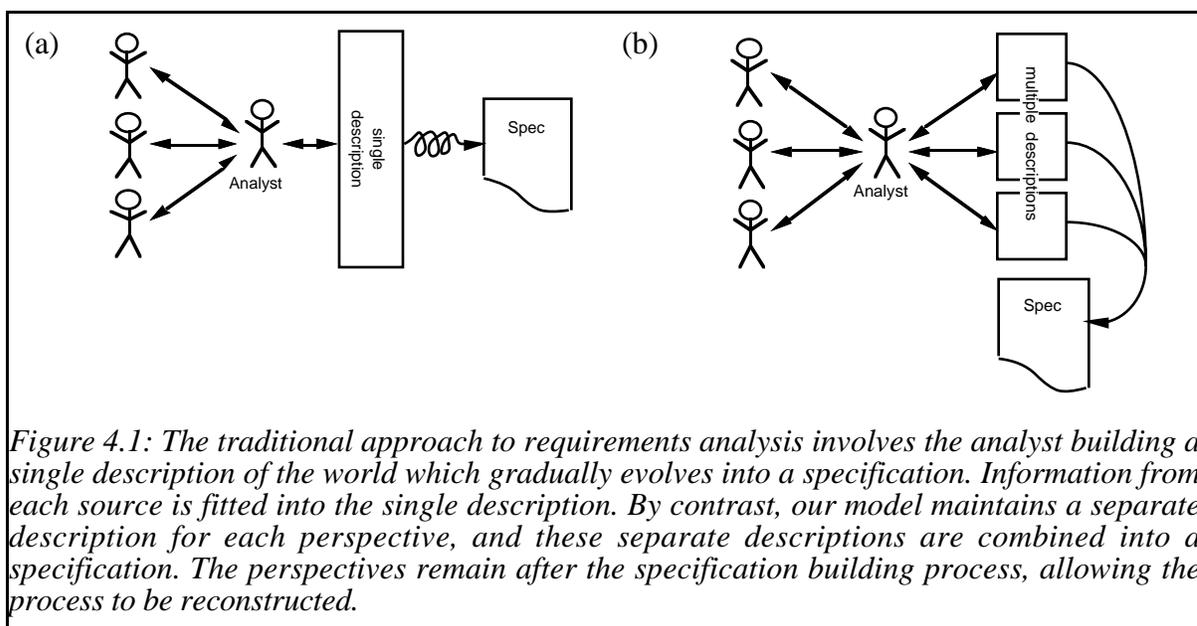
The main justification for introducing perspectives is the observation that the conceptual models which people use for their respective tasks seldom match exactly [Norman 1986], and that resolving the differences between them is a difficult problem. In the model we propose, a separate knowledge base is built for each perspective, to capture the knowledge offered by the person expounding that perspective. This ensures that each perspective is properly represented in the integration process, increasing the chances that the eventual specification will represent the views of all participants.

The model has important implications for validation of specifications. If the specification is to describe accurately the needs from which it is derived, then it must be validated by all the people whose needs it is supposed to represent. Validation of a complete specification is a difficult task, which should be divided up into manageable parts. The multiple perspectives model allows participants to validate their individual contributions separately. Having done this, they can then trace how their contribution relates to the specification, and to the decisions involved in building that specification. The model encourages their participation throughout the specification process.

4.2.1 Separating Perspectives

In the requirements specification process, individuals start with different knowledge and different preconceptions, which are gradually merged into a specification. Before these people have communicated some of their knowledge and established some common ground, it is difficult to combine their knowledge into a consistent specification without misrepresenting them, or neglecting parts of their contributions. Furthermore, the merging process involves important decisions which resolve conflicts between competing needs.

In the multiple perspectives model, the elicitation process involves the construction and evolution of individual viewpoints. As the viewpoints are developed independently, the analyst need not worry how the viewpoint being developed fits in with the rest of the information. This allows the analyst a much greater control over the process: the implications of a particular viewpoint description can be ignored while it is being developed, even if there is a direct contradiction with other viewpoints.



As each person's point of view is elicited separately, participants are able to describe what they see, rather than having to add to other people's descriptions. A description need not accommodate other points of view, and so remains an accurate representation of an individual's contribution. Furthermore, the model allows individuals to be inconsistent. Such inconsistencies can be represented by several viewpoints, and so do not prevent the analyst from reasoning with the evolving descriptions. As participants are not constrained to be consistent, they can explore what-if questions, and ignore inconvenient consequences of ideas as they are developed.

4.2.2 Combining Knowledge Sources

While the model does not impose specific phases of elicitation and integration of viewpoints, it allows the two processes to be identified and separated, even when they are interleaved. Without a framework for maintaining separate descriptions, it is difficult to avoid modifying incoming knowledge as it is elicited, to take account of what is already known: in this case the integration of knowledge from multiple sources remains an implicit process. In contrast, by representing the perspectives explicitly, the processes of resolution can be examined, with the perspectives captured separately, before integration takes place.

We have already observed that integration of knowledge from many sources is a decision-making process. Support for the explicit integration of separate perspectives provides a focus for this process. By maintaining a set of (often incompatible) perspectives, the analyst can build up the whole picture before attempting to combine them, delaying any decisions until an appropriate time (See §4.2.3). Where inconsistencies arise between perspectives, they are explicitly resolved by detailed discussion, as such inconsistencies often indicate the presence of important conflicts or the need for design decisions.

The model also facilitates the recording of the decisions. Given that the information used to make the decisions is already captured as a set of viewpoints, each decision can be captured and recorded, together with the context in which it was made. Furthermore, the process of documenting these decisions is automatic. The original viewpoints will remain available for examination, allowing the rationale behind the decisions to be seen.

4.2.3 Delaying Decisions

The previous section emphasised that the integration of knowledge is a decision-making process, and discussed how the model supports decision-making. We have also suggested that by accommodating inconsistencies, the model allows the analyst to delay making particular decisions until the relevant knowledge is gathered. This ability to delay decisions also contributes to the stated aim of not imposing any ordering upon the discussions between participants (See §4.1.2).

A decision about how to integrate several pieces of knowledge, or how to resolve a conflict represents a commitment: by making the decision, the analyst is committing herself to the outcome of that decision. These commitments inevitably involve a narrowing of attention, as the act of making the decision implies that there are other possibilities which will henceforth be excluded from consideration. A premature decision implies a commitment which may unnecessarily restrict the future development [Thimbleby 1988]. Most importantly, it can curtail the creativity process by discouraging exploration of novel ideas. Not only are other possibilities excluded from consideration, but the participants often forget they exist. Hence, while it is sometimes possible to reverse decisions if it becomes clear they are unworkable, in general, once a decision has been followed through it is hard to recognise that an alternative approach might be better. It is also hard to recognise which decision was faulty: if the alternatives are forgotten, in retrospect the decision will no longer be considered to have been a decision.

To prevent the unnecessary restriction of the development process, we distinguish between specification commitments and exploratory decisions. This is made possible by not maintaining a

single “correct” specification, which would embody all the commitments made so far. Instead the set of perspectives represents the many possibilities, and exploratory comparisons and integrations are treated as new perspectives, rather than as part of any specification. Eventually parts of these perspectives may become parts of a specification, but by avoiding treating them as such during the development process, we avoid restricting the exploration with premature commitments. The process can be regarded as delaying commitments, in that binding decisions are deferred until an appropriate time.

4.2.4 Constructing Specifications

In chapter 2, we described the role of the specification, concluding that it must be testable, unambiguously precise, modifiable, and representative. These criteria are affected both by the manner in which the specification is constructed, and by the form it takes. In particular, the degree of formality has a strong bearing on precision and testability, while the degree of involvement of clients will affect its representativeness.

An overriding concern is the validation of specifications. The multiple perspectives model assists with validation in two main ways: Firstly, participants can see how their contribution has been encoded, how it fits into the whole picture, and how their descriptions relate to others. This allows them to make changes where they feel a viewpoint is misrepresenting them. Secondly, as each person’s contribution is represented explicitly, the role it plays in the subsequent integration process can be traced. Hence any item in the final specification can be traced back to the viewpoints from which it originated. More importantly, a viewpoint can be used by its originator as a pathway into exploration of the complete specification.

Precision and the elimination of ambiguity are often achieved through the use of formal specification languages. However, these introduce their own problems: they can be difficult to read and will be unfamiliar to the people whose knowledge is being represented. This in turn introduces problems with validation. As the multiple perspectives model allows each person to examine how her contribution has been represented, in isolation from others’ contributions, the use of an unfamiliar notation is less daunting. We noted in §4.1.4 that the model make no commitment to a particular representation. In fact, the ability to mix representation schemes allows mediating representations [Johnson 1989] to assist with the presentation of a formal specification.

One of the strengths of the model is its encouragement of negotiation. The viewpoints are built by collaboration between the originator and the analyst. This collaborative process ensures that discussion takes place, allowing misunderstandings to be eliminated. The specification must be agreed between the participants, and therefore involves detailed discussion and resolution of conflict. This encouragement of discussion, and the common understanding it leads to, helps avoid ambiguities in the specification. A further source of ambiguity arises from differing uses of terminology. Such differences should not necessarily be eliminated, but they must be detected: it is when they remain undetected that ambiguities result. The resolution process described in chapter 6 provides a route to the detection of such differences.

While the model does not guarantee that the specification formed at the end is representative, it at least ensures participants have a greater role to play in the process. Where knowledge is accumulated in a single central description it rapidly becomes unrecognisable to the originators [Compton & Jansen 1989], partly through the encoding process, and partly through the elaboration by many other sources. This has a tendency to alienate the participants, as they lose any sense of ownership of the specification. In the multiple perspectives model, originators are the owners of their viewpoints, allowing them to be a part of the resolution process, and giving them a motivation to participate further.

The viewpoint knowledge bases developed during the requirements phase are retained as annotations to the specification. This allows experimentation to continue throughout the life of the

system, with the viewpoints providing the traceability and rationale to support the specification. The use of viewpoints provides a modularity that facilitates change: the viewpoints can be modified and added to as described in §4.1.5. Furthermore, as the viewpoint knowledge base used during development of the specification is retained as a part of that specification, a great deal of extra information is available to anyone who interrogates the specification .

4.2.5 An Example Problem

We have outlined a novel approach to requirements elicitation, using viewpoints to enable individual and idiosyncratic descriptions to be built. These separate descriptions can be associated with the perspectives people use when performing particular roles. In section 4.1.2 we described in very general terms a possible conflict between two roles of a librarian. However, we have not considered in detail how viewpoints are likely to differ.

There will clearly be a number of small difference of detail between viewpoints. For example, if the descriptions are in a predicate calculus, there will be a number of propositions which are true in one and false in another. Where quantifiers are employed to describe real-world concepts the potential for contradiction is enormous. Alternatively, if the descriptions are Entity-Attribute-Relation models, there might be some differences between what are chosen as entities in the domains, and what are considered to be attributes of other objects. Figure 4.2 presents an example adapted from Stamper *et. al.* [1988]: is the office of prime minister an entity, and the current holder an attribute; is the holder an entity, and the office he holds an attribute; or are they both entities with a relation between them? Although these examples may seem trivial, if such differences arise in the unstated assumptions on which people base their viewpoints, then their descriptions may

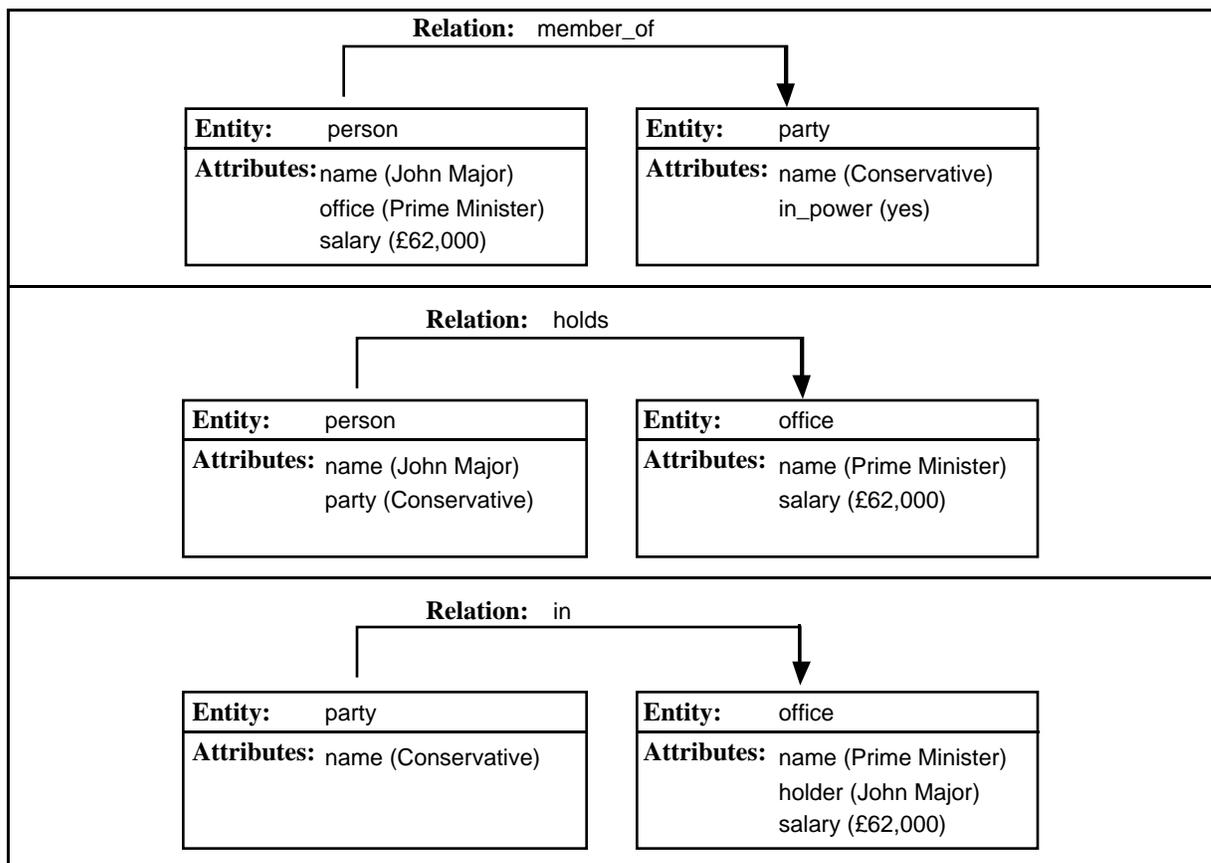


Figure 4.2: An example of different descriptions of the same concepts, all expressed using entity-relation-attribute diagrams (adapted from Stamper *et. al.* [1988]).

eventually seem completely incompatible.

We can illustrate the problems of basing reasoning on differing assumptions with a small example, analogous to that first described by Schoenmakers [1986]: A software analyst interviews two employees of the client organization to establish the requirements (in this case for a library system). The first employee states that some fact **A** is true:

We've got to have time limits for lending

whilst the second states that if **A** is true, then so is **B** (i.e. **A** \rightarrow **B**):

*If we've got to have time limits for lending
then we'll need fines to enforce them!*

The analyst puts these together, and concludes that **B** is true:

In that case we'll need fines as a requirement

Unfortunately, both employees know that **B** is false (neither really wants fines!), as is consistent with their individual statements. The second employee stated the implication believing the premise, **A**, to be false (she didn't want lending limits), quite possibly prompted by the analyst suggesting **A** because the first employee had mentioned it. The first employee, of course, wouldn't agree with the implication (other sanctions or even unenforced limits are not unreasonable for this particular library).

Any doubt as to whether the problem is a valid one can be expelled by listening to clients talking

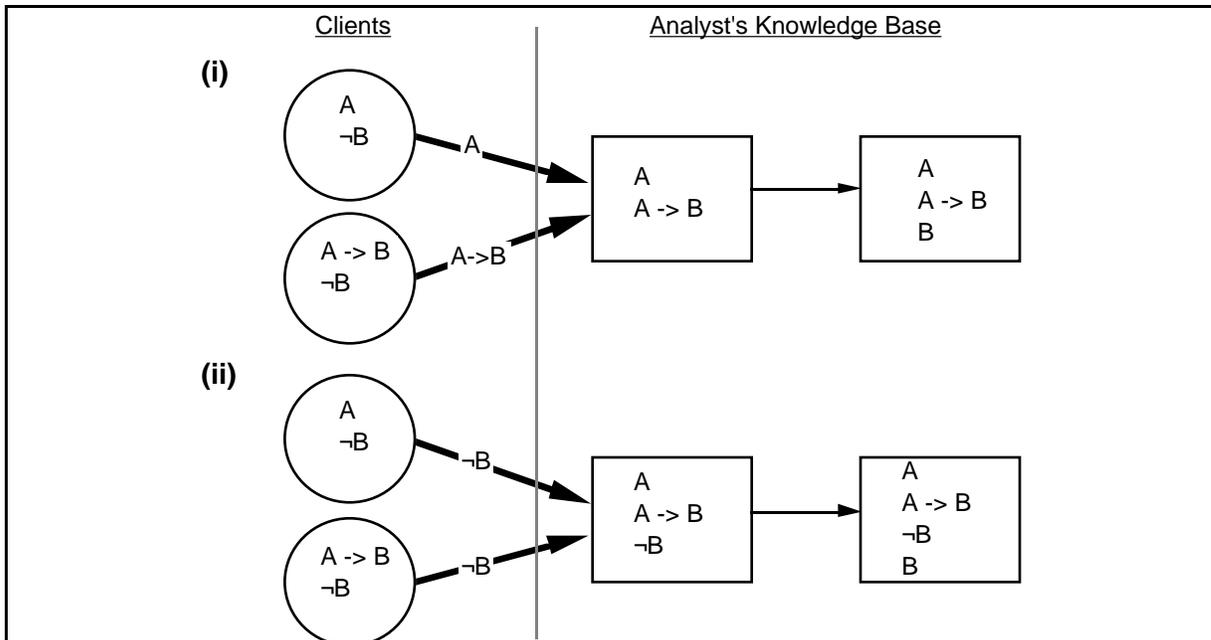


Figure 4.3: A conventional knowledge base cannot handle the problem. (i) shows the analyst drawing the wrong conclusion based on the simple evidence of the employees' first statements. There is nothing to indicate to the analyst that the conclusion is wrong. (ii) shows an attempt to rectify the problem by asking the employees about 'B'. This causes further problems, as the knowledge base then contains an inconsistency.

about the requirements. Phrases such as “Well, if **A** occurred we would need **B** to handle it”, and “If we had **A**, we would want **B**” are often used in the counterfactual sense: the speaker still suggests the implication, knowing that the premise is not true; or worse, reluctantly adds the implication once someone else has specified the premise. The rather simplistic scenario given above belies a tricky problem: it is more likely that the conclusion will be based on a long chain of inference, and the problem will go undetected until too late.

This problem cannot be detected in a conventional knowledge base, as no information will be available to say that the two statements **A** and **A** \rightarrow **B** originate from different sources, and might not be compatible (Figure 4.3a). Even if the analyst found out both employees believe that **not B** is true, perhaps by examining assumptions, and added it to the knowledge base, the problem would still remain, as the knowledge base is then inconsistent. Furthermore, the contradiction is not a direct one and might only be detected by generating all the consequences of the current set of assertions, which is often impractical (Figure 4.3b).

Spotting the inconsistency is only half the problem. Assuming a good tool could help to detect the problem, there is no information available about which of the three propositions (**A**, **A** \rightarrow **B**, and **not B**) are faulty. The analyst would need to decide which position is right before proceeding, as most inference systems cannot allow all three propositions to remain. This may force the analyst to make an early decision, in this case about lending limits, when such a decision may not yet be appropriate.

4.2.6 An Example Solution

We will now demonstrate how the model copes with the fines problem. The first thing to note is that simply adding the statements to a knowledge base discards important information about who said what, and in particular, the fact that the statements came from different sources. Using the model, the descriptions given by each person will be kept separate, and can always be traced back to their originator for clarification.

After the analyst has interviewed each employee, the system will contain two viewpoints, as shown in figure 4.4a. We can assume there will be other information as well as the statements about fines, including information about how lending limits and fines relate to the overall goals of the library, such as maximising access to the books. However, we will concentrate on the assertions already described. We can also assume that the analyst has represented these statements in some suitable formalism, for example predicate calculus: the first viewpoint contains a predicate such as `LENDING_LIMITS` while the second contains `LENDING_LIMITS \rightarrow FINES`.

At some point the analyst is considering the high-level requirements for the specification, and needs to know whether the library will use fines. A conclusion about fines cannot be drawn directly by either viewpoint, but nor is there any conflict between them. There are several possible avenues of exploration open. The analyst could create an exploratory viewpoint containing the assertions from both viewpoints, which can then generate the conclusion `fines`. Note that this does not affect the information stored in the original viewpoints. This conclusion can then be tentatively added to the specification.

There is a danger at this point that this tentative conclusion will become fossilised into the specification. However, several tactics are employed to prevent this. Firstly, the support system links this assertion in the specification to the originating viewpoint, which in this case is an exploratory one created by the analyst – no other participant has asserted it. Also, notes are attached to the other viewpoints for them to check whether their originators agree to this new item being added to the specification, as they had no information on it. This latter technique ensures that everyone has a chance to monitor the formation of the specification.

Eventually, one of the clients will point out that fines are not desired. This new information will be added to that person's viewpoint, and the process which added the requirement for fines to the specification will be replayed. This is possible because the evidence that supported the original process can be traced. The replay will detect a conflict between the tentative viewpoint asserted by the analyst and the modified viewpoint of the client. The analyst may decide either to withdraw the exploratory viewpoint, or invoke a conflict resolution process to explore the issues further.

As an alternative scenario, the analyst could have returned to either or both of the originators of the viewpoints and ask them directly whether there is a need for fines, before attempting to add it to the specification. In this case they will both assert that fines are not wanted, and this extra information can be added to the viewpoints (figure 4.4b). The analyst now has an answer about fines, and neither of the viewpoints has become inconsistent, and so can still be individually interrogated. The two original predicates will probably never be included in the specification, as they are not directly relevant: if ever they are, then some form of conflict resolution will need to be invoked to remove the inconsistency in the specification.

In summary, the problem can be solved with little difficulty by applying the model, as it records who said what, and what evidence is used for various inferences. There are several cases where conflict resolution might have become necessary: in these cases the resolution model described in chapter 6 would allow the participants to explore the issues and assumptions underlying their assertions, and at the very least find out why they disagree.

4.3 Outline of the Model

Having discussed some of the issues underlying the model and the reasons for adopting it, we will now take a closer look at the model and the tools that will support it. The model can be regarded as having two parts: the elicitation and integration of perspectives. Each of these will be explicitly supported; in other words the support system will provide a set of tools for knowledge gathering

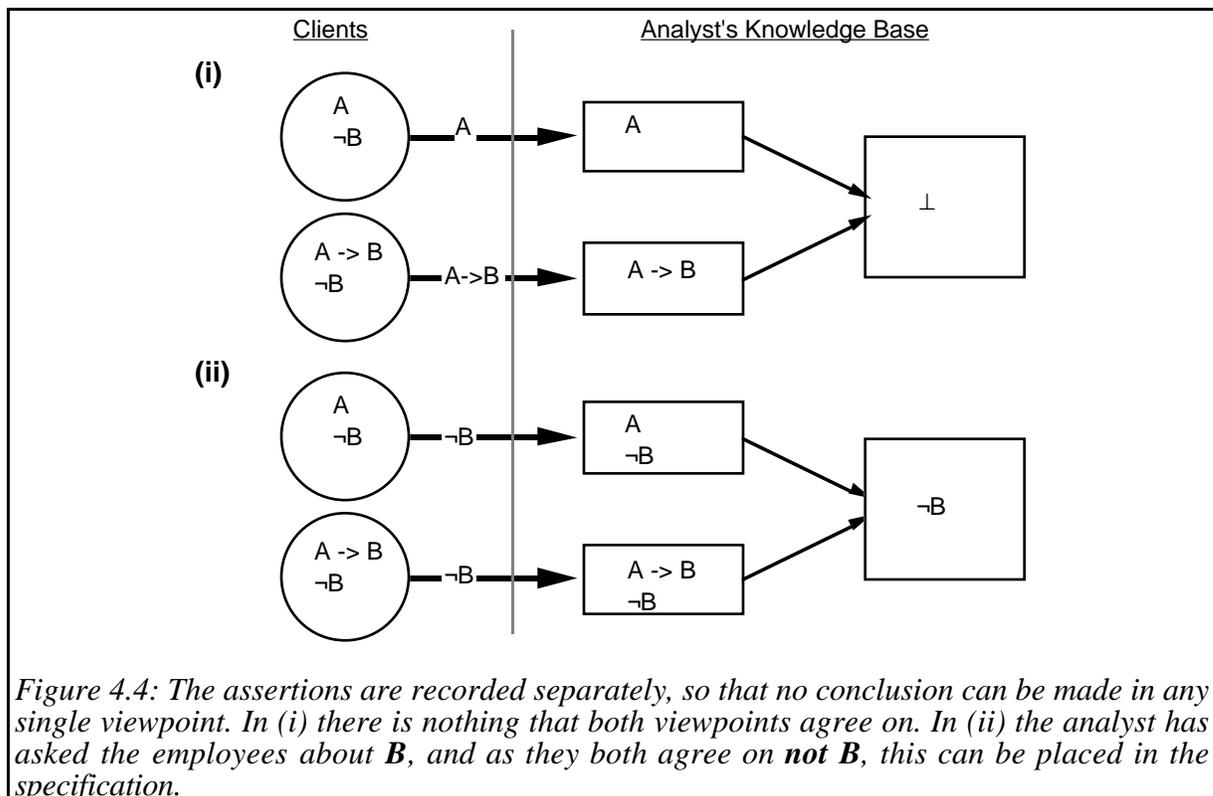


Figure 4.4: The assertions are recorded separately, so that no conclusion can be made in any single viewpoint. In (i) there is nothing that both viewpoints agree on. In (ii) the analyst has asked the employees about **B**, and as they both agree on **not B**, this can be placed in the specification.

and a set of tools for specification building. This is not to say that the two sets of tools won't interact: the normal use would be to interleave the two tasks, using exploration of the specification to guide the elicitation process.

The remainder of this section discusses a number of specific areas of difficulty in the model. The elicitation of perspectives introduces two main difficulties: how to identify the perspectives (§4.3.1); and how to build the viewpoint descriptions which represent the perspectives (§4.3.2). Integrating perspectives also introduces two areas of difficulty: how to compare disparate perspectives, for which there might be no common understanding (§4.3.3); and how to resolve conflicts between perspectives (§4.3.4).

4.3.1 Identifying Perspectives

Identifying the perspectives before the elicitation process begins is a difficult task, and the distinctions between the roles a person plays will often only become apparent during detailed discussions. This difficulty can be overcome by allowing the decomposition to evolve during the process. The analyst can begin with a set of agents based simply on people, or even groups of people, which can then be decomposed into roles as they are identified. This reflects the notion that the set of viewpoints identified will depend on the sophistication of the description. For example, a first attempt at describing a library system might include the role *user*, which will be shared by many people. Later refinement may distinguish different types of user, such as *student user*, *staff user*, and *visitor*, with different needs and different goals.

To develop a larger set of more specific perspectives from the initial generalised set, new distinctions between perspectives must be discovered. Overly-general perspectives might be split into several smaller ones, each of which focuses on a particular area of the knowledge contained in the original perspective. Chapter 5 discusses the criteria we used for deciding when to split perspectives, and the mechanisms used to perform the split. In general, there are three cases: when a viewpoint description becomes inconsistent, when a different representation scheme is needed for part of the knowledge, and when the analyst and originator decide that a distinct perspective is appropriate.

As well as criteria for distinguishing perspectives, rules are needed to determine to which perspective each piece of knowledge belongs. As there is no one-to-one mapping between perspectives and people, each person may be the originator of several perspectives. While each perspective has a distinct originator, and each person has a distinct set of perspectives associated with them, it is not always clear how new information offered by a person fits into their set of perspectives. We approach this problem by organising the perspectives into a hierarchy (See §5.1.3), where lower perspectives inherit from higher ones. Pieces of knowledge can then be moved around in the hierarchy until an appropriate level is found.

4.3.2 Developing Perspectives

Once a perspective has been distinguished, the viewpoint description representing that perspective can be developed. The viewpoints are formatted descriptions, intended to be analysed and reasoned with, while the knowledge offered by the originators is likely to be informal. Interpreting between the natural language used in the conversations and the appropriate formal symbols of the representation scheme is not a problem unique to this model. The model does not provide a novel solution to this problem, but rather assumes that existing elicitation and interpretation techniques can be used.

The problem is complicated by the fact that there is no unique interpretation for many statements. The model mitigates this complication in two ways. Firstly, extra perspectives can be added to represent alternative interpretations. Secondly, links between items in the formal description and the original transcripts are maintained, allowing inspection of the source material, for validation.

The freedom to make use of any appropriate representation scheme introduces two main problems. Firstly, for the adoption of a particular representation to be of use, the support tools must be able to manipulate descriptions in that representation. In particular, tools must be provided to check for consistency within a description, and to allow the display and interrogation of descriptions. The support environment we have developed incorporates sets of rules for manipulating a basic set of representations, and allows sets of rules to be added for any new representations which the analyst may wish to use. The second problem is that the unfettered use of multiple representations assumes that at some point they can be compared and consolidated into a consistent specification. It is by no means clear yet that this is always possible, but the method described in chapter 6 provides one possible approach, which is not restricted to any particular type of representation.

4.3.3 Comparing Perspectives

Once the development of viewpoints is underway, exploratory comparisons will need to be made. While we have emphasised that each viewpoint should be an uncompromising representation of a particular perspective, the participants will need to build an understanding of how the perspectives relate to one another. Such comparisons can reduce the amount of effort needed later to integrate the perspectives into a specification. Viewpoints which share identical subsets of assertions, especially definitions of terms, can be re-arranged to inherit these assertions from a common viewpoint. Also, viewpoints which contain very similar descriptions might be merged earlier rather than later.

Two very different frameworks in which to carry out this comparison process were investigated. The system described in chapter 5 used a blackboard as a shared exploration space, from which the specification was developed. Statements from any perspective could be placed on the blackboard, but could only pass from here into the specification if no other perspective disagreed. The model therefore required a consensual approach to specification. The system described in chapter 6 introduces a model more in keeping with the exploratory nature of the model. Any viewpoints can be compared at any time, and tools are provided for noting correspondences. If as a result of the comparisons some common description is evolved, this becomes a new viewpoint, which has as its originators the existing viewpoints that were compared. This approach successfully divorces the creation of a specification from exploratory comparisons between perspectives.

Given a model in which comparisons can take place, the actual process of comparing viewpoint descriptions is still a tough problem. The viewpoints may differ in use of terminology and use of representation as well as in the content material. Methodologies for comparing descriptions in particular representation schemes have been developed (e.g. Shaw & Gaines [1988] – See §3.3.2) and might be adopted for particular comparisons. However, the model does not constrain the choice of representation scheme, and so the general problem of comparing viewpoints remains unsolved. We make the assumption throughout this thesis that the originators of the viewpoints will not be wholly unfamiliar with their colleagues' knowledge, so that they will be able to suggest correspondences between the viewpoints. These in turn can be used as a basis for discussion of the relationships between perspectives.

4.3.4 Resolving Differences

The integration of perspectives to produce a consistent specification begins with the comparison process described above. The model allows differences between perspectives to be ignored until a resolution is needed, allowing time to gather the relevant information. If the difference is based on a misunderstanding, then delaying the resolution may give the participants a chance to discover the problem without having to make an unnecessary decision. Where the comparisons reveal real differences, then some form of conflict resolution is needed.

There are several difficulties in the conflict resolution process. If comparing perspectives is difficult then detecting conflicts must be equally so. Without a common understanding, all

differences will appear to be conflicts, whether based on misunderstandings, alternative interpretations, or genuine disagreements. We tackle this problem by considering all unexplained differences between perspectives to be possible conflicts: the definition of conflict given in §3.3.1 was deliberately broad for this reason. Part of the conflict resolution process involves exploration of the underlying causes of the conflict, before resolutions are generated. This exploration process involves deciding what form of resolution is appropriate: it may not even be necessary to generate a resolution. For example, conflicts over terminology might be formalised into the specification allowing a choice of terms in particular cases.

As the specification must be representative, the adoption of particular perspectives at the expense of others must be ruled out. Rather, for genuine disagreements, a constructive negotiation process is required, to find a solution that satisfies the concerns of all participants. Hence, when a conflict is detected, the issues which led to the conflict need to be elicited, so that solutions which satisfy those issues can be proposed and evaluated.

One of the biggest problems with conflict resolution is ensuring that all the interested parties take part in the resolution process. If a perspective has been neglected it may be necessary to discard the resolution and start again: if new perspectives are added after the resolution is generated, then this is always a danger. When a resolution is chosen for a conflict between perspectives, it represents an exploratory decision rather than a binding commitment. The process that led to the resolution is recorded along with the decision, so that the process can be re-examined later if necessary. Storing the resolution as a new perspective allows comparisons between this resolution and any other perspectives to be carried out.

4.4 Method

The previous section describes four areas of difficulty in the model. There are inevitably a number of important research themes arising in each of these areas, as the model covers a broad range of activities. However, this thesis concentrates on the development of the model as a whole. The model itself and the tools developed to support it are the key research contribution. Issues arising from the application of the model are illustrated with examples where appropriate.

Once the general approach was established, the applicability of the model was explored with a series of prototype support systems. The results from these were then used to refine the model. The systems took the form of a knowledge-based environment, with tools developed to manipulate the viewpoint descriptions. Section 4.4.1 describes the development history of these systems. Examples drawn from the case study of a library system specification were used to test the functioning of the these tools: section 4.4.2 explains the origins of this case study.

4.4.1 Tools

During the development of the model, three experimental systems were built. The first of these was built in response to the judge problem discussed in §4.2.5, and consisted simply of a set of rules for combining knowledge from separate sources (See §5.4), where all knowledge within the knowledge base was labelled with its source. The system satisfactorily solved the problems raised by the judge problem, in that it prevented incorrect solutions being drawn. However, it raised more questions than it answered, and in particular identified two areas which needed more research. The first of these was how to identify viewpoints: having established that viewpoints do not correspond to people, the attribution of each piece of knowledge to a viewpoint becomes problematic. The second problem was how to resolve differences into an agreed solution.

A new system was developed to explore the first of these problems: identifying and manipulating viewpoints. The paradigm of the blackboard system was adopted [Nii 1986a], where the rules governing how knowledge is combined on the blackboard were adapted from the first system. The

Consider a small library system with the following transactions:

1. Check out a copy of a book/ Return a copy of a book.
2. Add a copy of a book to/ Remove a copy of a book from the library.
3. Get the list of books by a particular author or in a particular subject area.
4. Find out the list of books currently checked out by a particular borrower.
5. Find out what borrower last checked out a particular copy of a book.

There are two types of users: staff users and ordinary borrowers. Transactions 1, 2, 4 and 5 are restricted to staff users, except that ordinary borrowers can perform transaction 4 to find out the list of books currently borrowed by themselves. The system must also satisfy the following constraints:

1. All copies in the library must be available for checkout or checked out.
2. No copy of a book may be both available and checked out at the same time.
3. A borrower may not have more than a pre-defined number of books checked out at one time.

Figure 4.5: This case study is a small library system. It raises a number of difficult issues but is particularly compact. The study was set as a problem for the Fourth International Workshop on Software Specification and Design and is based on a problem by R. Kemmerer.

need to identify some crude viewpoints before the elicitation process begins led to the development of an evolutionary approach, where viewpoints are split when they become inconsistent, resulting in hierarchies of viewpoints. Consideration of the actions performed on these viewpoints led to the commitment reasoning scheme described in section 5.2.3. The entire system is described in the next chapter.

Experience with this system suggested that it had one main failing: the rules for combining knowledge were inadequate. A method of resolving conflicts was needed to complete the model. Consideration of the various existing work on conflict resolution described in chapter 3 led to the development of a model for supporting human conflict resolution. This model and the support tools developed to demonstrate it are described in chapter 6.

The two systems described in the next two chapters are intended to be used in conjunction. The conflict resolution model assumes that viewpoints have been elicited and suitably represented, and would be invoked when conflicts arise between viewpoints. Together the two systems represent the two major parts of the model: the acquisition and resolution of perspectives. The support systems described are intended to be illustrative only, and do not address all the issues arising from the model. A discussion of further areas of research is given in chapter 7.

4.4.2 Case Studies

Throughout this thesis, examples from the domain of a library system are used to illustrate various ideas. The library problem is well-known in research on specification and has regularly been used as a set problem at the IEEE International Workshops on Software Specification and Design (IWSSD). The wording of the problem set for the workshops is given figure 4.5, and concerns a database to keep track of issue of books to users in a small academic library.

The library example has been criticised as a toy domain. However, its repeated use at the workshops, and the variety of different attempts that have been made to formally specify it and derive a design from it, indicate that it is by no means trivial. The main drawback with the problem as used at the workshops is that it is already an informal specification, which restricts the problem to a small sub-set of all the possibilities for a library system. A number of decisions and assumptions have already been made, and these represent a part of the requirements engineering

process. In other words requirements engineering includes the processes that went into the derivation of the of problem given in figure 4.5.

To provide a rich enough set of problems with which to demonstrate both the model and the tools which we have developed to support it, we consider not just the library problem but the processes that went into its formulation, and the processes that follow this formulation. For example, some of the analyses of the problem in papers presented at the IWSSDs can be compared to demonstrate different perspectives of the requirements, and the conflict that might ensue. Another set of examples come from consideration of the complications of requirements engineering catalogued in chapter 2, and how they might manifest themselves in the library domain. For example, disagreements may have arisen over the use of fines to enforce lending limits, leading to a problem like that given in §4.2.5.

Accounts of the requirements processes from real projects are hard to come by. Not only are they not published for commercial reasons, they are usually not even recorded: the first tangible document is the specification. However, some research projects have attempted to construct realistic case studies, recording both protocols from designers/analysts, and dialogues with clients. Such studies (e.g. Adelson & Soloway [1986]; Fickas, Collins & Olivier [1987]) provide some evidence that the type of problems we describe do in fact arise, and some of our examples are drawn from these studies.

4.5 Summary

This chapter has outlined a model of requirements engineering. The model is based on the construction of a set of viewpoints to represent the perspectives of the participants of the process. A perspective is defined as an area of knowledge that has internal consistency and an identifiable focus of attention; typically it will be associated with the context in which some role is performed. A viewpoint is defined as a formatted description of a perspective, and represents the elicited part of a perspective.

The model is based on a number of principles. Firstly, requirements engineering is seen as a conversational activity, where conversations between analyst and client explore the current state of the specification. Secondly, the participants should be free to choose any appropriate representation for different areas of knowledge. Finally, the construction of specifications is an exploratory activity, so that a specification must be allowed to evolve.

There are a number of reasons for adopting this model, which were presented in section 4.2. Underlying these reasons is an overriding concern for validation. By representing each person's contribution separately from any others, each person can identify their contribution, and can trace how their contribution affected the specification. As viewpoints are elicited and represented separately, the process of integrating them becomes explicit, and so can be guided and recorded. By building a description of the perspectives, the analyst can focus on the interaction between them, and understand how the different views arise. Furthermore, the decisions involved in the integration process can be delayed, as there is no compulsion to combine viewpoints. Finally, the model encourages participation in the requirements process, resulting in a more representative specification, and facilitating validation.

An important innovation is that the model separates two key activities: elicitation of requirements and integration of perspectives. By separating them, the problems of each activity can be considered in isolation. Section 4.3 described the four main areas of difficulty: identifying perspectives; building the viewpoint descriptions; comparing perspectives; and resolving differences. The remainder of this thesis examines these difficulties in more detail.

5 Modelling Separate Perspectives

This chapter describes a system called *Analyser*, which provides an environment to support the multiple perspectives model introduced in the last chapter. *Analyser* allows the analyst to build and maintain a set of viewpoint descriptions, and construct a specification from them. The viewpoints are created to represent the perspectives as they are identified. The system uses a blackboard to provide the mechanism for interaction between viewpoints; the tools which operate this blackboard allow the analyst to develop a specification by combining the descriptions held in the viewpoints.

Analyser does not attempt to provide a method for elicitation of requirements, but simply provides a framework in which the elicited knowledge can be represented and manipulated. The viewpoints consist of sets of statements, in some suitable formalism, and have no internal structure other than that provided by the formalism. As such they capture partial descriptions of the world (“perspectives”), without prescribing the components of those descriptions. Similarly, the system does not prescribe what a requirements specification should consist of, but concentrates on the need to ensure the participants agree its contents.

The first four sections of this chapter describe the main features of the system; each of these sections concludes with a summary of the relevant tools. Section 5.1 discusses how perspectives are identified, and introduces the notion of a viewpoint hierarchy. Section 5.2 describes how the viewpoints can be interrogated, using a commitment reasoning scheme. The viewpoints themselves are formatted descriptions and are usually derived from an informal set of statements: section 5.3 describes how statements in natural language are translated through a process of annotation. Section 5.4 discusses the operation of the blackboard, and the construction of specifications. Finally, section 5.5 presents our conclusions from the *Analyser* system, including the shortcomings which eventually led to the development of the conflict resolution system described in the following chapter.

5.1 Identifying Perspectives

The first problem described in section 4.3 was that of identifying perspectives. Before the acquisition process begins, there is little indication of what the relevant perspectives are, and yet it is desirable to know which perspective statements belong to as they are elicited. Most attempts to provide intelligent support for requirements definition make use of a pre-existing domain knowledge base (e.g. Reubenstein [1990]). In general, however, the prior existence of such a knowledge base cannot be assumed: the very nature of requirements elicitation is that it is the initial exploration of a new application. The knowledge needed to distinguish perspectives is unlikely to be available prior to the elicitation of the knowledge embodied in those perspectives.

Furthermore, distinctions between perspectives might only become apparent in retrospect. As noted before (see §4.1.2), the perspectives do not correspond to people, but rather are associated with roles. Not only is it not always clear which role a person is playing at any one time, but also roles can be shared by groups of people or organizations. This can make it difficult to identify an appropriate perspective for any particular statement as it is uttered.

5.1.1 Evolving Perspectives

As the elicitation of viewpoints proceeds, the distinctions between them will become clear, and new distinctions will arise. Rather than create an extensive set of viewpoints beforehand, the

Figure 5.1: Evolving viewpoints. A viewpoint representing a librarian contains the following statements:

```

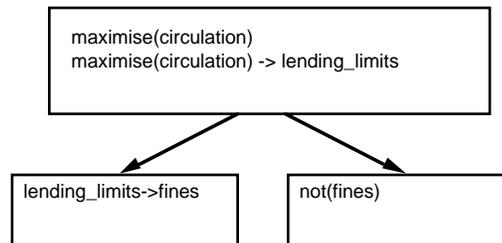
maximise(circulation)
maximise(circulation) -> lending_limits
lending_limits -> fines
    
```

The librarian then adds the assertion 'not(fines)':

```

maximise(circulation)
maximise(circulation) -> lending_limits
lending_limits -> fines
not(fines)
    
```

The viewpoint is then inconsistent by the inference rule modus ponens, and so a family of viewpoints is created, to isolate the inconsistency:



analyst needs to be able to evolve them as the process proceeds. The support environment must allow a degree of fluidity in the representation of viewpoints, so that the descriptions can be re-organised into new viewpoints as new distinctions are discovered.

As a starting point, an initial set of viewpoints corresponding to the people with whom the analyst interacts might be used. All the assertions that a particular person makes are collected in a viewpoint, building up a description of that person's view of the requirements. Each viewpoint is then an accurate representation of the interactions between the person concerned and the analyst. As an example, for the library system, the analyst may initially talk to two people: a librarian and a user. Two viewpoints would be created, to keep their statements apart. As the elicitation proceeds, other people may need to be interviewed, and so new viewpoints can be added to represent them.

We noted that perspectives do not correspond to people, as people may use several incompatible perspectives (§4.1.2). Hence, if a conflict arises within a viewpoint, then either a mistake has been made, or the viewpoint actually represents two different perspectives. In this case the viewpoint must be split into several separate viewpoints to represent these separate perspectives, as our definition of a viewpoint specified that it be a self-consistent description. There are several situations in which the need for such a split becomes clear; these are discussed in more detail in section 5.2. In general terms, a viewpoint will need to be split if it contains conflicting knowledge, or if a different representation is needed for some part of the knowledge.

The *Analysier* system copes with conflicts within viewpoints by creating families of viewpoints to handle them. New viewpoints are created as descendants of the original viewpoint, so that they inherit the original description. The conflicting statements are placed in different descendants, so that individually, each remains self-consistent. Any remaining statements which are consistent with both descendants remain in the original viewpoint, to be inherited by all descendants. The process is illustrated in figure 5.1.

Viewpoint Splitting Algorithm

- 1) Test for inconsistencies when new statement (**A**) is added to a viewpoint. This is likely to result in a number of inferences (the supposition set) being drawn from the union of **A** with the viewpoint. However, neither **A** nor any of the supposition set are added to the viewpoint yet.
- 2) If there are no inconsistencies then add **A** to the viewpoint, and add the supposition set to the viewpoint's list of suppositions.
- 3) If there is an inconsistency, then:
 - i) Create a descendant to contain **A**. This is the motivating statement for this descendant.
 - ii) Create another (empty) descendant. Any previous descendants of the original viewpoint now become descendants of this second descendant.
 - iii) If **not(A)** is in the original viewpoint, then move it to the second descendant. Otherwise add **not(A)** as a *supposition* in the second descendant. This is the motivating statement for this descendant.
 - iv) For each descendant, if any statements in the original viewpoint are inconsistent with a descendant, move them to the other descendant.

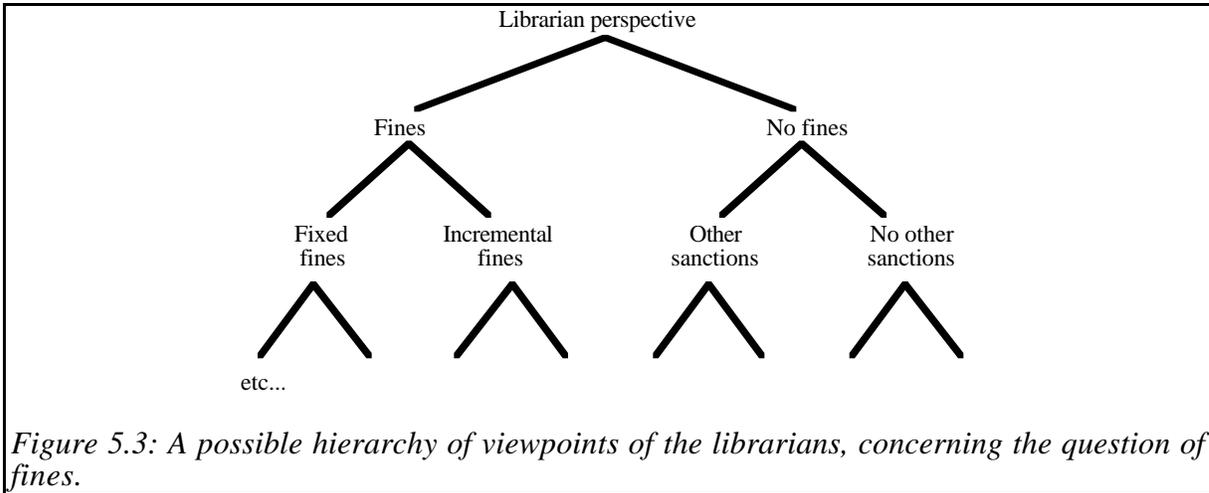
Figure 5.2: The algorithm for splitting viewpoints. In the Analyser system, each new statement added to a viewpoint is tested for inconsistencies, and if there are any, the viewpoint is split.

When choosing which statements to move into the descendants in order to isolate the conflict, there are often several combinations to choose from. For example, in figure 5.1 it would be sufficient to separate any pair of statements to break the chain of inference. As the viewpoint was (by definition) consistent before the newest statement was added, the new statement can be considered to have caused the inconsistency. Hence the newest statement is always chosen for one descendant, and the system attempts to identify the closest conflicting statement for the other. In this case it is the last step in the inference chain which generated the inconsistency. The new statement which caused the inconsistency, and its negation, are the *motivating statements* of the two descendants respectively. The algorithm used for splitting viewpoints is given in figure 5.2.

There are several ways of viewing the resulting family of viewpoints. The original viewpoint still exists as far as the analyst is concerned, and can be seen to contain competing descriptions of some particular sub-topic. The remainder of the viewpoint remains consistent, and so can still be used for the analysis process. At some later date, a choice might be made between these, and the descendants merged back into a single consistent description. An alternative interpretation is that there are now two separate viewpoints which happen to share some areas of description. Each of these is composed of the union of a descendant together with the parent viewpoint.

The process of refinement will eventually turn the original set of viewpoints based on people into a hierarchy of viewpoints, where each inherits the contents of its ancestors. In particular, all the viewpoints can be regarded as having an original global ancestor which holds any consensus information. As the specification process is concerned with establishing consensus, then this global viewpoint can be regarded as the evolving specification. As conflicts are resolved between viewpoints, the resolutions are added to this global viewpoint. The process is described in section 5.4.

This approach is similar in effect to DeKleer's "Assumption-Based Truth Maintenance System" (ATMS) [DeKleer 1986]. This maintains separate *contexts*, to describe separate points in a search space. The system makes assumptions while exploring this space. When an inconsistency arises, an appropriate assumption is retracted so that the inconsistency disappears. Often, any of a number of assumptions could be removed, and a new context is generated for each of the choices. The

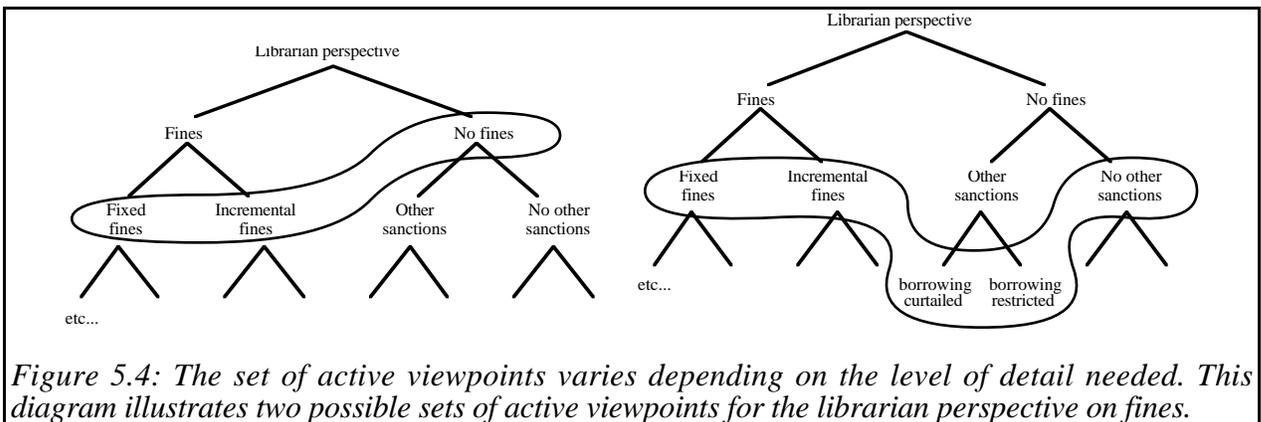


system assumes, however, that inconsistencies must be the result of previous assumptions. In contrast, *Analysers*'s viewpoints do not contain any assumptions (see §5.2.1); inconsistencies arise from alternative perspectives.

5.1.2 Viewpoint Hierarchies

Given the procedure described above, hierarchies of viewpoints will develop as the elicitation proceeds. These hierarchies are unlimited in depth, as descendants themselves may contain conflicts. The process can be regarded as exploration of possibilities: if each split represents a conflict requiring a decision, then each choice can be explored in more detail, giving rise to further conflicts. For example, consider the librarian perspective. There may be disagreement over whether fines are needed, and so two descendants are created to represent these positions. The viewpoint that advocates fines might itself be divided over the type of fine needed (e.g. fixed or incremental), and then further divided when the actual level of fine is considered (see figure 5.3). The discussion of these latter issues does not pre-suppose a decision has been taken about whether to have fines, and the viewpoint which excludes fines is still part of the model.

This example also serves to illustrate that the set of relevant perspectives varies depending on how the system is viewed, and the hierarchies can be used for information hiding. In the above example, when concentrating on another part of the specification, we may wish to ignore completely the conflict over fines, and consider there to be a single librarian perspective, which is the common ancestor in figure 5.3. At some point, another part of the specification may depend on



whether we have fines, and so it would be useful to consider there to be two librarian perspectives, one which wants fines and one which doesn't. At a greater level of detail still we might consider there to be several perspectives, representing the different types of fines, or other possible sanctions in the absence of fines. Figure 5.4 illustrates two possible sets of active viewpoints. Clearly the set of relevant perspectives varies depending on the level of detail needed.

Analyser supports this process of information hiding by keeping track of which set of viewpoints are considered active. Active viewpoints take part in the blackboard procedures (see §5.4) and are displayed in full to the user on request. Inactive viewpoints are not displayed, but their presence is indicated by flagging the disputed part of their parent viewpoint when the viewpoint is displayed. By selecting this flagged part, the user can ask for the descendants to be displayed, and if necessary added to the list of active viewpoints. Making descendants active in this way can be regarded as instantiating what the entire viewpoint would look like if each of the descendants were chosen to resolve the conflict. This can be useful for exploring consequences of decisions. Note that when descendants become active, their common parent is no longer considered as a single viewpoint, and so is no longer active. Once active, descendant viewpoints can be de-activated in favour of their parent.

The the inheritance structure means that the higher a statement in the hierarchy, the more widely agreed it is. However, the hierarchies are developed as distinctions between viewpoints are discovered. There is no guarantee that the more fundamental distinctions will be recognised earlier than those concerned with detail, even though the former should appear higher in the hierarchy. While interviews with originators naturally tend to start with general concepts, and gradually focus in on details, fundamental disagreements are often not discovered until the disparities of detail are explored. For example, in figure 5.3, the hierarchy should probably be arranged with the question of sanctions higher than the question of fines, as fines are a form of sanction. However, the question of fines occurred early in the discussions, and discussion about sanctions in general only occurred when considering what would happen if there were no fines. Clearly, the hierarchy will not always develop by expansion of the leaves; there may occasionally be a need to rearrange things higher up. In the next section we discuss how to determine at what level in the hierarchy a split should occur.

There are many ways to structure the set of viewpoints into hierarchies. The example given in the previous section illustrated that it can be hard to determine which statement(s) are in dispute. It is

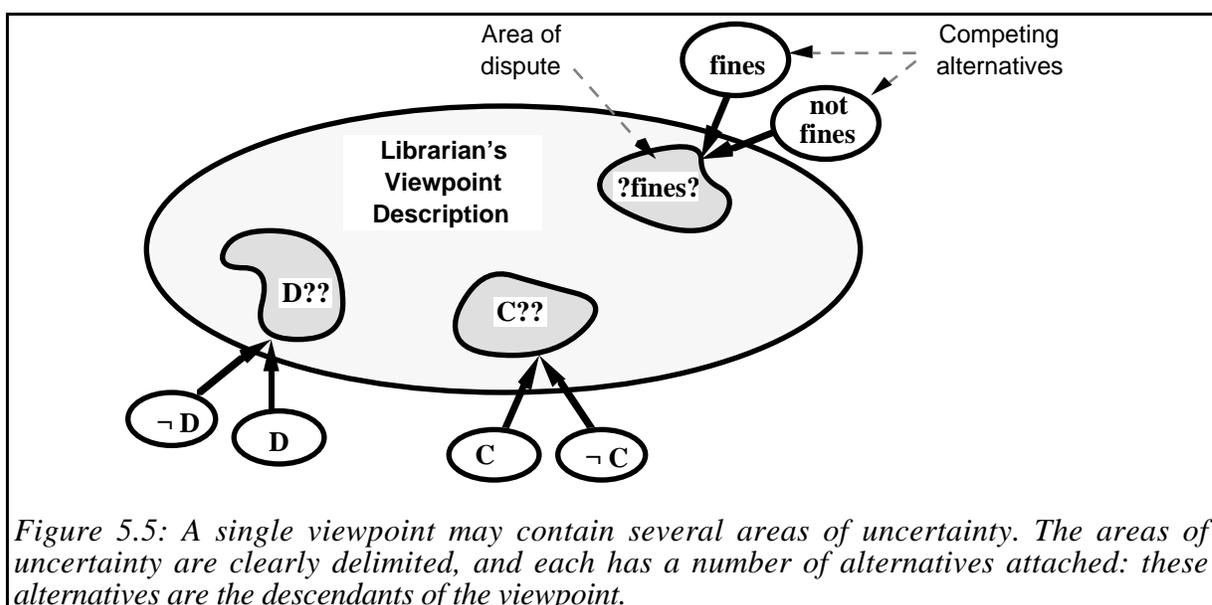


Figure 5.5: A single viewpoint may contain several areas of uncertainty. The areas of uncertainty are clearly delimited, and each has a number of alternatives attached: these alternatives are the descendants of the viewpoint.

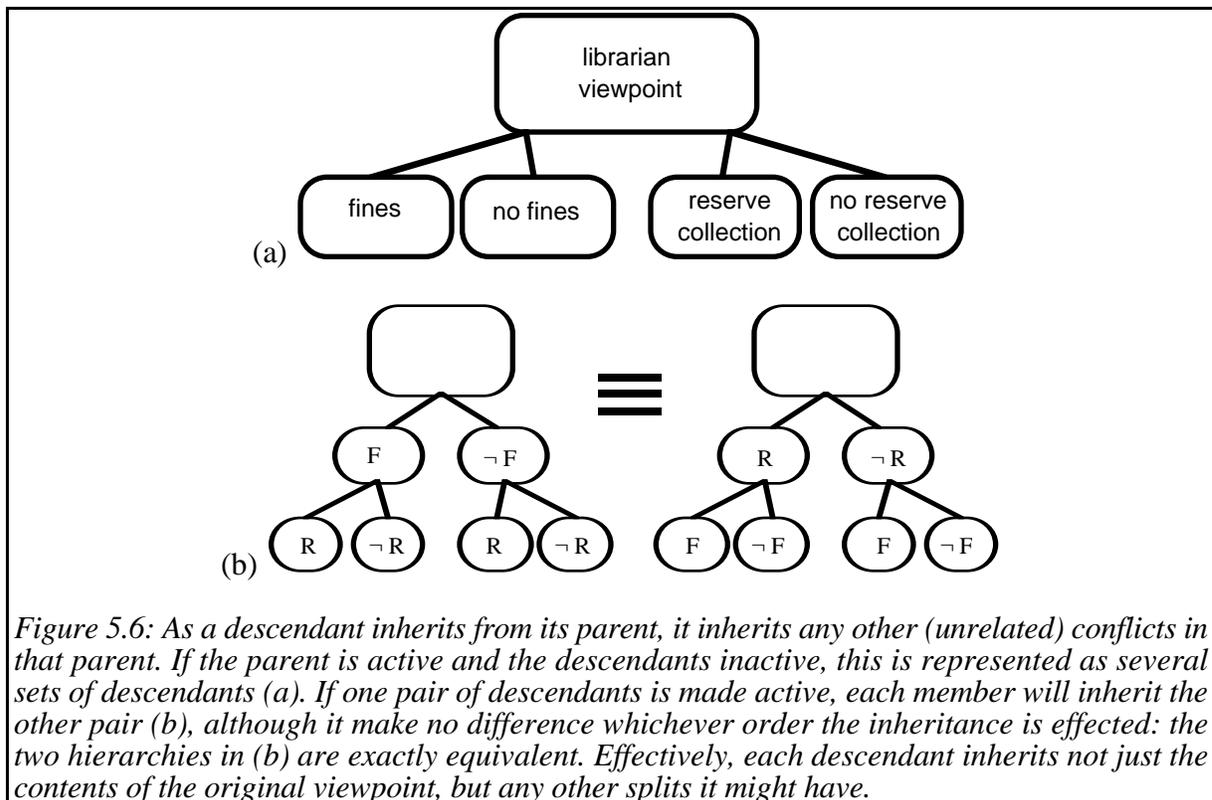
correspondingly hard to determine how viewpoint splits interact. In many cases a split only affects a small part of the viewpoint, so that the new descendants inherit a large body of common material. A future split in a different area of this common description might be entirely independent of the first split. Hence a viewpoint may be split in several different places, and have several different sets of descendants.

When a viewpoint description has several different sets of descendants, it can be regarded as a description containing several areas of uncertainty. For each of those areas, several options might exist (see fig. 5.5). Furthermore, some of those options might also contain areas of uncertainty. This view of the situation applies when the original agent is the active one. However, if we wish to make some of its descendants active, in other words, construct what the viewpoint would look like if particular decisions were made, the case isn't quite so simple. As each descendant inherits the whole of the parent description, it inherits any areas of uncertainty within that description. Hence if one set of descendants are activated, any other descendants of the parent viewpoint appear to be descendants of each of the newly activated viewpoints. The situation is shown in figure 5.6.

5.1.3 Placing Statements

Initially, when viewpoints are created to represent the people with whom the analyst interacts, there is no difficulty determining which viewpoint each piece of information belongs to. However, as the viewpoint hierarchies develop, the simple relationship between people and viewpoints breaks down. While each viewpoint has a specific originator, each originator may be represented by several viewpoints. When a person is describing some area of knowledge, it may not be clear which perspective that person is using, and hence which viewpoint to place the description in.

As splitting a viewpoint involves creating new descendants for it, all the viewpoints corresponding to a particular originator will be in a single hierarchy, having a single ancestor. All other



viewpoints in the hierarchy inherit from this ancestor, and so any new statement by that originator could be placed in this top-level viewpoint. If the new statement is consistent with all previous statements from the same originator, the new statement does belong in this viewpoint. Unfortunately, checking consistency with all the descendant viewpoints is computationally expensive: one of the aims of using viewpoints was to reduce the need for consistency checks.

The problem can be solved by the observation that new knowledge is rarely elicited in isolation. In a dialogue, the majority of statements will be related to their immediate predecessors; during validation new information is added in reaction to an existing viewpoint. The system records, for each originator, which viewpoint was last accessed, and uses this as the default for any new statements. The analyst may override this default when entering new statements, and choose any of the active viewpoints.

The new statement is checked for conflicts with the viewpoint to which it is added. If there is no conflict, it can be added directly. However, it is possible that the new statement may conflict with one of the descendants, as the descendants contain more detail. Rather than check all the descendants for conflicts immediately, this checking can be deferred because the descendants are inactive. As the viewpoint to which the statement was added is active, none of its descendants can be. The descendants are flagged as possibly inconsistent, to be checked when (and if) they are made active.

If the new statement conflicts with the viewpoint to which it is added, then the viewpoint needs to be split. In fact, it may not be appropriate to split the current viewpoint, as the conflict might occur higher up the hierarchy. The viewpoints at each level in the hierarchy contain less information than the ones below, so the system moves up the hierarchy until a viewpoint is found which does not conflict with the new statement. The last viewpoint with which the statement conflicted is the one which is split. Figure 5.7 illustrates this process. Figure 5.8 then describes what happens to any existing descendants of the viewpoint to which the statement is added.

We noted that the statement which caused a viewpoint to be split is recorded as the motivating statement for the new descendants (see §5.1.1). If this statement is retracted or modified, then the

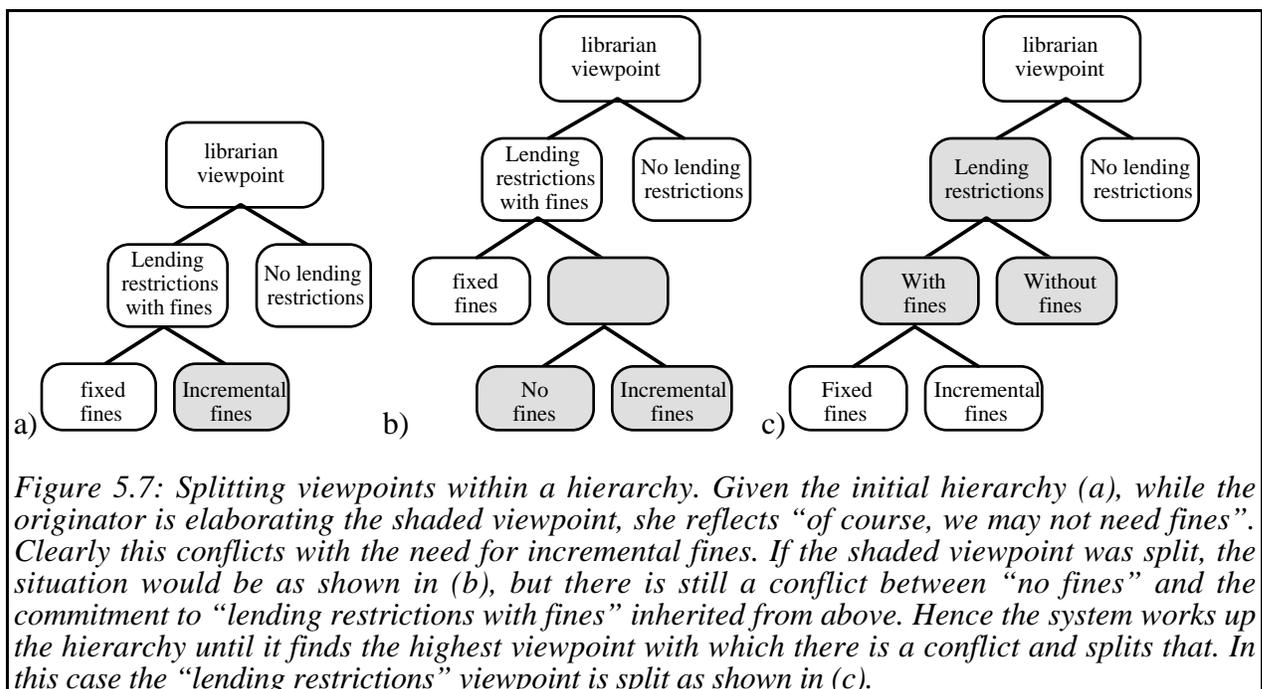


Figure 5.7: Splitting viewpoints within a hierarchy. Given the initial hierarchy (a), while the originator is elaborating the shaded viewpoint, she reflects “of course, we may not need fines”. Clearly this conflicts with the need for incremental fines. If the shaded viewpoint was split, the situation would be as shown in (b), but there is still a conflict between “no fines” and the commitment to “lending restrictions with fines” inherited from above. Hence the system works up the hierarchy until it finds the highest viewpoint with which there is a conflict and splits that. In this case the “lending restrictions” viewpoint is split as shown in (c).

conflict which the split represents may have disappeared, making it possible to re-unite the descendants.

Inheritance Rules for New Descendants

- 1) If no descendant exists, the usual algorithm (fig 5.2) is used (a, b).
- 2) If the statement is inconsistent with the viewpoint, then it follows that it is inconsistent with all descendants. In this case, the two new descendants are created. Any previously existing descendants now become descendants of the second new descendant (c).
- 3) If the statement is consistent with the viewpoint, it might be inconsistent with some existing descendants. For each family, test whether the new statement is consistent with each descendant. The following situations are possible:
 - i) The new statement is consistent for all existing descendants – in this case it can be added directly to the original viewpoint (d).
 - ii) The new statement is inconsistent with all existing descendants – in this case rule 2 above applies (c).
 - iii) The new statement is consistent with some descendants and not with others – if there is only one descendant in each pair with which the new statement is inconsistent, it is placed in the alternative to this descendant (e, f). Otherwise, the two new descendants are created as in rule 2. Any pairs which are both consistent with the new statement become descendants of the first new descendant; any that are both inconsistent become descendants of the second (g).

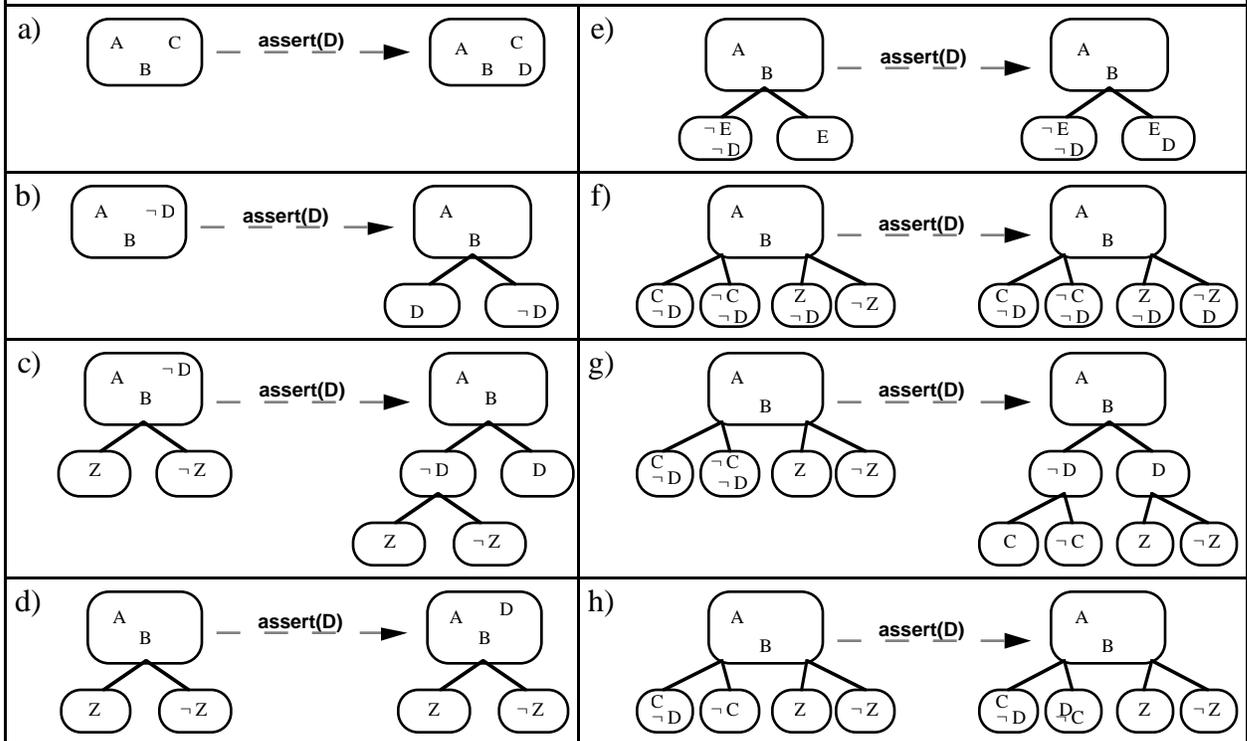


Figure 5.8: Rules for creating descendants, with examples. These compliment the algorithm given in figure 5.2. It is important to note that when a new statement is added to one descendant, it is incorrect to add it to any other families too, as they automatically inherit the split which contains the new addition, as shown in (h).

5.1.4 Functionality of Viewpoint Creation Tools

The *Analyser* system is a menu-based system for the creation and manipulation of a set of viewpoints. All viewpoints within the system are either added directly by the analyst, or created by the system to handle conflicts. Viewpoints added by the analyst are identified by name, where the name used is entered by the analyst. This will usually be either the originator's own name, or the name of a role played by the originator. Viewpoints which are created automatically are identified by their motivating statements, that is, the statements which generated the conflict that the viewpoints were created to handle. These viewpoints can be renamed by the analyst, if they appear to represent identifiable perspectives.

The basic commands for handling viewpoints are provided as a single menu. As well as commands for creating and renaming viewpoints, there are commands to list the active viewpoints and to display the contents of a particular viewpoint. From the list of active viewpoints, viewpoints can be selected to be de-activated, in favour of their immediate ancestor. In this case the selected viewpoint together with its sibling are removed from the list, to be replaced by their parent viewpoint (This action is not available for top level viewpoints). When the contents of an active viewpoint are displayed, any areas of conflict are flagged with question marks. These can be selected and the descendant viewpoints which handle the conflict can be activated. In this case the original viewpoint is removed from the list of active viewpoints. Note that when a viewpoint is displayed, all the statements inherited from ancestor viewpoints are also shown.

The commands for adding statements to viewpoints are described in section 5.2.4.

5.2 Reasoning Within Viewpoints

The set of viewpoints developed form a knowledge base which various people will need to interrogate. This interrogation includes finding out what has been asserted and testing various properties of descriptions. It also includes comparing descriptions, which we discuss later. This section is concerned with the processes of reasoning with individual viewpoint descriptions.

5.2.1 Viewpoints and Commitments

It is important to emphasize at this stage that viewpoints are not intended to be models of their originator's beliefs. Rather, we restrict the descriptions in the viewpoints to what the originator has explicitly stated: the viewpoint represents only what the originator has *committed* herself to by stating. This distinction is important when it comes to validation, as it is far easier (though not always trivial!) to get a person to agree that she *said* some statement, rather than agree that she *believes* some assertion. Where a person wishes to alter a description during validation, they are more likely to be successful in revising their own previous statements, rather than someone else's model of what they are thinking. Information added by other people or by the system using inference rules remain separate.

This adherence to commitments means that the system does not need to reason about what a person knows or believes. Hence, many of the epistemological problems of reasoning about belief (e.g. see Halpern & Moses [1984]) can be ignored. However, there will be many occasions on which the analyst will wish to speculate about what else a person might be willing to commit themselves to, and what assumptions their commitments make.

When interrogating viewpoints, a number of inferences will be made concerning the descriptions, which may or may not be subscribed to by the viewpoints' originators. Note that if we allow people the luxury of being inconsistent, then they also have the right to refuse to accept (or simply ignore) the logical consequences of their assertions. It therefore becomes necessary to distinguish between the assertions made by originators, which we call *commitments*, and the inferences drawn

from them using the inference rules, which we call *suppositions*. The latter represent the system's hypotheses of how the originator might extend the viewpoint description, and become commitments if the originator later confirms them. In effect, the suppositions act as a scratch-pad associated with each viewpoint, while the viewpoint descriptions contain only the commitments made by their originators.

As the suppositions are not a part of the viewpoint, but are associated with it, it is not always clear what should happen to them when a viewpoint is re-organised. Furthermore, suppositions can become invalid, as new commitments are added or old ones retracted. We solve these problems in *Analyser* by maintaining a dependency map between suppositions and the commitments on which they are based. If a commitment is deleted by being retracted or moved to a descendant, then the related suppositions are deleted or moved as appropriate.

5.2.2 Inference Rules and Conflict Detection

As the model makes no restrictions on the representation schemes used for viewpoints, the type of reasoning that can occur within the knowledge base can vary. We assume that a theorem prover or inference engine is provided for whichever representation languages are used. The inference rules for each representation are held as a separate viewpoint, from which the viewpoints using that representation inherit. This maintains the modularity of the environment, and allows new representations to be introduced as necessary.

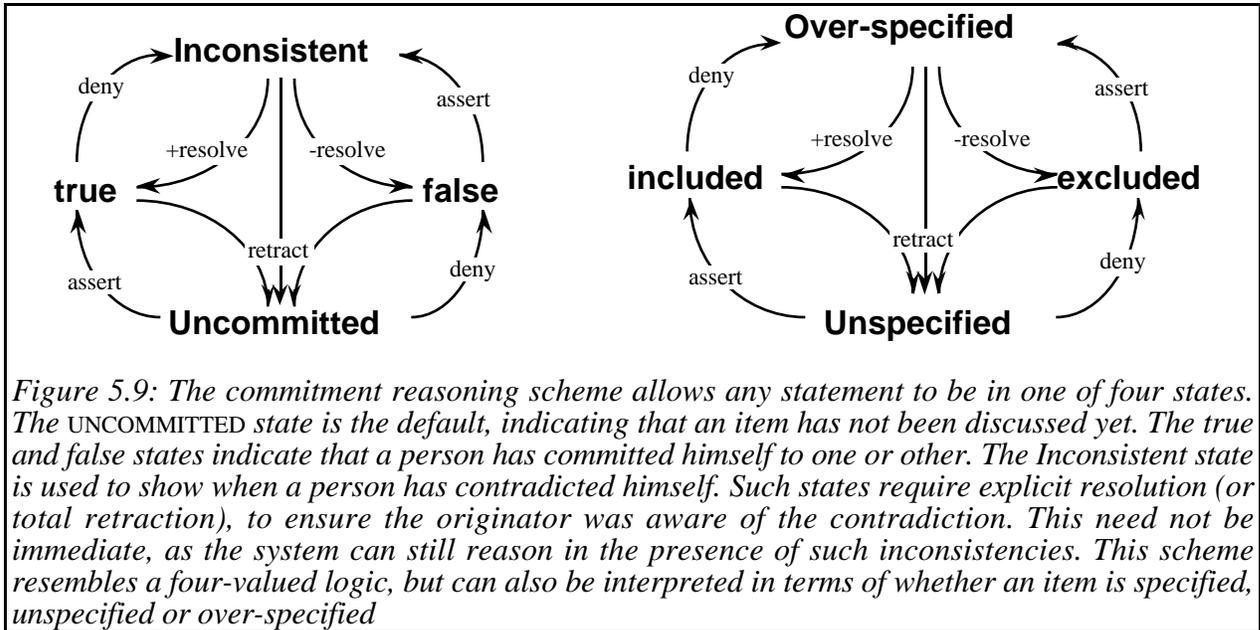
When we discussed splitting inconsistent viewpoints, we didn't clarify how conflicts are detected. A set of routines to test for conflicts are needed for each representation scheme used, which are stored with the inference rules and inherited by the viewpoints. In this way the kinds of conflict tested for in each representation scheme can be varied as desired. For example, the rules for detection of conflict might be based on detection of logical inconsistencies, together with tests for clashes of terminology.

Representing inference rules as viewpoints, to be inherited by other viewpoints introduces an extra degree of flexibility. When a person offers a description in a particular language, it is usually assumed that they subscribe to the rules governing the use of that language. However, if disagreements do arise over what forms of inference are valid in a particular representation, the viewpoint containing the rules can itself be split. In this way the system can handle representation schemes for which several varieties have evolved. This flexibility ensures that viewpoints do not have to make use of any aspects of a representation language with which their originators disagree.

Using this approach, new representations can be added to the system by adding a viewpoint containing an appropriate theorem prover, and a set of rules for detection of conflicts. We have not attempted to investigate the various mechanisms in detail, but assume that inference rules have been developed elsewhere for each representation scheme used. Currently *Analyser* only supports a first order predicate calculus, which was sufficient for our initial experiments with viewpoint development. This restriction also greatly simplified the problem of comparing viewpoints, and hence the rules for operation of the blackboard. The predicate calculus is supported with a simplified set of rules for detecting conflicts, based on the generation of contradictions through the application of rules such as modus ponens.

5.2.3 Commitment Reasoning Scheme

The viewpoint descriptions are built up through the addition of statements, or commitments. The statements correspond to the units which compose the descriptions: for example, predicates in a logic representation, or nodes and arcs in a graphical representation. Commitments can be added to (assertion) and removed from (retraction) a viewpoint. When a statement has been asserted, the viewpoint is *committed* to that statement; if a statement is retracted (or has never been asserted) the viewpoint is *uncommitted* on that statement. If a viewpoint contains two conflicting statements, it



is *over-committed* for these statements. Note that a viewpoint will only appear to contain conflicting statements: in reality these will be contained in separate descendants.

Analyser was initially built to handle descriptions in a first-order predicate calculus. As viewpoints are partial descriptions of the world, the closed-world assumption of formal logic does not hold: if a predicate is not in the viewpoint, it is not necessarily false. Rather, a distinction is drawn between UNCOMMITTED and FALSE. In effect, a three-valued logic [Blamey 1986] is used within viewpoints, with the three values TRUE, FALSE, and UNCOMMITTED.

Furthermore, asserting the falsity of a statement does not cause any previous commitment to the truth of that statement to be retracted, and vice versa. Effectively the result is a fourth truth value: INCONSISTENT. In practice, the inconsistency is isolated in descendant viewpoints, as noted above. However, unless those descendants are active, the hierarchy will be treated as a single viewpoint, which is inconsistent in some well-defined areas. These areas are represented as statements with the value INCONSISTENT, and which cannot take part in any inferences drawn from the active viewpoint.

This scheme was developed from the dialogue logic presented in [Finkelstein & Fuks 1989], which used a set of dialogue rules to enable a set of agents to exchange information. The rules allowed agents to assert, deny and retract statements, and to question or demand evidence for other agents' assertions. These actions are applicable in *Analyser* because a viewpoint represents only explicit statements made by the originator. Assertions and denials are used to add statements to viewpoints, where assertion means asserting some statement is true, while denial means asserting some statement is false. Similarly, retraction can be used to remove any commitment. The scheme can also be compared to that used in the Cyc project [Lenat *et. al.* 1990], although Cyc uses a different definition of "deny", as it does not distinguish between axioms and inferences in the knowledge base.

In summary, the commitment reasoning scheme attaches one of four values to any statement within a viewpoint, and these values are altered by assertions and retractions. The transitions are laid out in figure 5.9. An analogy can be drawn with the notions of over-specification and under-specification, as shown in figure 5.9b. We have not attempted to extend this scheme to other representation schemes, particularly those where conflicts cannot be characterised as logical

inconsistencies. However, we anticipate that the values of UNCOMMITTED and INCONSISTENT would still be needed to label statements in any representation scheme used within the viewpoints.

5.2.4 Functionality of Viewpoint Manipulation Tools

A set of tools for building, interrogating and manipulating viewpoints is provided in the *Analyser* system. The functionality of the viewpoint building tools follows closely the transitions of the commitment scheme described above. The actions apply to the current default viewpoint unless another viewpoint is explicitly selected. In the latter case, the selection is made from a menu of active viewpoints.

The following commands are provided in a single menu:

- assert(X) – add the statement X to a viewpoint. If the statement X is already included in the viewpoint, the user is warned, and no action is performed. If the statement X conflicts with the viewpoint, according to the conflict detection rules, the value INCONSISTENT is given to the statement. The user is asked to confirm this, and the viewpoint is split as described in section 5.1.3.
- deny(X) – add the statement not(X) to a viewpoint. This is exactly equivalent to assert(not(X)). For representations which do not permit negations, this menu selection would be unavailable.
- retract(X) – removes both the statements X and not(X) from a viewpoint, if either appears as a commitment in the selected viewpoint. If neither exist, the user is warned, and no action performed.
- resolve(X) – invokes a resolution process for a statement X which is inconsistent. The user chooses X from a list of statements for which the viewpoint is split, i.e. those which are the motivating statements of the viewpoint's descendants. The chosen statement is placed on the blackboard, and the usual blackboard procedure invoked (see §5.4.1)

5.3 Deriving Viewpoint Descriptions

So far we have discussed viewpoints as formatted descriptions consisting of sets of statements, without considering where these statements came from. In fact, the result of consultations with clients is invariably large quantities of both textual and non-textual material. This might include transcripts of interviews, notes, diagrams, and manuals. The contents of these need to be formalised to some degree, and this formalisation will involve a translation from the original informal representation to a more formal one. Many existing methodologies, both from knowledge acquisition (see Cordingley [1989]) and from requirements analysis (see Sommerville [1989]), have devoted attention to this translation process. Many such methodologies make explicit assumptions about the type and form of knowledge being collected, and provide a representational framework based on such assumptions.

For the viewpoints model, we deliberately make no assumptions about the knowledge being gathered, as we are not primarily concerned with the elicitation process. There are many existing elicitation methods which might be used to collect the knowledge initially, and different methods might be appropriate for different viewpoints. For the *Analyser* system, some basic tools are provided to assist with the task of interpreting textual material, and in particular, interview transcripts, which are assumed to be available on-line. These tools are based (loosely) on ideas drawn from speech act theory [Searle 1969], and are intended to complement the commitment reasoning scheme described earlier.

5.3.1 Interpreting Interview Transcripts

In linguistics, the study of pragmatics (see for example Levinson [1983]) has devoted much attention to the process of understanding utterances. In particular, speech act theory distinguishes between the illocutionary force and the propositional content of utterances [Kaplan 1989]. In the elicitation process, the illocutionary acts are limited in scope to actions of requesting, giving, receiving and denying information, where the propositional content of the utterance constitutes the information being exchanged. Hence a limited set of actions like those set out in section 5.2.4 suffice to represent the majority of utterances found in a dialogue between analyst and client. What remains is to determine the propositional content of the utterances.

Text, and dialogues in particular, can be divided into chunks, where each chunk focuses on a particular area of knowledge. The first task in processing the textual information is to break it up into chunks, and identify an originator for each chunk. The originator might be the speaker in a dialogue transcript, the author of a note, and so on. The chunks themselves will be of arbitrary size, typically containing a single sentence or a group of closely related sentences.

Each chunk of text can be interpreted into a set of propositions (“statements”) which act as a formal representation of the information contained in that chunk. These statements are then added to the appropriate viewpoint for their originator. Furthermore, the system records a reference link between the chunk of source text, and the statements in the viewpoint. Using these links it is possible to determine where each item in the viewpoints came from, and for any piece of source text, how it has been interpreted. Such links play a vital role in the validation of viewpoints. The

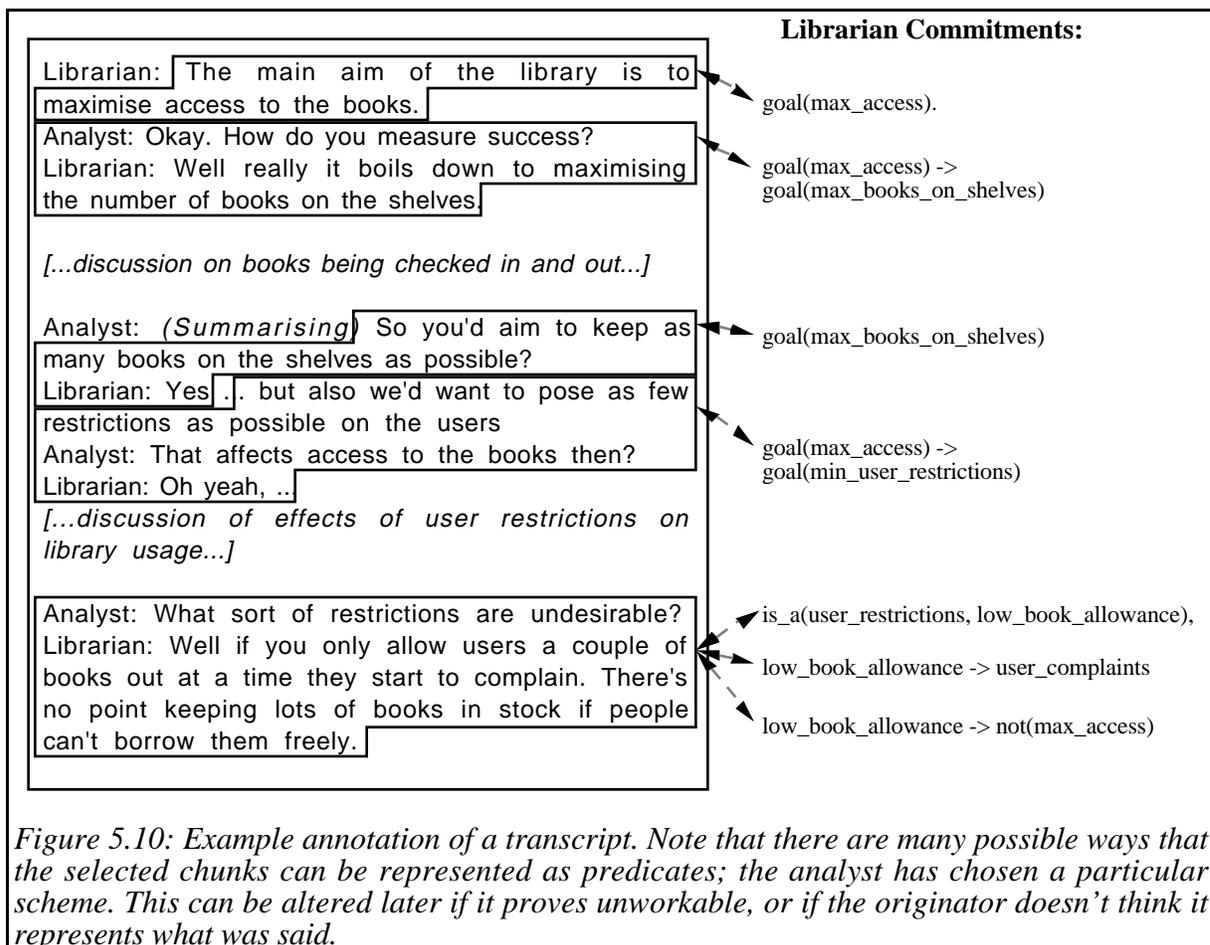


Figure 5.10: Example annotation of a transcript. Note that there are many possible ways that the selected chunks can be represented as predicates; the analyst has chosen a particular scheme. This can be altered later if it proves unworkable, or if the originator doesn't think it represents what was said.

entire process can be seen as one of annotation, in which the source text is annotated with formal interpretations (see figure 5.10).

For many utterances, there is no unique interpretation into a set of formal statements. The viewpoint model allows alternative interpretations to co-exist, as separate viewpoints can be used for the alternative interpretations. The interpretation can be changed if it becomes clear that the chosen interpretation deviates from the intention of the originator. This is possible due to the ability to retract commitments. Changing an interpretation is a compound action consisting of retraction of the original statements, and the assertion of the statements comprising the new interpretation.

In summary, the system does not attempt to impose any particular elicitation methodology, and it is anticipated that such methodologies could be used in conjunction with the system. The system does provide tools to support the process of annotating dialogues with formal interpretations. These tools make use of the analyst's skill at interpretation, merely supporting and documenting the process.

The result is a series of chunks of text, linked to the formal statements which represent them. Although such linkage is reminiscent of hypertext, the current generation of hypertext systems do not allow any automated reasoning with the information they contain. Instead, they implement passive networks in which the nodes are chunks of text which are impenetrable to the machine. The linking of formal objects to textual chunks is what Gaines [1989] terms *shadowing* (see §3.4.4.3) and has been used in a similar way in knowledge acquisition systems such as Cognosis [Woodward 1988].

5.3.2 Functionality of Annotation Tools

The *Analyser* system supports the annotation of protocols by allowing the user to select areas of text to be linked to viewpoint's commitments. The text files can be loaded as needed, and are assumed to contain transcripts of dialogues between clients and analysts, but can in fact contain any textual source material. The system keeps track of which files it has accessed, but cannot process the files in any way, except to record selected areas of text. Selections are recorded using the character positions of the start and finish, which means that files are assumed not to change. While this is appropriate for transcripts, if the source files are likely to change a more dynamic implementation of selections would be necessary.

Textual sources can be linked to relevant statements in the viewpoint descriptions in two ways. Firstly, areas of text can be selected and interpretations for that chunk of text added to the appropriate viewpoint using an appropriate representation. Commitments added in this way are linked to the selected chunk of text (see fig. 5.11). Note that this operation is appropriate for when the analyst has loaded a transcript for the first time and is processing it. The ability to add to the viewpoints without having to stop to consider inconsistencies is most useful during this activity.

Alternatively, commitments can be added to the viewpoints directly. In this case the user will be prompted to select a chunk of text as a source for the new commitment, although it is not compulsory to provide one. This prompting has two advantages: it encourages the user to consider what the viewpoint's originator actually said, and in so doing may prompt the user to consider other commitments arising from the same chunk of transcript.

The links can be traced from either direction. For example, the user can select a piece of text, and ask for display of all the commitment which are linked to it. Note that the division of the source text into chunks is not indicated when the text is displayed, hence the selected text might overlap with any number of chunks. Also, commitments can be traced back to the linked chunk of text. The system will re-load the transcript file if necessary, and highlight the relevant chunk.

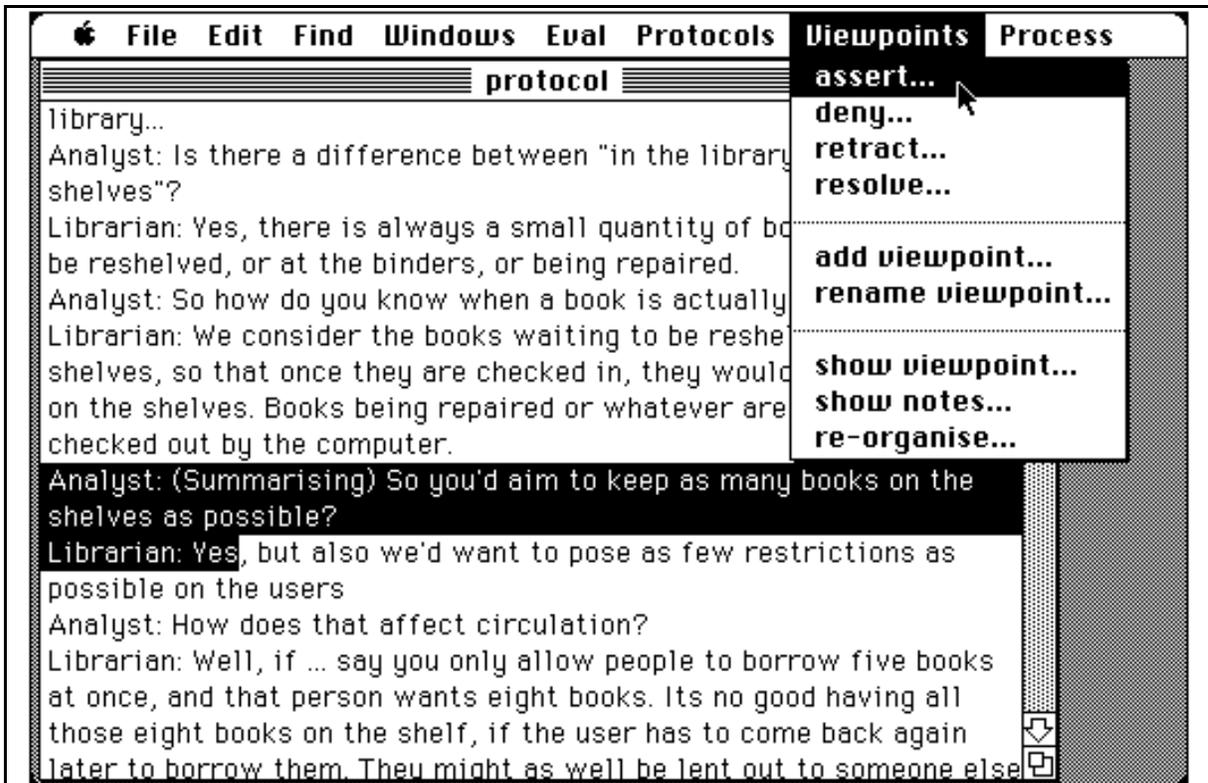


Figure 5.11: A Screen-dump from Analyser, showing the process of annotating a transcript. Any alterations made to a viewpoint will be linked to the selected text in the transcript file.

5.4 Integrating Viewpoints

So far we have considered how viewpoints are identified and what form they take. The development of a viewpoint does not occur in isolation from other viewpoints, and comparisons can shed new light on a particular perspective. Hence at any point in the process, parts of two or more viewpoints might be compared. The comparisons may involve conflict resolution, but are intended to be exploratory – any resolutions generated need not be incorporated in the final specification. Feedback from these explorations can be used in the development of the viewpoints, for instance to modify terminology, or to elicit information that the originator neglected.

The exploratory comparisons are made on a global blackboard. At any time the analyst can explore the current state of the viewpoints knowledge base by experimenting with the blackboard. The specification is built incrementally by the process of co-operation using the blackboard. The analyst acts as the scheduler, enabling any viewpoint to place an item on the blackboard for discussion, while the viewpoints act as representatives for their originators. This section describes the operation of the blackboard.

5.4.1 Blackboard

The system adopts the principle that anything in the specification must be agreed by all perspectives. The blackboard acts as a space in which the agreement can be checked on various statements. The rules that govern the operation of the blackboard are relatively simple: viewpoints place statements on the blackboard, which can then be altered or annotated by other viewpoints.

Once an item on the blackboard is agreed by all viewpoints, it can be removed from the blackboard and added to the specification. This rule requires a consensus to be reached before any item is added to the specification, although a viewpoint which has no information about some item (and hence no “opinion” on it) is deemed to agree to it. The analyst controls the entire process, initiating and halting it as necessary.

Any statement that a viewpoint is committed to can be placed on the blackboard. This will normally be in response to an action by the analyst, such as a query about whether some item can be agreed for the specification. Once on the blackboard, the statements can be amended or expanded by any other viewpoints.

Any viewpoint can object to the addition of a statement to the blackboard if that statement conflicts with the viewpoint’s commitments. The analyst can decide whether to allow the objection to stand, and if so, whether it replaces the original disputed statement. In the spirit of exploration, it is possible to allow both to stand, to explore the consequences of each possibility. This makes the blackboard inconsistent, but there is no problem, as multiple hypotheses can co-exist on the blackboard. Typically, the analyst will then examine the relevant evidence, pursuing further discussions with the viewpoint’s originators if necessary.

When a statement on the blackboard is not disputed by any viewpoint, we define it as *agreed*, and becomes part of the specification. As the specification only holds agreed statements, it could be regarded as the global ancestor viewpoint to be inherited by all other viewpoints. However, it is possible that statements in the specification are not commitments for particular viewpoints, and a viewpoint should only be committed to the assertions its originator has explicitly made. To prevent the agreed statements being imposed as commitments for all, the usual inheritance rules are relaxed so that the viewpoints do not inherit commitments from the specification. Any statements in the specification that a viewpoint does not hold as commitments are added to that viewpoint as suppositions, so that they may be discussed with the viewpoint’s originator if necessary.

There is a further complication caused by the fact that the development of viewpoints may continue while the specification is constructed. It is possible that new commitments cause a viewpoint to conflict with a previously agreed statement. Hence, all new commitments are first checked against the specification, before being directed to the chosen viewpoint. If there is a conflict with the specification, the disputed statement is returned to the blackboard. Also, originators may later retract commitments on which parts of the specification were based. For this reason, all items on the blackboard are linked to the viewpoints which placed them there, as are the contents of the specification. If a viewpoint subsequently retracts the commitments on which the item was based, the relevant part of the specification can be returned to the blackboard for renewed examination.

5.4.2 Exploration

The blackboard is used as a tool to create the specification. Its simplest use is to ask whether some assertion is supported by the viewpoints. Such a query results in the viewpoints being polled to discover whether any are committed to the item. If a viewpoint is found which is committed to the assertion, then the commitment is placed on the blackboard, and the blackboard procedure described above is invoked.

If no existing viewpoint is committed to the assertion, the analyst may explore further, by placing the queried statement directly on the blackboard. In order to keep track of the user’s explorations, an additional viewpoint is maintained, which contains all the exploratory assumptions made by the analyst while using the system. Any direct alterations to the specification or the blackboard made by the analyst become commitments in the analyst viewpoint. This viewpoint is treated similarly to any other: if any assumption is inconsistent with previous interventions, the analyst is warned using the same mechanism as the viewpoints. However, the analyst can choose to ignore

inconsistencies, causing this viewpoint to be split. This allows the user to explore alternatives and compare the results.

5.4.3 Agenda

The process of specification building is exploratory, and will proceed before all the necessary information has been gathered. Feedback from these explorations can guide the acquisition process. To assist with this process, *notes* can be added to viewpoints as reminders to the analyst that further discussion is required about some item, in much the same way that Adelson and Soloway [1986] observed that designers make notes to remind themselves to come back to some item at a more appropriate time. Taken together, the notes attached to any viewpoint can be displayed as an *agenda* to guide the development of the viewpoint.

Notes are generated by a number of events in which the viewpoint takes part. For example, if a viewpoint needs to be split to handle a conflict, a note is created identifying the commitment which caused the conflict. This will then serve to remind the analyst to check with the originator whether the conflict is genuine, or whether a mistake was made.

Notes are also created in response to events on the blackboard. A viewpoint will generate a note for itself if anything passes into the specification that the viewpoint had no information on. This provides a safe-guard that prevents any perspective being missed out in some discussion areas, and can also help in detecting when different terms are being used to describe some concept. The notes will remind the analyst to raise the point with the viewpoint's source. Similarly, a viewpoint will generate a note if the analyst overrides it on the blackboard.

5.4.4 Functionality of Integration Tools

The operation of the blackboard system in the *Analyser* is achieved through a set of commands available to the user via a menu. The commands form a means of interrogating the specification: through this process of investigation, it is sometimes necessary to invoke the blackboard procedures. These commands are as follows:

- specify(X) – checks what the specification has to say about the statement X. For example, in a logic-based representation, whether X is true or false according to the specification. If there isn't enough information in the specification to answer this (i.e. if X is unspecified), the blackboard procedure is invoked for the statement X.
- evidence(X) – This can be read as “what evidence is there in the currently active viewpoints that X might be true?”. All the statements contained in the viewpoints which could be used to determine the truth or falsity of the statement X are listed. Note that if X is on the blackboard this action can be used to find out which viewpoints support X and which oppose it.
- advise(X) – This can be read as “which questions should be asked of which originators to determine whether X is true?”. Any suppositions attached to viewpoints which would add enough evidence to determine the truth or falsity of X are listed. While in the simplest case, the analyst could just ask all originators about X, some originators might not know anything about X, and there may be some cost involved in contacting the originators. The advise command provides an indication whether there are any viewpoints from which X (or its negation) can already be inferred, or could be inferred given some other premise. This in turn indicates which originators are most likely to be able to offer information about X.
- explore(X) – commits the user to the statement X, so that it can be placed on the blackboard. The usual blackboard procedure is then invoked. Note that the system keeps track of all the statements the user is committed to through this command (see §5.4.2).

5.5 Conclusions

We have described the parts of *Analysers*. This section considers the system as a whole. Section 5.5.1 is concerned with the current implementation, and describes particular weaknesses associated with the implementation, which could be improved in a subsequent version. Section 5.5.2 describes the advantages offered by the system. There are inevitably a number of problems with the system: these are described in section 5.5.3. The main problem is that the blackboard mechanism is incapable of handling conflict. This problem led to the development of the system described in the next chapter.

5.5.1 Implementation

Analysers is implemented in PROLOG running on an Apple Macintosh. It has been tested on a number of small examples drawn from the requirements for a library system, and as used for illustration throughout this chapter. It was developed primarily in response to the problem described in section 4.2.5, which it handles adequately.

There are a number of weaknesses of the system, and in particular the blackboard proved unable to handle conflicts, for a number of reasons which we discuss in section 5.5.3. There are also a number of areas in which the current implementation might be improved which we discuss in this section. These areas have not been attended to in *Analysers*, as the entire system was subsumed into *Synoptic*, which is discussed in the next chapter.

As mentioned earlier, the only representation scheme supported at present is a first-order predicate calculus. While in principle it is possible to add new representation schemes just by providing set of rules for reasoning with them, this has not been attempted for *Analysers*, as it is not clear how other representation schemes can be accommodated on the blackboard, nor it is clear how multiple representations might be combined using the blackboard mechanism. A further weakness of the current implementation is that the only rule provided for conflict detection is to check for direct contradictions. Detection of conflict is a difficult problem, and it is likely that a collection of heuristics is needed. We have not attempted to develop such heuristics, but return to the problem of conflict detection in the next chapter.

Another area of weakness is in the *notes* facility: it is not yet clear when and how the notes should be reactivated. At present the analyst can list the current outstanding notes, in the order in which they were generated. Clearly, there are other orderings possible, the most obvious criteria being importance and urgency. One way of determining the relative importance of notes is by their type: for example, a viewpoint being overruled might be more important than a gap in the viewpoint's knowledge. In the current implementation, notes are typed according to how they were generated, but no tools are provided for sorting or selecting particular types of note.

5.5.2 Advantages

Analysers provides a degree of support for the multiple perspectives model described in chapter 4, and so has many of the advantages that were used to justify the model. It allows an analyst to build a knowledge base out of the elicited information, in which conflicts can be explicitly captured and represented. The presence of conflicts does not interfere with the process of reasoning with and interrogating this knowledge base. Furthermore, each piece of knowledge exists within the context of a viewpoint, and this context provides extra information about the reliability and applicability of the knowledge. The use of commitments emphasises the conversational nature of requirements analysis, and enables contributors to validate the knowledge they have provided.

The system addresses in detail the first two problem areas identified in section 4.3: identifying and developing perspectives. By developing hierarchies of viewpoints as the elicitation proceeds, there is no need to define the perspectives beforehand, and an initial set of viewpoints corresponding to

people is sufficient. The viewpoint hierarchies allow the analyst to explore possibilities within viewpoints. The level at which the hierarchy is viewed can be changed by making different descendants active, and this allows the analyst to explore what the viewpoint would contain if a particular option was chosen.

As conflicts are explicitly represented as splits between viewpoints, resolution of these conflicts becomes explicit, and the blackboard provides a focus for this process. The specification built through this process is linked back to the viewpoints on which it was based, so that the originator of any item in the specification can be traced. This linkage has a another benefit: it allows the viewpoints to build up areas of expertise within the specification. Although the boundaries of these areas remain fuzzy, it is possible to find out which viewpoints contributed to a particular group of statements. If a conflict arises over a statement related to this group, then the viewpoints which have the greatest expertise for the group can be consulted for their agreement.

5.5.3 Problems

The blackboard mechanism was chosen as a basis for the integration process, for its simplicity. Unfortunately, the model has not proved to be powerful enough, in that the co-operation is restricted to agreeing or blocking other viewpoints' suggestions. As there is no direct communication between the originators, they cannot seek creative solutions to conflicts, nor even reach any form of compromise through negotiation. The blackboard restricts how viewpoints can affect each others contributions.

Part of the problem seems to be that the blackboard model assumes perfect co-operation: there can be no conflict between the viewpoints. As has already been noted, conflict is inherent in the analysis process. Also, the requirement for complete consensus on any statement before it can go into the specification is too strong. It assumes that split viewpoints are either eventually reconciled, or the items which caused the split are not included in the specification. This is against our intuitive notion that disputes are likely to be important areas.

One of the problems that this implementation does not address is how to recognise and handle the use of different terminology by different people. Shaw & Gaines [1989] point out that this is a difficult problem when combining contributions from many people. There are, however, some mitigating factors in *Analysers*. Firstly we assume that to a certain extent the participants will recognise some of the instances of mismatching terminology, possibly during the translation into a formal representation. Secondly, viewpoints create notes as reminders to consult their originators when anything enters the specification for which they have no information; this will often trap situations where viewpoints use a different terminology. Other features could be added to ease the problem, for example, allowing viewpoints to define synonyms. However, these techniques do not constitute a satisfactory solution.

The problem of differing terminologies raises another question. Interpretation of natural language utterances into formal predicates involves the formulation of a suitable ontology. It is clear that different viewpoints will use different terms to build their description, and there might not be a simple correspondence between the sets of terms. Certainly there there is unlikely to be any pre-existing common ontology. However, in order to compare the viewpoints, and for communication between viewpoints, the descriptions must use the same terms. In fact, this ideal can be relaxed: there does not need to be a shared ontology, as long as correspondences between the separate terms can be found. The conflict resolution model described in the next chapter addresses these problems in more detail.

5.6 Summary

This chapter has described a system called *Analyser*, which implements part of the framework described in chapter 4. In particular, it concentrates on the identification and development of viewpoints, and introduces a method of handling conflicting knowledge by creating a hierarchy of viewpoints. Each viewpoint contains a description in some suitable representation, and has a unique originator.

The problem of identifying perspectives is handled by evolving distinctions between viewpoints as the descriptions are elicited. As perspectives are self-consistent areas of knowledge, inconsistencies reveal distinctions between them. Hence, initial viewpoints based loosely on sources of knowledge are split whenever they become inconsistent. Descendant viewpoints are created to isolate inconsistencies. These inherit a common description from the ancestor viewpoint. The result is that initial viewpoints are developed into hierarchies. Section 5.1 described the mechanisms by which the hierarchies are created.

As each viewpoint is consistent, it can be interrogated as a knowledge base, and inferences can be drawn from it. A distinction is drawn between statements actually made by the viewpoint's originator, which are termed *commitments*, and inferences drawn from those statements, which are known as *suppositions*. Viewpoints only contain commitments: suppositions may be associated with a viewpoint, but do not become part of the viewpoint unless they are confirmed by the originator. The descriptions can be in any representation scheme, provided that inference rules for that scheme have been added to the system.

The viewpoints contain formatted descriptions which doesn't necessarily translate directly from the language of interaction. For this reason *Analyser* provides a set of tools for processing transcripts. These record, for every statement added to a viewpoint, a link to a chunk of the transcript. Hence, the analyst and the originator can trace how statements in the transcript have been translated, and also where items in the viewpoint descriptions originate from.

Analyser uses a blackboard for integrating the viewpoints into a specification. Statements from the viewpoints can be placed on the blackboard, to be added to or amended by other viewpoints. When there is no disagreement, the statements are placed in the specification. The blackboard allows a certain amount of exploration, in that the analyst can initiate and control the operation of the blackboard. Viewpoints keep track of events on the blackboard, and notes are automatically created if their originators need to be consulted. For example a note is created if a viewpoint is over-ruled by the analyst, or if a viewpoint has no information about items that are added to the specification.

The main weakness of the system is the blackboard, which only supports a consensual approach to specification. The ability to detect conflicts is poor, and there is no mechanism for resolving conflict. Consideration of this weakness led to the development of a model of conflict resolution. The next chapter describes this model, together with a system called *Synoptic*, which extends *Analyser* to include a form of computer-supported negotiation.

6 Computer-Supported Negotiation

This chapter presents a model for the resolution of conflicts in the requirements engineering process. The model forms a part of the multiple perspectives model, and can be regarded as a form of computer-supported negotiation. Specifically, it addresses the last two problem areas identified in section 4.3, namely, comparing perspectives and resolving the differences between them. The intended use for the model is very specific: it allows the analyst to compare and merge previously elicited viewpoint descriptions, with assistance from the viewpoint originators. It is not intended to be a problem exploration tool, nor a form of meeting support, although the model may have applications in these areas.

A support tool, *Synoptic*, has been implemented to demonstrate the feasibility of the model. *Synoptic* displays elicited descriptions side by side, allowing the participants to compare and annotate them. Discrepancies noted by the participants are then used by the system to prompt for underlying assumptions and issues. In effect, the system provides guidance and clerical support as the participants break the conflict down into a number of components in order to propose options for resolution.

The model is prescriptive in that it acts as a set of guidelines, without being a rigid formal process. The tools which support the model are highly interactive, and are designed to provoke discussion of the conflict situations as much as elicit a suitable resolution. The model draws heavily on the behavioural approaches used in organisational psychology, and in particular takes note of the need to separate the people from the problem, in order to avoid the polarising nature of arguing from entrenched positions. There are three phases in the model: an Exploration phase, a Generative phase, and an Evaluation phase. These phases are discussed in detail in the rest of this chapter, after a discussion of the context for the model.

6.1 Conflict Resolution

In §3.3 we defined conflict as any interference in one party's activities, needs or goals caused by the activity of another party. In software engineering, the specification encapsulates the needs of the participants as a set of requirements. If two parties have opposing requirements, then any attempt to represent these requirements in the specification will give rise to conflict, as each would exclude the other. On the other hand, if one set of requirements is ignored completely, then a potential conflict has been suppressed. Depending on the actions of the 'injured' party, the conflict may resurface later.

The same principle holds for descriptions of the world ("domain descriptions"). In the case of requirements elicitation, part of the information elicited is a description of the system as it is at present. This includes both activities that may eventually be subsumed by the system, and the environment with which the eventual system must interact. These descriptions are rarely objective; opposing views which might not be expressed directly will manifest themselves as differences in these descriptions. Hence, conflicts will frequently be expressed as discrepancies between the viewpoint descriptions.

6.1.1 Sources of Conflict

It has been demonstrated that conflict, as defined above, is common in group interactions [Robbins 1989]. We can therefore assume that any application domain involving more than one person will

be subject to typical group conflicts. While it might be argued that a design process with a single goal, perceived in the same way by all participants, might be free of conflict, few real design processes are of this nature. This immediately suggests two possible sources of conflict in a real-world design process: conflict between the participants' perceptions of the domain, and conflict between the many goals of a design.

The extent of conflict in software engineering has recently been revealed by a major field study of software projects [Curtis, Krasner & Iscoe 1988]. Focussing on the behavioural aspects of software design, this study identified three major problem areas: the thin spread of application domain knowledge; fluctuating and conflicting requirements; and breakdowns in communication and co-ordination. Each of these problem areas is a source of conflict, and each depends crucially on communication between participants as a basis for any solution. A good conflict resolution approach necessarily emphasises communication between parties.

Conflicting and fluctuating requirements have many causes, from change in the organisational setting and business milieu, to the fact that the software will be used by different people with different goals and different needs. Handling constant change in requirements (which has been termed *requirements maintenance* [Finkelstein *et. al.* 1989]) requires an evolutionary approach that must be based on accurate capture of rationales and process information.

Unless the application domain with which the software deals is free of conflict, then the resulting software must incorporate this conflict. For small programs, the domain can be restricted until the conflict is excluded. For any large scale software, this is not practical. When the application knowledge is spread over many people, there is likely to be much disagreement between them, and fitting together the many contributions will inevitably lead to inconsistency.

Even if a domain appears to be free of conflict, quite often there will be areas in which there are different ways of looking at things. While such perspectives may not be fundamentally incompatible, they are likely to appear inconsistent, and so lead to conflict. Even if participants are describing essentially the same concepts, the style in which these are described may vary: even formal representation schemes allow enough variation in style so that there may be many different ways of saying the same thing.

Other sources of conflict include: conflicts between suggested solution components; conflicts between stated constraints; conflicts between perceived needs; conflicts in resource usage; and discrepancies between evaluations of priority.

The viewpoints model provides a basis for the study of conflict, by allowing viewpoint descriptions to be developed separately. Each viewpoint is a consistent description, so that conflicts are expressed as differences between viewpoints. Note that conflicts may occur within the requirements of a single person. As the viewpoints do not correspond to people, they take account of the conflicts between a single person's roles, as well as inter-person conflicts.

6.1.2 Consequences of Suppressing Conflict

As noted in chapter 3, existing software process models generally ignore conflict. This can lead to a number of problems. Where conflicts do occur, they are likely to get suppressed, because there is no means of expressing them within the framework. It is possible that these conflicts will remain suppressed, leading to dissatisfaction with the specification and the process that led to it. Often, a single perspective will be adopted as the basis for the specification at the cost of any alternative perspectives.

If these conflicts are eventually resolved, the resolution must be carried out outside the framework of the method and consequently is likely to be carried out at an inappropriate time, using an

undesirable means. In addition, resolution thus achieved is untraceable, making rationales invalid, and results irreproducible.

Suppression of conflict will have serious effects on the remainder of the software development process. In the worst case, suppressed conflicts may lead to the breakdown of the requirements process, or the withdrawal of participants. Failure to recognise conflict between the perspectives of the participants will cause confusion during the requirements phase, which will then continue throughout the lifecycle. The participants' understanding of the specification will differ, leading to further misunderstandings during design and implementation.

Research into group behaviour indicates that conflict can produce higher quality solutions [Brown 1988]. Certainly, exploration of the areas where participants descriptions differ can lead to a much better understanding of the domain. This is a strong argument for conflict to be carefully managed in the software process, with participants encouraged to express divergent views. This will ensure that the resulting system does not reflect just one point of view, and does not ignore concerns which interfere with the dominant concern.

In software design, effective collaboration is essential. It is vital that there be no losers from any conflict in the specification process, as the commitment of all participants must be maintained. Hence, encouragement of conflict must be matched with resolution methods which strive to satisfy all parties. An integrative approach should be adopted, to ensure that when divergent views arise they are incorporated into the process. The ultimate goal of the requirements process should be to produce a specification which represents all concerns.

6.1.3 Role of Communication

The need to maintain collaboration implies that any model for conflict resolution in requirements engineering must be based on collaborative modes of interaction (see §3.4.3). The two key collaborative methods for conflict resolution are (integrative) negotiation and education. Both of these emphasise communication between participants, and both greatly ease conflicts based on communication problems.

Communication between participants has an important role in conflict resolution. As Robbins [1974] notes, increased communication leads to decreased conflict up to a certain level, but that too much communication can lead to increased conflict. A possible explanation is that a certain amount of communication allows participants to discover commonalities, iron out perceived conflicts, and correct misunderstandings. However, a high level of communication highlights the details on which participants do disagree. Such conflicts are likely to be well-founded, and should not be discouraged. However, arguing over trivial details can be counter-productive, and so a balance must be struck between encouraging communication and devoting appropriate amounts of effort to resolution of particular differences.

Comparison of descriptions derived from different sources forms an important part of the process of eliminating errors. The multiple perspectives model facilitates this by separating the elicitation and comparison of descriptions into separate activities. In the comparison process, participants compare their own viewpoints with those elicited from others. By discussing areas of divergence, underlying assumptions are revealed, as are any omissions in the viewpoint descriptions. In so doing, problems of group-think are avoided, as each viewpoint is elicited prior to the comparison.

6.2 Conflict Resolution Model

The model of conflict resolution presented in this chapter differs from the blackboard approach described in §5.4 in a number of important ways. For example, resolution of differences between viewpoints does not necessarily result in a specification. The model allows parts of viewpoints to

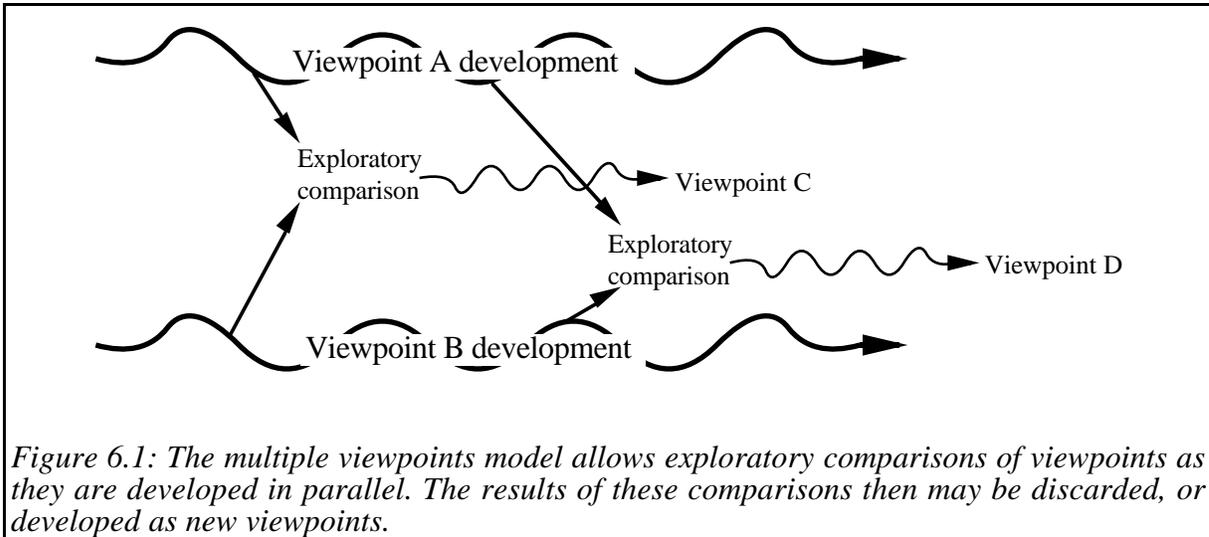


Figure 6.1: The multiple viewpoints model allows exploratory comparisons of viewpoints as they are developed in parallel. The results of these comparisons then may be discarded, or developed as new viewpoints.

be compared, while ignoring the remainder of them. This will result in an agreed description of a small part of the participating viewpoints. It then makes sense to regard this new description as a new viewpoint for two reasons: other viewpoints might still conflict with it, and later elaboration of the participating viewpoints may invalidate the resolution.

Furthermore, the blackboard approach attempted to merge viewpoints automatically, involving their originators only when problems arose. In recognition of the difficulties of conflict resolution, and the need for creative solutions to difficult conflicts (see §3.4.5), the conflict resolution model described here is an interactive process. The initial comparisons might be made by the analyst, but in many cases the analyst will only be able to speculate about the reasons a perspective was described in a particular way. Hence, the process is likely to involve the originators directly, in comparing their viewpoints with others. For this reason, the model is termed *computer-supported negotiation*.

6.2.1 Resolution Context

The process of knowledge acquisition never stops. Throughout the lifecycle of the system, the users' perspectives will change, and so will their needs. On the other hand, the elicitation of a viewpoint does not occur in isolation from other viewpoints, as comparisons can shed new light on a particular perspective. Hence at any point in the elicitation process, parts of two or more viewpoints might be compared.

The comparisons between viewpoints may involve conflict resolution, but are intended to be exploratory – any resolutions generated need not be incorporated in the final specification. Feedback from these explorations can be used in the development of the viewpoints, for instance to modify terminology, or to elicit information that the originator neglected. The results of any exploratory integrations can be treated as new viewpoints which can continue to take part in the development process (Figure 6.1). Such derived viewpoints effectively represent coalitions of perspectives, which have been shown to arise in software projects [Curtis, Krasner & Iscoe 1988].

The modelling of viewpoints allows differences between perspectives to be captured and accommodated. If only a single description was maintained, differences between parties would tend to be avoided or suppressed, and often go unnoticed. As the viewpoints contain formal descriptions, it is possible to combine parts of different viewpoints to reason with, and detect inconsistencies. This process of parallel development of viewpoints – with exploratory integrations being initiated at any point – provides the context for the conflict resolution model.

6.2.2 Detection of Conflicts

The first problem for conflict resolution is to recognise that a conflict exists. This might be harder than it seems for a number of reasons. The terminology used by the participants is unlikely to match exactly [Shaw & Gaines 1988], and the styles in which knowledge about an issue is expressed will differ. This difference may be because of different representation schemes, or different descriptions within the same representation scheme. Also, participants will have different areas and different amounts of knowledge, making it difficult to make comparisons. These problems make it hard to tell where participants are agreeing, let alone where they are disagreeing.

The definition of conflict in §6.1 was based on interference: two parties are in conflict if the activities of one adversely affect the interest of another. Hence, viewpoints are free to differ, and only conflict when that difference matters for some reason, leading to interference. There are a number of situations in which the differences matter:

- ⌘ When viewpoints need to be compared.
- ⌘ When there is a need to reason with knowledge from several viewpoints.
- ⌘ When the originators insist their viewpoints are “better” than others (and so perhaps should be adopted at the expense of them).
- ⌘ When a coherent description is needed for further progress.

Under normal circumstances, differences between viewpoints are ignored, allowing them to develop independently. By only entering the conflict resolution process when differences between viewpoints matter, we avoid attempting to resolve conflicts unnecessarily. A conflict, then, is simply a difference that matters.

Note that defining conflicts as differences that matter will include many things that might not normally be regarded as conflicts. The distinction that Deutsch [1973] draws between real and apparent conflicts is deliberately ignored. Apparent conflicts here might include: where one party has misunderstood another’s position; where viewpoints use different terminology to describe the same thing; and where the interests at stake do not interfere, and can be combined directly. All these are treated as conflicts, and part of the task of the negotiation process is to identify what type of conflict has occurred, and hence whether or not a resolution is needed. The rationale for this approach is simple: it is impossible to tell without exploration whether a conflict is real or apparent, and this exploration is an essential step in the conflict resolution process anyway. The first phase of the model encourages participants to talk around the conflict, allowing any missing information to be gathered, and the participants to learn about each other’s positions. This may even cause the conflict to disappear.

6.2.3 Example Conflicts

Examples of conflicts in domain descriptions are not hard to find. Throughout this chapter, two such conflicts, in the domain of the library system, will be used to illustrate the working of the conflict resolution model.

The first of these is taken from the literature, and concerns the case study used at the fourth IEEE International Workshop on Software Specification and Design (see figure 4.5). Many of the papers presented at the workshop analysed this case study, giving descriptions of the domain. However, the descriptions differ in various ways, having been developed independently, usually with a perspective chosen to illustrate the authors own work. Excerpts from two such papers are given in figure 6.2, both of which use dataflow diagrams in different ways.

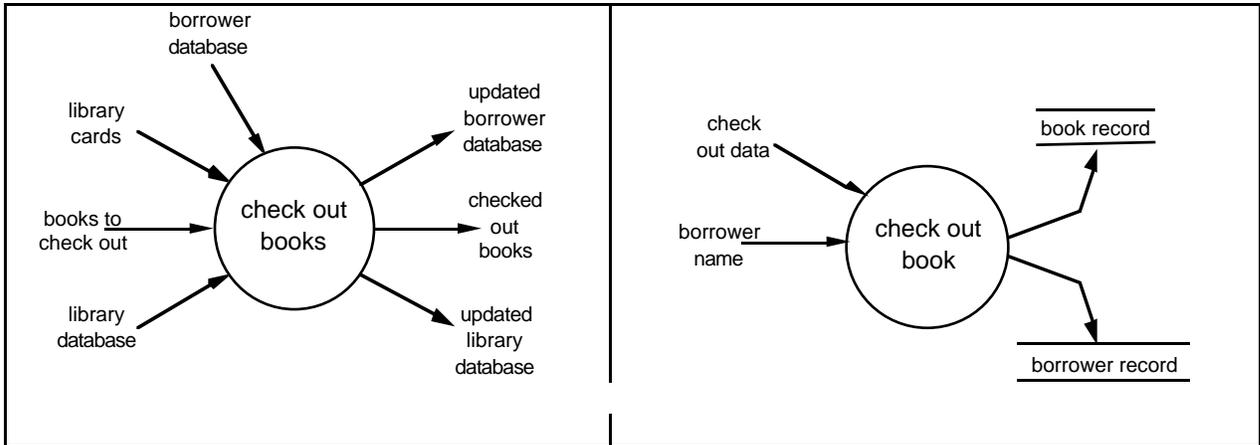


Figure 6.2: Excerpts taken from dataflow diagrams for the library case study at the fourth International Workshop on Software Specification and Design. (a) is from Lubars [1987], p68 and (b) is from Kerth [1987] p183.

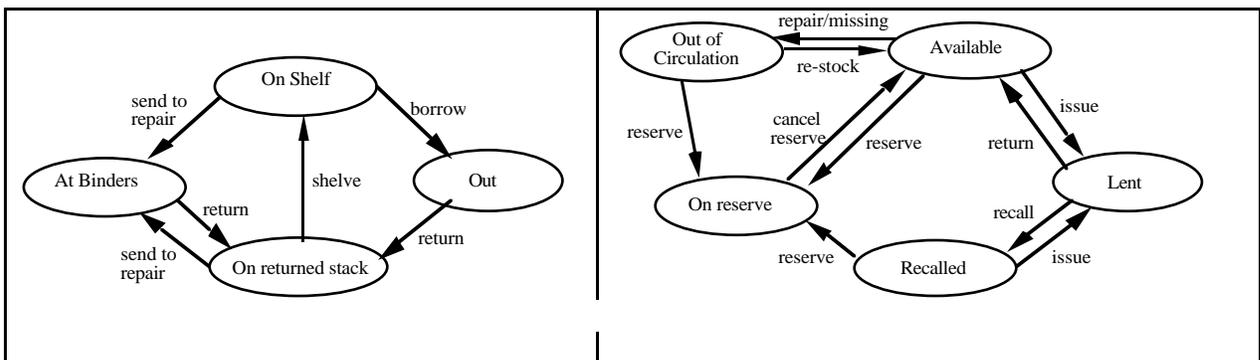


Figure 6.3: The possible states for a library book: (a) from the perspective of physical whereabouts; and (b): from the perspective of accessibility of a book.

A more complete (but hypothetical) example is shown in figure 6.3. Two possible state transition diagrams for the books of the library are given, which may have been elicited from two different librarians. One gives a description based roughly on a book's physical whereabouts, whereas the other gives a description based on how a book can be accessed. While it may appear that there are a number of correspondences between the two descriptions, these are not as simple as they seem. For example, the concepts ON SHELF and AVAILABLE are similar, except that the latter includes books waiting to be shelved: it assumes that librarians are able to locate unshelved books for loan. OUT and LENT are also similar, except that the former includes books being used within the library, while the latter only includes such books if they are from the reserve collection. To make things worse, both could have used the same terminology.

6.3 Exploration Phase

The first phase of the model is exploration. The aim of this phase is to arrive at a better understanding of the conflict. Essentially, this is a process of knowledge elicitation, as additional knowledge is needed about the descriptions in conflict. This phase involves identifying why the conflict occurred, and hence the type of conflict, the extent of the conflict, and the issues involved. Such information might be represented in a number of ways ranging from formal to informal,

through a process of annotating the descriptions and linking elements of them together. In particular, links showing correspondences and discrepancies can be used.

The exploration phase begins once a conflict has been detected. The information available consists of the relevant parts of the viewpoint descriptions, and an indication of where the conflict was detected. For example, given the descriptions in figure 6.3, let us say that the analyst is trying to establish when a book is available for loan. The states `ON SHELF` in figure 6.3(a), and `AVAILABLE` in figure 6.3(b) seem to correspond roughly, but there is conflict, as neither the names, nor the transitions attached to these states match. In this case we begin the exploration with these two diagrams and an indication that the conflict is between `ON SHELF` and `AVAILABLE`.

The result of the exploration phase is a map of the conflict. This includes a list of correspondences and differences between the descriptions. In other words, the original disparity between viewpoint descriptions has been broken down into specific lists of items in the descriptions which correspond, and items which do not. Together, these comprise the *components* of the conflict. The exploration phase also elicits any issues underlying the correspondences and differences, and for each issue, criteria for its satisfaction.

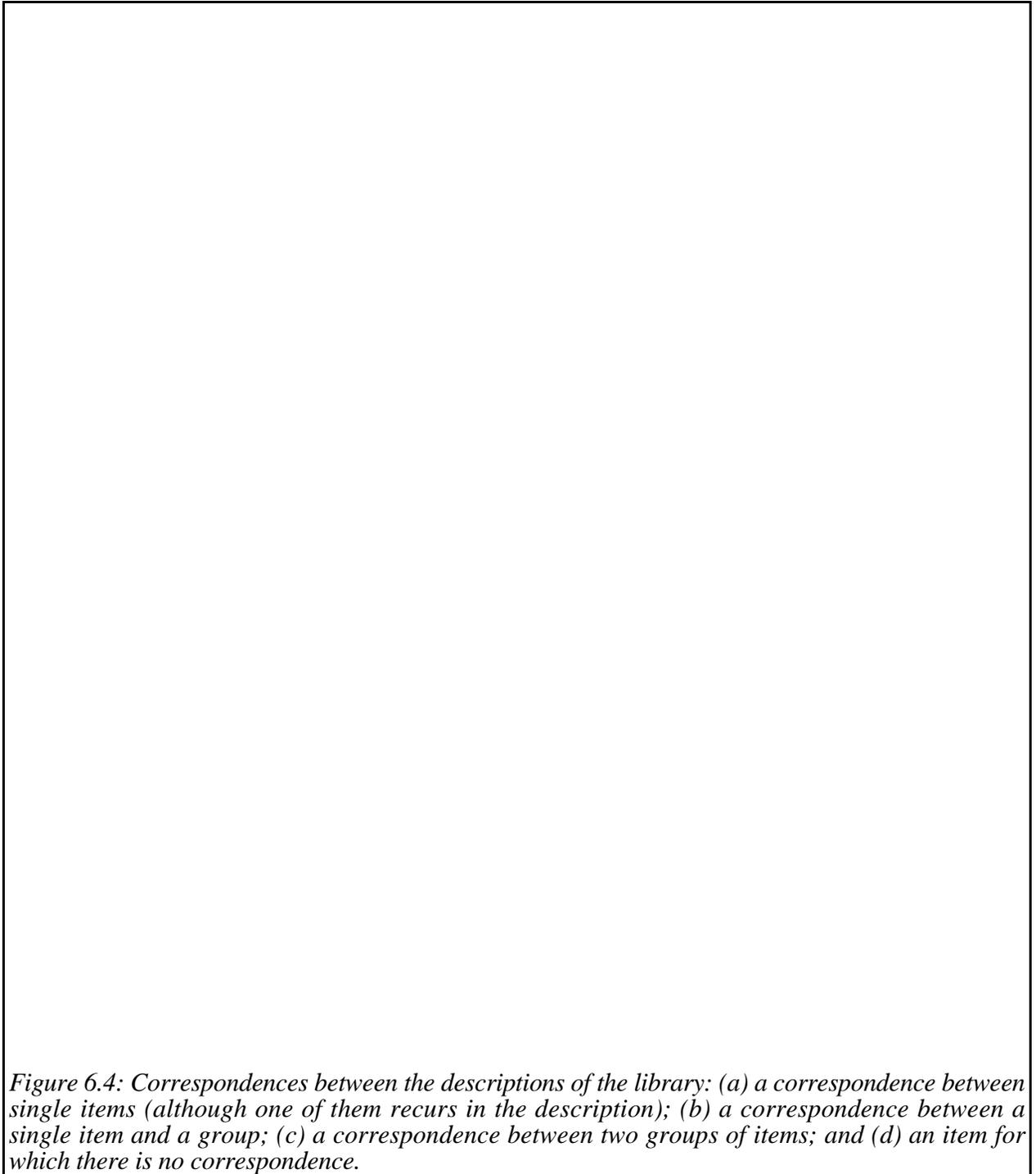
6.3.1 Establishing Correspondences

The first problem is to establish some common ground between the descriptions. This is important to delimit the extent of the conflict, and to provide the participants with a basis for communication. The process starts with two descriptions, within which particular statements are known to conflict. To determine the extent of the conflict, the statements around the conflicting ones need to be compared, as these provide a context for the conflicting statements. Initially, this context consists of those statements in the original viewpoints which are directly connected to the ones in question. In a graphical notation, these are the arcs and nodes connected to the items in question, and for a chain of inference, all the immediate antecedents and consequences are used.

The participants begin by identifying correspondences between the items in the descriptions. Such correspondences may be exact or approximate. There is an exact correspondence if the items are agreed as having the same definition; the correspondence is only approximate if the meanings are similar, but differ in certain details. Many terminological differences will be discovered at this point: methodologies for recognising terminological mismatches, such as that of Shaw & Gaines [1988], as well as tools for detecting graph isomorphism, could be usefully employed here. The tool support provided by *Synoptic* is somewhat less sophisticated than these, relying on the user to identify correspondences.

To illustrate this process, consider the library books example. The originators of the descriptions compare the arcs attached to the states `ON SHELF` and `AVAILABLE`. Firstly, they might note that there is a correspondence between `SEND TO REPAIR` and `REPAIR/MISSING`. The correspondence is not exact but it appears that the former is included in the latter. This raises the issue of how books which go missing are handled in the first viewpoint, and whether this needs to be represented. The actions `BORROW` and `ISSUE` appear to be identical, but note that the return action in one diagram is the inverse of issue, while in the other, it leads to a new state, `ON RETURNED STACK`. In this case we can assume that the state `AVAILABLE` in the second diagram is the composition of `ON RETURNED STACK` and `ON SHELF`. Other correspondences can similarly be found, and items may be involved in more than one correspondence.

The examples described demonstrate a number of different types of correspondence (Figure 6.4). Most obviously there is equivalence, as in the case of `ISSUE` and `BORROW` (Figure 6.4a). Often, different terms will be used to describe the same thing from different perspectives. In this example, one is the name of an action described by a librarian, and the other the same action described by a borrower. Both terms are useful, and could be recorded as synonyms. The comparison raises the issue of which should be used where.



As well as correspondences between single items, frequently groups of items will be linked. Where a single item in one description corresponds to a group in another, the representations are at different levels of decomposition (Figure 6.4b). This is a common problem in systems analysis, as there is no standard way of deciding whether different parts of a description are at the same level of abstraction. The measure of level of abstraction is a subjective one, and different analysts will decompose the parts of a description in different orders. In many cases, such correspondences will not be exact, as decomposition usually reveals details about a description not considered at a

coarser grain. Again, such comparisons yield issues that one description may not have addressed, which could be usefully discussed.

Correspondences between a group of items in one description and a different group of items in another description reveal where different types of decomposition have taken place. For example, the states `ON SHELF` and `ON RETURNED STACK` in the first description correspond to the group `AVAILABLE, ON RESERVE` and `RECALLED` in the second (Figure 6.4c). In this example, both groups are decompositions of “In the library”. The two groups will not necessarily match exactly. For example, the `RECALLED` state seems to include recalled books both before and after they are returned, and so is not totally captured within the group `ON SHELF / ON RETURNED STACK`. Different decompositions reveal different concerns within the system modelling process, in a similar way to Feather’s parallel elaborations [Feather 1989a].

Finally there is the case where an item or group of items in one description has no correspondence in the other. This may be because it has been omitted, or because the role played by such an item has been filled in other ways. For example, the `SHELVE` transition in the second description has no correspondence in the first (Figure 6.4d).

The result of this stage is a list of correspondences between items in the viewpoints. Each correspondence may be recorded as exact or partial; but note that exact correspondences do not imply identical structure. The former indicate where there is agreement, and so restrict the area of conflict. However, where the correspondence is partial, there is still conflict to be resolved. In effect, the conflict has been broken down into its components: the initial rough description is replaced with a list of specific disparities between items in the descriptions.

6.3.2 Identifying Conflict Issues

For a conflict to be resolved constructively, the reasons the parties are in conflict must be ascertained. These reasons may vary from lack of communication and poor understanding of other viewpoints, to differences in priorities and areas of concern [Robbins 1989]. Often the actual conflict is unrepresentative of the real issues which led to the conflict. Deutsch [1973] calls these *displaced conflicts*, and discusses the psychological reasons which cause people to express conflict in indirect ways. Although Deutsch’s considerations are mainly to do with conflicts in personal relationships, there are other reasons why conflicts may appear displaced. For example, the descriptions being compared might be the result of long chains of development which are not necessarily based on the same initial assumptions and motivations.

Easterbrook [1989] notes that simply asking people to state any assumptions made by their descriptions is unlikely to be fruitful. There will be many assumptions, goals, and motivations involved in any description, some very trivial, and only a few will be relevant to the analyst. They are idiosyncratic in that what is obvious to one person may be an important decision to another [Kaplan 1989]. Discussion of assumptions must be prompted in some way, by asking “Do you assume X?” rather than “What do you assume?”.

The systems `gIBIS` [Conklin 1989] and `Argnoter` [Stefik *et. al.* 1987] made use of the notion of *issues*, which are simply points that the design needs to address. They may take the form of suggested requirements (e.g. “The check-out process should ensure the borrower has not taken too many books”), or questions which need to be resolved (e.g. “How many books is too many?”). However, in these systems, issues are elicited unprompted: in `gIBIS` as a prelude to identifying positions and in `Argnoter` as supportive arguments for proposals. Our approach is to elicit issues only as a response to specific conflicts. Conflict provokes discussion of the issues, as participants raise any issues they feel other party’s descriptions neglect. This prompting avoids having to ask participants simply to list any assumptions they made. It also avoids time wasted discussing issues on which there is already agreement, or which are irrelevant to the current context.

To assist with the elicitation of issues, four types of free-text annotation may be attached to the items in the descriptions, and to the correspondence links between items:

Comments - these are general purpose annotations, which can be attached to any item or group of items in the conflict. A typical use would be to attach to a correspondence between items to suggest a reason for the difference or similarity of items. Example: A comment might be attached to the state `ON RETURNED STACK` noting “librarian B’s description does not include a returned stack”.

Assumptions - these are like comments but allow the user to note where a description appears to make some unstated assumption. These often arise when two descriptions are compared, and the comparison reveals issues that have been neglected in either description. Example: An assumption may be attached to the comment above, to note that “librarian B’s model assumes that books waiting to be shelved can be located for loan”.

Issues - these are points that need to be addressed. There are many circumstances under which issues arise, but often comments and assumptions will result in an issue. Example: the assumption above might lead to the issue: “How can books that have been returned but not shelved be traced?”.

Justifications - These are added to support a particular viewpoint or proposal. Often these will be added in response to assumptions and comments to provide a rationale for the original item. They will also be added in the next two phases of the process, to relate solution components to issues.

Several of the examples in the previous section showed how issues arise during the comparison of descriptions. Typically, issues are prompted by the creation of a correspondence, and the supporting tools prompt the user to note any assumptions or issues that arise when creating a correspondence. Assumptions have an issue attached automatically, questioning whether the assumption is reasonable, to ensure that the assumption is discussed when the issues are considered later in the process.

6.3.3 Agreeing Resolution Criteria

The final part of the exploration phase is the establishment of criteria by which to judge possible resolutions. Fisher & Ury [1981] suggest that objective criteria should be agreed before any resolutions are generated, to ensure that an agreement can be reached. This is to prevent participants moving the goal posts to get their personal preference accepted. However, it is often not clear before any solutions are proposed what the criteria should be. It is an open question as to how the prior agreement of criteria affects the generation of a resolution.

Our model treats the establishment of criteria as part of the exploratory phase, as it involves elicitation of further information from the participants. The criteria allow potential resolutions to be judged and compared. As the measures by which participants evaluate their satisfaction, they represent the participants’ goals for the resolution process. As such, these are the final part of the picture of the conflict built up by the exploration process. Issues represent the key points in the conflict; criteria show how the participants feel about these key points.

In our model, every issue has a related criterion describing desirable outcomes for that issue. In the simplest case, the issue asks a question, and the criteria attached to it provides an answer (often approximately), by reference to the participants’ concerns. Effectively we treat issues as points that the original viewpoints left unclear, and the criteria as the clarification of these points. In most cases, the participants will agree on the criteria for an issue with little difficulty, reserving disagreements for the order of priority among criteria. However, there are issues on which participants will define opposing criteria, and in this case, both are recorded, and the dispute added to the list of items in conflict.

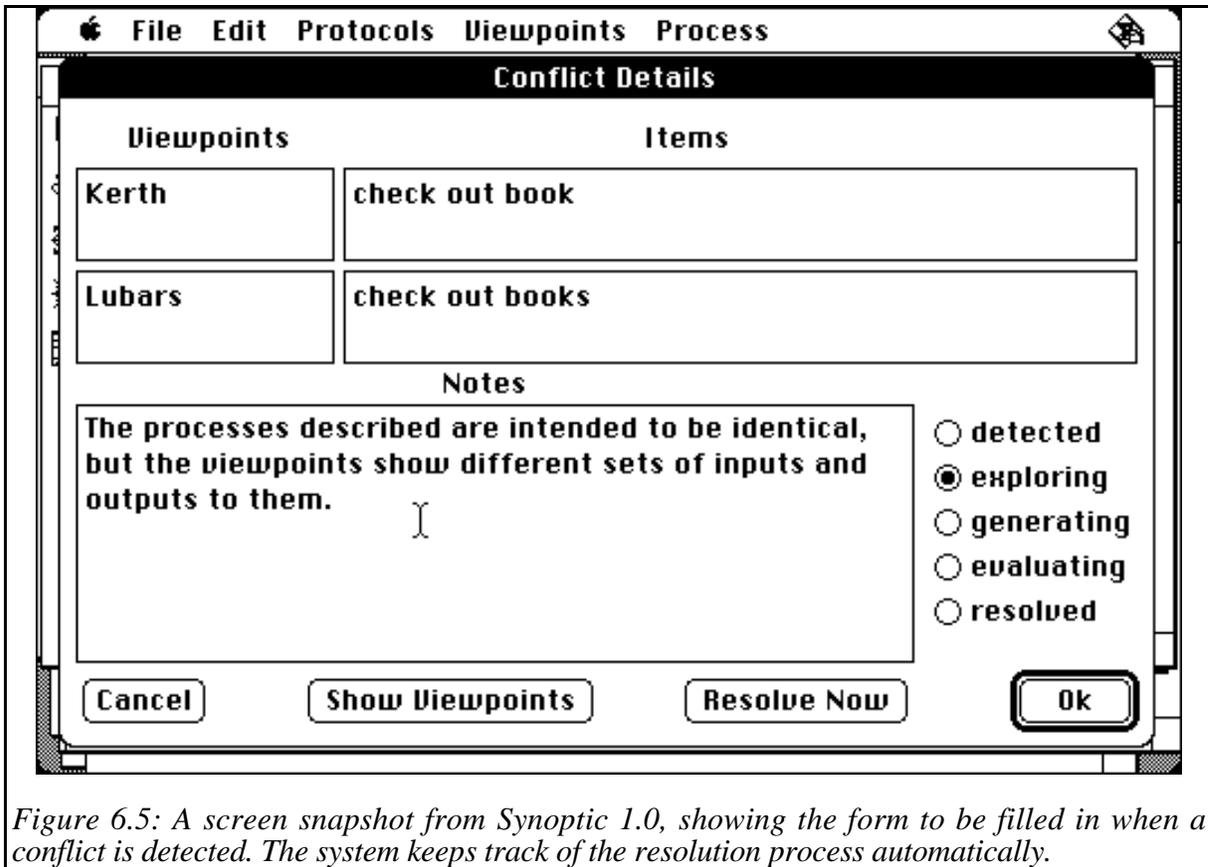


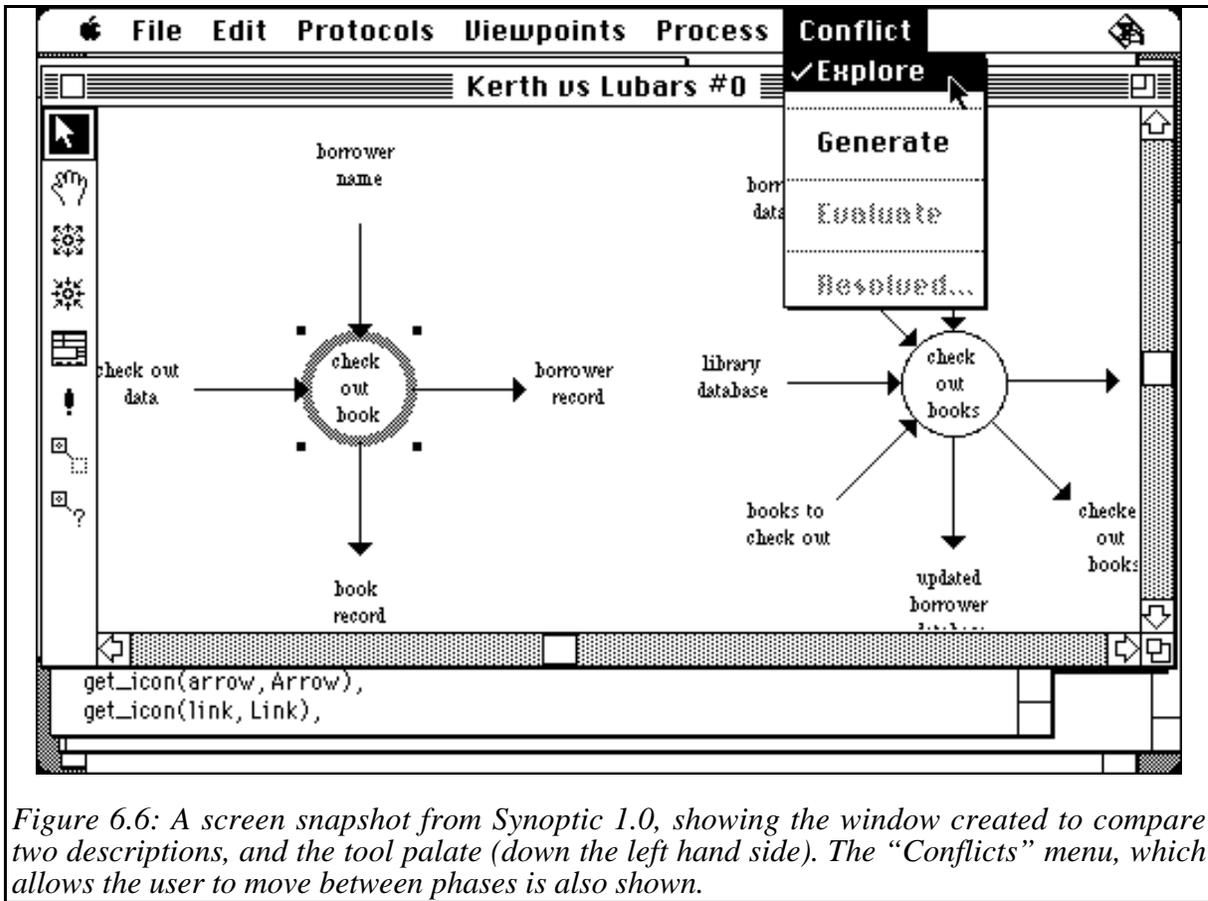
Figure 6.5: A screen snapshot from Synoptic 1.0, showing the form to be filled in when a conflict is detected. The system keeps track of the resolution process automatically.

Criteria can be used by participants to object to an issue. Issues are elicited in response to conflicts, and so will usually be agreed as being valid, if only because they are important enough to disagree over. However, occasionally, participants will object to an issue on the grounds that it is not really an issue, or is irrelevant. In this case, they can attach a null criteria, effectively stating “this issue can be ignored (in my opinion)”. Normally a null criteria will have either an assumption or a justification attached, explaining why. For example, a participant may object to the issue “There must be a way of locating unshelved books” because books can be assumed to be unavailable until shelved.

6.3.4 Functionality of the Exploration Tools

Synoptic is an extension of the *Analyser* system described in chapter 5. It retains all of the functionality of the viewpoint manipulation tools, with the exception of the blackboard system described in section 5.4. This is replaced with a set of tools to support the conflict resolution model. A single menu selects which phase of the model is in operation, and within each phase a palate of tools is available.

Conflicts between viewpoints can be noted by filling in a conflict form, as shown in figure 6.5. When a difference between viewpoints needs to be resolved, the conflict resolution process is invoked by selecting the exploration phase from the conflict menu. The same menu is used to move from one phase to the next, and to move back to a previous phase if necessary. The display of this menu is modified to show the current state: completed phases are marked with a tick, while phases beyond the next are shaded to show they are unavailable (see figure 6.6).



When noting a conflict, the user is asked to select those items in the viewpoint descriptions which are in conflict. In the exploration phase, these items, together with their immediate context are displayed side by side in a ‘synoptic’ window. A palate of tools is attached to this window to allow the following operations:

- Selector (arrow icon) - for selecting items within the displayed descriptions, for some subsequent action, such as attaching a note. The selected items are displayed in grey.
- Mover (hand icon) - for moving a displayed description around. As items can be added or removed from the displayed descriptions, it may become necessary to adjust their relative positions within the synoptic window.
- Extend description (explode icon) - This tool extends descriptions in the synoptic window by adding more items from the source viewpoints. For any selected item in the synoptic window, all immediately connected items in the viewpoint description that are not already displayed in the synoptic window are added.
- Trim description (implode icon) - This tool allows the user to trim items from the descriptions displayed in the synoptic window. Items that are listed as part of the conflict on the conflict form cannot be trimmed.
- Conflict form (form icon) - This displays the conflict form.
- Attach note (exclamation mark icon) - This tool allows the user to attach a note to any item or link. The user will be asked to select the type of note to attach (see §6.3.2). Each type of note has a form to fill in. In the case of issues and assumptions, the form has

slots for criteria and justifications. for any type of note the user is prompted for a brief title by which the note can be referred.

Create correspondence (link icon) - A correspondence is created between the selected items. The user will be asked whether the correspondence is exact or approximate, and will be prompted for any issues to attach.

Find correspondence (link-question icon) - Displays any correspondences involving the selected items.

These tools allow the basic actions for the exploration stage. Note that *Synoptic* only plays a supporting role in this phase; the emphasis is on the human participants recognising and discussing correspondences. However, the tools perform clerical duties such as recording correspondences. Guidance is provided in that the system keeps track of task completion. For example, the system checks that all forms are filled in fully before allowing the user to move on to the next phase.

6.4 Generative Phase

The result of the exploration phase is a “map” of the conflict, which can be used to guide the search for possible resolutions. The second phase is concerned with generating these resolutions, and is essentially a design process. The aim is to propose solutions which overcome the limitations of the original viewpoints, and respond to the issues identified in the exploration phase. At this stage, the options are not evaluated, nor are they checked against the issues. This prevents the creative process being stifled by pragmatic considerations [Stefik *et. al.* 1987]. The options might be generated in a variety of ways, from directly combining elements of existing viewpoints to techniques such as lateral thinking and brainstorming.

The result of the generative phase is a list of options for resolution. These options are not intended to be complete resolutions, but might be combined in various ways to arrive at one. It is also possible that some of the options will be incompatible with one another: the evaluation phase will examine how the options can be combined.

6.4.1 Types of Conflict

Before the generative process gets underway, it is useful to characterise the type of each component of the conflict revealed by the exploration process. This will help to decide what form the generative phase will take, and what a possible resolution might consist of. We can identify three broad categories of conflict that might arise in systems analysis, as follows:

Conflicting interpretations - descriptions of the current situation or the current requirements do not match, usually because different perspectives interpret things differently. This category corresponds to the category *Beliefs* (or “how things are”), as described by Deutsch [1973].

Conflicting designs - suggestions (or partial designs) for how the system should be do not match. This roughly corresponds to the Deutsch’s category *Values*, or “How things should be”. While a requirements specification would not normally be expected to contain design information, participants are likely to express some of their requirements as partial designs, representing their preconceptions of the system.

Conflicting terminologies - the terms in which things are described do not match. This covers the communication problems suggested by Robbins [1989] as being a major cause of conflict.

In addition to these three categories of conflict, a scale for the severity of the conflict is used. This ranges from *non-interference* at one end to *mutual exclusion* at the other. The former implies the items in conflict can be combined directly without compromising either, whilst the latter indicates that each totally negates the other, and only one can be used (Figure 6.7).

Using this schema, conflicts identified as non-interfering can be eliminated from further resolution work, as the direct combination of the two viewpoints provides an instant resolution. Where the two viewpoints provide alternative views or alternative terms, the circumstances under which each should be used still needs to be examined. For the remaining conflict types, there is plenty of scope for the design of novel resolutions which circumvent the conflict, by satisfying the underlying issues in other ways.

Table 6.1 describes typical examples of each of the categories and levels of severity, together with examples from the library books conflict. The examples are from the list of specific correspondences and differences discovered in the exploration phase. Some of the examples are phrased in a way that suggests possible resolutions; consideration of where these conflicts should appear in the table helped identify potential solutions. Note that these individual options are not exhaustive, and may obscure other possibilities. For example, exploration of this particular conflict revealed that one viewpoint was concerned with the physical whereabouts of a book, while the other is describing accessibility: it might make sense to retain both viewpoints entirely, and record for each book both its physical whereabouts and its loan status.

6.4.2 Generating Resolution Options

The model does not prescribe a particular method of generating resolutions. As already noted, this is a design problem, and design methods are already well documented elsewhere (e.g. Finkelstein & Finkelstein [1983]). A range such methods might be usefully employed for generating options, depending on the components of the conflict, and the form of resolution required. Consideration of the category of the conflict components, as described in the previous section, may immediately provide one or more resolution options.

The categorisation of conflicts helps to determine what form a resolution should take. For example, conflicts in terminology, once detected, can be resolved fairly simply. Participants can be prompted for distinguishing terms if the same terms have been used for different concepts. Each such suggestion is a possible resolution, to be evaluated in the same way as other proposals. Where different terms have been used for the same concept, it is likely that both will be useful, and proposals will include circumstances under which a particular term might be favoured. In many cases, negotiating terminological differences is a waste of time, and participants should agree to

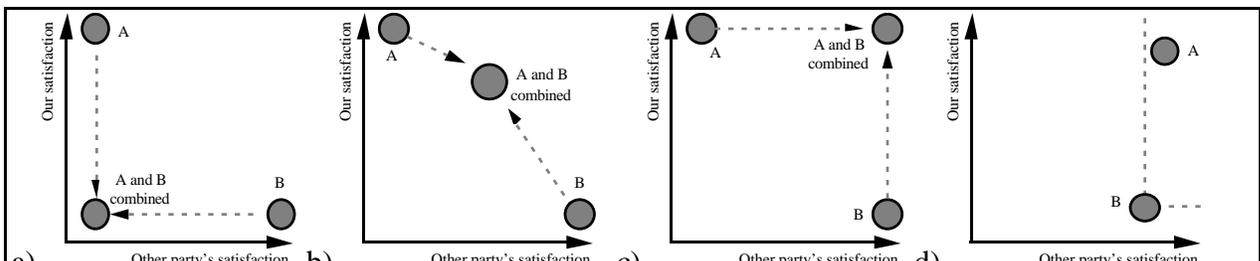


Figure 6.7: These diagrams show conflicts of different severity. In (a) the viewpoints are mutually exclusive, as their combination satisfies neither party (the combination might not even be possible). In (b) the viewpoints can be combined, but with some loss of optimality for each party, and in (c) the viewpoints are non-interfering and can be directly combined. A variant of the non-interfering type is shown in (d), where one of the viewpoints already satisfies the other's concerns.

	Conflicting Interpretations	Conflicting Designs	Conflicting Terminology
non-interfering	Either interpretation can be used without affecting the other (need to find out which to use when). <i>Example: the possibility of books going missing has been omitted from the first viewpoint, and could be added directly if necessary.</i>	The design can be directly combined without compromising either. <i>Example: The recall facility, which is assumed to be a design suggestion, could be added directly to the first viewpoint</i>	Different terms have been used for the same concept (need to find out which to use when). <i>Example: "borrow" and "issue" apply to the same action. A borrower is more likely to use the former term, and a librarian the latter.</i>
partially interfering	Interpretations are not wholly consistent, and if both are to be used, some resolution is required. <i>Example: The "shelve" action is not wholly consistent with the second viewpoint as "available" does not quite correspond to "on shelf".</i>	Designs can be combined but interfere, and the direct combination may not be the ideal resolution. <i>Example: A reserve collection could be added to the first viewpoint by splitting the "on shelf" state to indicate the type of shelf.</i>	The same labels have been used for similar concepts. The differences need to be resolved. <i>Example: "Out of circulation" and "At binders". The latter is more specific, and implies that these books will eventually return.</i>
mutually exclusive	Interpretations totally contradict one another, and cannot be used in conjunction. <i>Example: There is no "return" action for recalled books in the second viewpoint, contradicting the notion of a returned book stack.</i>	Designs are completely incompatible, or tend to negate one another when combined.	The same labels have been used for different concepts, and some distinguishing terms are needed. <i>Example: The "return" from "at binders" is indistinguishable from the "return" from "lent". These might be completely different actions.</i>

Table 6.1: Different types and severities of conflict, and for each a description of the kind of situation covered, and an example from the library books conflict.

differ, or make the effort to translate. It may be sufficient just for the participants to be aware of such conflicts.

Conflicting interpretations are slightly harder to resolve. Sometimes these will be based on incorrect information, which can be investigated. More often, they will arise from alternative ways of looking at things. Both interpretations might be useful, and proposals can be made which attempt to combine them, or which suggest circumstances under which one or other might be used. Proposals might also recommend that one interpretation should be discarded, in which case the issues raised by the discarded description need to be satisfied in other ways.

Conflicting designs involve a higher level of uncertainty. Often the conflict will be the result of conflicts not tackled at earlier stages, and the issues arising out of conflicting designs will indicate the concerns that lead to them. As the exploration stage has broken down the original conflict into its specific components, the designs can be examined more closely. Possible resolutions include combining the requirements underlying the designs, adapting one design to incorporate issues raised by another, or creating a totally new design which addresses the issues in new ways.

The result of this phase is a set of options for resolution. These will vary from the very specific (such as a particular change to a description), to entire viewpoints. Where an option is only applicable under certain conditions, these are also described as part of the option. Note that the original viewpoints could be considered as possible resolutions: one or other could be accepted unaltered, if it turns out to be a satisfactory resolution. In addition, some proposals will be candidates for combination to produce a more complete resolution, while others will be combinations which might need to be dismantled, if only a part is needed. At this stage, the proposals have not been evaluated or compared in any way.

6.4.3 Support for the Generative Phase

As the model does not prescribe any particular generative method, support for this phase is minimal. The system allows users to record resolution options as either free text, or replacement

descriptions, or partial descriptions, annotated with justifications. Each recorded option has a space to record any conditions which might apply. Also, the user is prompted for a brief title by which the option can be referred.

Two tools are provided to assist in the process of generating these options. These represent the two main sources of resolution: consideration of the category and severity of each of the conflict components, and consideration of how each of the conflict issues might be satisfied. These tools are relatively simple, in that they cycle through the conflict components, and the issues, encouraging the user to fill in the appropriate information, prompting for suggestions. Each suggestion is then recorded as a potential resolution. The tools do not enforce any particular ordering on this process, and the user may even move to the next phase without completing either of these tasks – in this case a warning message is displayed.

The support provided by *Synoptic* for the generative phase is relatively limited, and there are a number of ways in which this support could be improved. For example, information about the various ways in which particular types of conflict might be resolved needs to be encoded into *Synoptic*. This would allow the system to suggest possible resolutions and to ask leading questions to guide the user through the generation process.

6.5 Evaluation Phase

The final phase, evaluation, consists of taking the options for resolutions and relating them both to the map of the conflict generated in the exploration phase, and to each other. The aim is to find the option or combination of options that best resolves the issues involved in the conflict. The approach is similar to that of the exploratory phase, and consists of linking items together and eliciting extra information to supplement the links.

The evaluation phase begins once a sufficient number of options has been generated. In fact, there is no distinct end to the generative phase. When participants feel that a good range of options has been generated, the evaluation phase can be initiated. The generative phase and the evaluative phase are kept deliberately separate, to prevent premature evaluation of the options from stifling generation of new suggestions. However, it is possible that the evaluation phase will also lead to new options, causing a cycling of these two phases.

6.5.1 Relating Options to Issues

The first task is to relate the suggested resolutions to the issues underlying the conflict. This may be done by taking each option in turn, and selecting the issues that it satisfies, or by taking each issue in turn, and deciding which options would satisfy it. Both approaches have merit, in that either may reveal additional links missed in the other. Satisfaction of issues is measured using the criteria attached to them.

The links between options and issues vary in the extent to which the option satisfies the criteria. Also, the relationship may be either positive or negative, where the former indicates the option contributes to the satisfaction of the issue, and the latter indicates it frustrates the issue. Unfortunately, the complex relationship between options and issues cannot be satisfactorily expressed using a simple numeric scale. Instead, a qualitative scale is used, along with explanatory notes. Participants may attach one of five values to each combination of option and issue. The values are: fully satisfies; partially satisfies; no effect; partially frustrates; and totally frustrates. The value “no effect” is the default. If the satisfaction or frustration is partial, an explanatory note is attached. These values will later be used to compare the options which contribute towards each issue.

6.5.2 Relating Options to One Another

The individual resolution options may interact in interesting ways. Some might usefully be combined to produce a resolution which satisfies more issues than either individually: for example, the suggestion of adding a “missing” state to the first viewpoint, and the suggestion of renaming the arrow from both this state and the “at binders” state to “restock” might be combined to give a more complete solution. For other options, combination will negate some of the benefits: for example the suggestion of adding a reserve collection to the first viewpoint is not compatible with the suggestion of maintaining two types of state information, whereabouts and loan status. The range of interactions between options is analogous to the possible interactions between the parts of the original viewpoints, as shown in figure 6.7, which were evaluated using a scale of severity.

Where two or more options can be combined, the combination is recorded as a new option. In creating the combination, the way in which the combination satisfies the issues may need to be reconsidered. In most cases the combination will satisfy all the issues that the individual options satisfied. However, this is not always the case, and in particular, it is not clear how options with differing strength links with an issue might be combined. This information need to be elicited from the participants. Additionally, the combination might only be possible under certain circumstances, which need to be recorded as conditions for the new combined option.

6.5.3 Choosing a Resolution

Once the options have been linked to the issues and to each other, the only remaining problem is to select the best option or combination of options as a final resolution. In many cases, an agreed resolution will have emerged during the process, making much of the evaluation phase redundant. However, in cases where there is no obvious resolution, the options need to be compared. If there is an option (or combination) which satisfies all the issues, then this is a likely candidate. If any participants are unhappy with such a resolution, their reasons need to be elicited: these are likely to indicate issues that were missed in the exploration phase.

To a certain extent, if there is still no clear resolution at this stage, this can be seen as a failure of the negotiation process. The aim of the entire process is to explore the conflict and educate participants about each other’s viewpoint: if this is successful, a resolution should emerge from the process, or the conflict should disappear. In the last resort, the participants might either agree some decision making procedure, or agree to leave the conflict unresolved. In the case of the former, the procedure will depend on the perceived importance of the conflict. An unimportant conflict might be resolved by an arbitrary method, while a more important conflict may require a process of ranking the issues, to select the option that best satisfies the most important issues.

The chosen resolution is represented as a new viewpoint which can be used instead of the original conflicting descriptions. Where the conflict involved only a part of the original viewpoints, the viewpoints can now inherit the relevant section from the resolution viewpoint. The original descriptions are retained as part of the record of the resolution process. The conflict map is recorded as a rationale for the resolution viewpoint, so that it is available for later re-examination if necessary.

6.5.4 Support for the Evaluation Phase

Linking issues to options is a straightforward interactive activity. Two approaches can be used: an option is displayed and the user asked to select those issues which it addresses, or an issue is displayed and the user asked to select the options which address it. The user can switch between these two approaches. In either case the procedure is the same: the option (or issue) is displayed alongside a list of the titles of the issues (or options) to which it may be linked. The full details for any title can be displayed by clicking on it. The user selects the relevant titles, and for each is prompted for the strength of the link, which is then indicated by flagging the title with one of the

symbols: ++, +, -, --, representing totally satisfies, partially satisfies, partially frustrates, and totally frustrates respectively. The links can have explanatory notes attached to them.

For the process of linking options to one another, the user may select a group of options to link together from a list, or may ask *Synoptic* to suggest a possible combination. In the latter case, combinations are chosen to maximise the number of issues satisfied, and may not always be sensible. The chosen combination is recorded as a new option, for which the process of linking to issues is repeated, as described above, with the links already filled in where possible. Where the link cannot be calculated automatically, for instance because one of the combination frustrates an issue which another satisfies, a question mark is displayed to remind the user to fill in the information.

Support for the final stage, selecting a resolution, is limited for the reasons set out in the previous section. An option is available to attach a numerical importance to each issue, so that the system can calculate a numerical score for each resolution option. The mechanics of this are very simple: the user is presented with each issue in turn and asked to select an importance value. These values are then combined with the values on the links between options and issues to generate a score for each option. The system does not allow for disagreement between participants over the importance measures. No attempt is made to support any other type of decision procedure.

6.6 Summary

This chapter has described a model of conflict resolution which can be used to integrate conflicting domain descriptions. This forms a part of the multiple perspectives model introduced in chapter 4. In recognition of the fact that carefully managed conflict can help eliminate errors and improve the quality of the requirements specification, the multiple perspectives model encourages the expression of conflict by allowing participants to describe their viewpoints separately. Expression of conflict needs to be balanced with productive resolution methods, to encourage collaboration and to ensure that conflicts do not become counter-productive. The model described in this chapter was designed with this aim in mind.

The model consists of three phases: exploration of the participants' perspectives; the generation of suggestions for resolving the conflict, and the evaluation of these suggestions. During the exploration phase, the initial conflict is broken down into its components, represented as specific correspondences and differences between items in the viewpoint descriptions. These are annotated with comments describing any assumptions they make and issues they raise. These links and annotations act as a map of the conflict to guide the later stages. Resolution takes the form of designing novel ways of satisfying the issues. In the final phase, the ideas generated are then compared with one another and measured against the issues to determine the level of satisfaction. The option or combination of options which best satisfies the issues is chosen as a resolution.

The model combines the two most co-operative methods of conflict resolution: education and negotiation. Emphasis is placed on the exploratory phase in which participants learn about other viewpoints by comparing them to their own. Participants are encouraged to compare their viewpoint descriptions with others, and this comparison facilitates the elicitation of additional information, such as hidden assumptions. In fact, the final resolution is not necessarily the most important product of the negotiation process – the extra information elicited during the process, and the participants new understanding of one another's viewpoints may be far more valuable.

The entire process is highly interactive, and acts to structure the elicitation of additional information concerning the conflict. Rather than imposing a strict methodology on the participants, the various activities can be freely interleaved, so as to support discussion and exploration. Where an agreement is not reached, the arguments and understanding that have been built up aid judicial arbitration.

The model does have limitations. One of the weaknesses of the model is in the generative phase, where no guidance is offered regarding techniques for generating resolutions. More research is needed to determine which design techniques are appropriate for which types of conflict, so that guidance can be provided. Another weakness is that there is no way of ensuring that all relevant viewpoints participate in the conflict resolution. When a difference between particular viewpoints becomes important enough to attempt to resolve, there may be other viewpoints which contain extra information relevant for the resolution. It is not clear how these relevant viewpoints can be detected, especially in the presence of the mismatches in terminology and style discussed in section 6.1.1. If other viewpoints conflict with the generated resolution, then the resolution process may have to be repeated.

It is tempting to assume the model provides a general framework for conflict resolution. However, the model was designed to fill a specific need, that of comparing and merging previously elicited viewpoint descriptions. The analyst would carry out most of the work, usually over a period of time, involving the viewpoint originators when necessary. The model is not intended provide a form of meeting support, such as that provided in CoLab [Stefik et. al. 1987], nor is it intended to be a problem exploration tool, such as gIBIS [Conklin 1989], although it shares some features of these systems. It is possible, however, that the model could be adapted for use in these situations.

The support system, *Synoptic*, which was built to demonstrate the working of the model is a prototype. It demonstrates the basic processes of the model, but the support it provides is often minimal. Its main deficiency is that it requires the user to do much of the work. By incorporating more knowledge about the conflict resolution process, the system would provide much more guidance. For example, by comparing the issues which resolution options satisfy, the system might generate the most likely combinations of options, and use them to prompt the user for more information. Such a technique was used successfully in the knowledge acquisition system KSS0 [Shaw & Gaines 1987]. Another, major shortcoming of *Synoptic* is that it only allows two viewpoints to be compared at once.

Despite these shortcomings, *Synoptic* provides a solution to the problems raised in the previous chapter, of comparing viewpoints and resolving conflicts between them. We have demonstrated how example conflicts might be resolved using the system. Additional advantages include the ability to mix representations, and the elicitation of additional information. The former is important as it is notoriously difficult to compare knowledge represented in different ways. *Synoptic* provides a means to explore correspondences between different representations. In using the comparison of existing descriptions as a basis for exploration, the system is able to elicit underlying assumptions which might otherwise have remained hidden. Furthermore, the comparison process draws out the issues that the descriptions address.

7 Conclusions and Further Work

This thesis has presented a model for requirements elicitation from multiple perspectives. The model was inspired, in part, by the observation that negotiation forms a vital part of the requirements engineering process, but has not been addressed in existing software engineering methodologies. Using the model, the analyst develops a set of viewpoint descriptions to represent perspectives. These descriptions may be compared when necessary, and conflicts between them resolved. The conflict resolution process emphasises the need for exploration, as issues underlying the apparent disparities need to be elicited.

This chapter presents conclusions from the thesis. Section 7.1 provides a summary: it reviews the problems of requirements engineering that motivated the thesis, restates the objectives, and summarizes the model, showing how it meets the objectives. Section 7.2 is a critical review of the model, while section 7.3 describes areas of further research.

7.1 Summary

This section summarizes the central argument of the thesis. We identified a number of difficulties in requirements engineering, which are accentuated by the importance of the specification (§7.1.1). In addressing these difficulties, we argued that a model of requirements engineering is needed and set out a number of objectives for such a model (§7.1.2). Finally, we proceeded to develop a model which meets these objectives, based on the notion of capturing and representing multiple perspectives (§7.1.3).

7.1.1 Problem Domain

In order to produce software which more closely matches the needs of an organisation, greater attention must be given to requirements engineering. The requirements specification plays an important role in the software development process, as a communication channel between the clients and the software development team. The specification must describe the requirements precisely and unambiguously. Furthermore, it should be representative of the many different people whose needs it describes. In order to validate it, these people should be able to identify with the specification, and preferably will have been involved in its development.

Requirements engineering is difficult because it involves elicitation of knowledge from many different sources and requires use of many different areas of knowledge. A great deal of negotiation is needed, for example to decide the scope of the requirements and to resolve competing needs. If this negotiation is neglected, the participation of the clients is hard to maintain, resulting in a specification that is hard for the clients to identify with and validate.

There is a danger that the specification will represent only a single perspective: the analyst's. Information gathered during requirements elicitation is used to build upon the analyst's preconception of what is required. It is clear, however, that there are many concerns which need to be taken into consideration, often conflicting with one another. It is also clear that conventional software engineering methods not address the identification and integration of these concerns, but rather are geared towards developing and maintaining a single consistent description.

7.1.2 Objectives

Although requirements engineering is unlikely to be completely automated, better support can be provided. The analyst needs a model to guide her expertise, but which is not overly restrictive. This model must facilitate and encourage the collaborative aspect of requirements engineering. Such a model would then lead to the development of tools which assist with gathering and organising requirements knowledge. This would ease the burden of knowledge management.

Chapter 2 set out a number of objectives for a model: the model should encourage the participation of the people whose requirements are being described; it should facilitate an exploratory approach, in which discussion plays an important role; it should allow the analyst to take into account many concerns; and it should support negotiation and conflict resolution. These objectives help ensure that the specification is representative and accessible, reflecting its role as a communication channel between the clients and the development team.

Additionally, tool support for the model should enable the analyst to handle inconsistent and incomplete information. As validation is of prime importance, tool support should provide assistance here too; for example by assisting with the presentation of elicited information back to the originators in a form they are familiar with. The tools should also provide traceability: this implies that all decisions need to be recorded, together with their rationale.

7.1.3 Solution

The model presented in this thesis meets these objectives. It is based on the capture of *perspectives* as self-consistent descriptions of areas of knowledge. Perspectives do not necessarily correspond to people, as one person may use several perspectives, and a perspective might be shared by several people. A perspective can be thought of as a view of the world from the context of a particular role. Perspectives are represented using *viewpoints*, which are formatted descriptions in some appropriate representation scheme.

The model concentrates on two key areas of requirements engineering: elicitation of requirements knowledge and integration of (possibly conflicting) perspectives. Elicitation is based on the capture of perspectives, and involves two main areas of difficulty: identifying perspectives and building the viewpoint descriptions. Integration is based on a model of computer-supported negotiation, and again introduces two key problems: comparing viewpoints and resolving differences.

The part of the model concerned with capturing perspectives was described in chapter 5, along with a system called *Analyser*, which was developed to support this part of the model. The problem of identifying perspectives is tackled by splitting viewpoints when distinctions between perspectives are discovered. The split viewpoints form an inheritance hierarchy, so that shared knowledge is inherited from a common ancestor. Descendant viewpoints represent specific areas over which perspectives disagree. The viewpoints themselves are restricted to represent only that which their originators have explicitly stated. This ensures that they remain accurate models of elicited knowledge. Any additional commentary added by the analyst is represented in the analyst's viewpoint.

The negotiation model was described in chapter 6. As conflicting viewpoints may co-exist in the model, there is no compulsion to resolve conflicts until a resolution becomes necessary or desirable. The resolution process itself involves three phases, of which the initial, exploratory phase is the most important. In this phase, participants compare the conflicting viewpoints, and identify points of correspondence and disparity between them. Issues underlying these correspondences and conflicts are elicited, in order to come to a better understanding of the conflict. During this process, some of the conflicts may disappear. In the second phase, a number of resolution options are generated, according to the types of conflict involved. The final phase

involves comparing these options to one another and to the underlying issues, in order to choose the combination which best satisfies the participants.

7.2 Critical Review

The previous section summarized the objectives of the thesis, and the model developed to meet those objectives. We now examine the strengths and weaknesses of the model, and its support tools. Section 7.2.1 sets the scene by describing the scope of the model and its derivation from diverse fields of research. Section 7.2.2 sets out the principle advantages of the model, while section 7.2.3 describes the remaining problems.

7.2.1 Rationale

The multiple perspectives model provides a framework for requirements engineering. This covers a broad range of activities from the initial elicitation and formulation of requirements, through to the construction and validation of a specification. Some of the activities encompassed in this range are not explicitly addressed by the model; rather the model provides the general structure, within which existing tools and techniques might be applied to individual tasks. On the other hand, the model makes particular activities explicit, such as the exploration of conflict, and in these cases, the model has been used to guide the development of new tools.

The thesis draws together diverse threads from the literature, applying them to the requirements process. Some of the difficulties of requirements engineering are similar to those of knowledge acquisition, and some parts of the model, for example the processing of transcripts and incremental construction of knowledge bases, are based on existing work in knowledge acquisition. The application of models of group interaction, negotiation and decision making to the requirements process is an important feature of this thesis, and, with the exception of Robinson [1990] has not been attempted before. The thesis also draws in ideas from hypertext and distributed artificial intelligence, to create an environment of interlinked knowledge bases.

This appeal to the literature was driven by an identified set of needs, as summarised in section 7.1.2. In particular, existing methods provide little support for the processes of negotiating requirements with the clients, and integrating many competing perspectives.

The model meets this need by using the capture of perspectives as a driving principle. This approach modularises elicited knowledge into recognisable chunks, each of which is associated with an originator. These chunks provide context for each of the statements within them. The ability to represent conflicting perspectives encourages negotiation without stifling exploration. Furthermore, the integration of perspectives is explicitly supported, so that the decisions involved can be recorded and validated.

7.2.2 Advantages

The model achieves the objectives set out at the beginning of the thesis. In addition, the model has a number of advantages. Some of these are a direct consequence of the objectives; for example, the model facilitates reconstruction of the negotiation process by explicitly recording the decision process. This section describes some additional advantages which may not be readily apparent.

The requirements process is normally a team effort. If the process requires a single consistent specification to be maintained, then team working is difficult: careful co-ordination is needed for any changes to the specification. However, by dividing the task up into the elicitation of separate perspectives, the task can be shared. Each viewpoint is constructed independently of the others. The model does not prescribe any particular ordering on the elicitation and discussion process, as it allows consequences of statements to be ignored, and resolution of conflicts to be delayed.

Another advantage is that the analyst does not need to consider the reliability of each new piece of information as it is gathered. In many cases the analyst will know whether elicited information can be treated as undisputed fact, generally agreed principle, or contentious opinion, although there will be cases where this is not clear. All such information provides valuable background, and needs to be recorded. The multiple perspectives model facilitates this by removing the need to consider the reliability of statements added to viewpoint descriptions. In addition, policy and preference can be represented by attributing them to particular perspectives. Information about the reliability and importance of knowledge emerges in the comparison process.

One of the major strengths of the model is in the elicitation arising from comparison of viewpoints. The comparison allows implicit knowledge about the underlying assumptions and goals to be elicited. It gives the participants a chance to explain the rationale underlying their descriptions. It also allows the analyst to gauge the level of agreement between participants over the various elements of the descriptions.

It is important to note that the thesis describes more than just a model of requirements engineering. The model has been used to develop tools to support the requirements process, and these tools have been demonstrated on a number of examples. One major criticism of software engineering methodologies has been the amount of organisational change necessary for them to be adopted by existing institutions. The multiple perspectives model does not suffer this problem. As no restrictions are placed on the form of interaction between analyst and client, nor on the representations used, the model can be integrated with existing practices. Parts of the model can be used and adapted for different circumstances. In particular, many of the tools could be used separately to provide assistance with specific tasks.

7.2.3 Remaining Problems

Requirements engineering encompasses a wide set of activities, and this thesis has not been able to tackle all of these in depth. There remain a number of problems with particular aspects of the model, and a number of activities which the model does not really address. The latter are generally ignored on the basis that they are activities which analysts routinely perform already, and hence existing techniques may be applied. An example is the formulation of requirements knowledge into an appropriate representation scheme: *Analyser* provides the ability to link items in the representation to the original source material where it is available, but gives no guidance for the formulation itself.

One area in which there is a problem with the model is in distinguishing between perspectives. The only criteria used for identifying perspectives is presence of inconsistencies. Whilst conflicts are a major source of inconsistency, it is not clear that inconsistencies can be equated with conflicts, nor is it clear that detecting inconsistencies using a theorem prover is tractable. In fact, the approach used in *Analyser* is slightly more flexible: a set of rules are provided for detecting conflicts in each representation scheme used. However, the development of such rules is not a trivial task, and more work is needed to examine how conflicts can be detected. An approach which can take into account information about the roles, motivations and goals associated with perspectives might be fruitful.

A related problem is that of recognising differences in terminology. At present the support tools rely on the user to spot these, typically during the exploration phase of the conflict resolution process. Assistance might be provided for this problem by comparing the structures of the descriptions for isomorphism.

The use of viewpoint hierarchies solves a number of problems including representing shared information and isolating areas of detail over which there is disagreement. However, the hierarchies may cause problems when handling large quantities of knowledge. In particular, when there are a large number of viewpoints, the inheritance structures may become unwieldy, and need to be reorganised by the participants. Presenting the structure to the users in these cases will be

difficult. A further problem arises when the resolution viewpoints generated by *Synoptic* are added. The original conflicting viewpoints need to be stored as part of the development history of the resolution, and in some cases the resolution will not replace the original viewpoints, but will simply provide another perspective. It is not clear how the new viewpoints fit into the inheritance structure.

Finally there is a problem in ensuring that all relevant viewpoints take part in the resolution process. At present *Synoptic* only supports the comparison of two viewpoints. This limitation could be removed fairly easily, but this would not solve the problem of recognising which viewpoints should participate. When a conflict is detected between two or more viewpoints there may be additional viewpoints which have useful information to add to the resolution process, and might even provide a ready-made resolution. Unfortunately, this may go unnoticed due to differences of terminology.

7.3 Future Work

Several aspects of the model require further work. The above discussion of remaining problems addressed some of these. One further area not covered above is the provision of guidance, which is tackled in a rather ad hoc basis in the existing tools. For example, *Analysier* allows the analyst to associate “to do” notes with any viewpoint, but does not do anything particularly useful with these. *Synoptic* checks that various forms have been completed when the user attempts to move to the next phase, but does not prevent the user from doing so. Ideally, the tools should allow the analyst to plan activities and record and monitor progress, without unnecessary restrictions.

The work in this thesis opens up many exciting new research areas. For example, the model of computer-supported negotiation was developed explicitly for the requirement process, but has other potential applications in software engineering, and in knowledge acquisition. The success of the model in prompting for underlying assumptions and in exploring the issues involved in a conflict suggest it may have applications in the design process.

In order to explore potential applications in other areas, the limits of the model need to be tested. For example, the model is geared towards handling formatted descriptions, and may not cope with highly formalized descriptions, nor with more informal descriptions such as free text. Similarly, the model is suited to one-to-one discussions, typically the analyst and one participant, or two participants with the analyst observing, and may not adapt to use in larger groups. Empirical investigation is needed to establish the scope of the model.

The occurrence of conflict in requirements engineering needs to be studied further. An examination of the types of conflict that occur may lead to the development of a set of heuristics to guide the generation of resolutions. The role that conflict plays also needs to be examined: it is clear that conflicts can reveal important disagreements between participants, but it is not clear how to tell which conflicts are likely to be productive and which are counter-productive. It is not even clear how to measure the productiveness of a conflict. The relation of conflict to the level of communication between participants, the level of abstraction of descriptions, and the degree to which the task is focussed all need to be studied and may yield important results for the handling of conflict.

We have mentioned that the model allows different representations to be used and compared, but have not discussed the wider issues of multiple representations. Translating between representations can be a difficult task, and the multiple perspectives model offers one possible approach. Translating into an intermediate representation [Diederich 1987] may provide further assistance in comparing perspectives. The ability to mix representation schemes also suggests that mediating representations (see §3.2.3) might help in the validation process, as a bridge between the

informal language of the participants and the formal languages used for specification. The applicability of particular representation schemes in the requirements process needs to be explored.

Finally, this thesis has barely skimmed the surface of the problems of validation. One of the concerns of the model was to ease the validation process; this was achieved through the use of viewpoints to represent individuals contributions, together with a higher level of involvement in the requirements process. However, the actual mechanics of validation have not been addressed, and the model needs to be extended to provide guidance for validating both individual viewpoints and resolutions generated from the negotiation process.

7.4 Summary

We have presented and demonstrated a model of requirement engineering based on capturing multiple perspectives. The model was developed in response to the observation that the requirements specification plays an important role in software engineering, and that constructing this document is a difficult task. A set of objectives for a model of requirements engineering were described, which emphasised the need for support for exploration and negotiation. The multiple perspectives model meets these objectives by addressing the elicitation and comparison of viewpoint descriptions. The model was used to develop support tools for these activities.

In developing the model, many diverse threads from other fields were brought together, including models of negotiation and decision-making. The application of these models to the requirements process is novel, and opens up an exciting new area of research.

8 References

- Addis, T. R., 1985, "Designing Knowledge Based Systems", London: Prentice Hall.
- Addis, T. R., 1989, "Knowledge for Design", Proceedings, Fourth AAAI Knowledge Acquisition For Knowledge-Based Systems Workshop, Banff, October 1989.
- Adelson, B., and Soloway, E., 1985, "The Role of Domain Experience in Software Design", IEEE Transactions on Software Engineering, Vol SE-11, No 11, p1351-1360.
- Adelson, B., and Soloway, E., 1986, "A Model of Software Design", International Journal of Intelligent Systems, Vol 1, p195-213.
- Alford, M. W., 1977, "A Requirements Engineering Methodology for Real-Time Processing Requirements", IEEE Transactions on Software Engineering, Vol SE-3, No 1, p60-69.
- Anderson, J. S., and Fickas, S., 1989, "A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem", Proceedings, Fifth IEEE International Workshop on Software Specification and Design, Pittsburg, Penn.
- Appelt, D. E., 1980, "A Planner for Reasoning about Knowledge and Action", Proceedings, First AAAI National Conference on Artificial Intelligence, p131-133.
- Arrow, K. J., 1967, "Public and Private Values", in Hook, S., (ed) "Human Values and Economic Policy", New York University Press, p3-31.
- Axelrod, R., 1984, "The Evolution of Co-operation", Basic Books Inc, NY.
- Balzer, R., 1985, "A 15 Year Perspective on Automatic Programming", IEEE Transactions on Software Engineering, Vol SE-11, No 11.
- Balzer, R., and Goldman, N., 1979, "Principles of Good Software Specification and their Implications for Specification Languages", in Gehani, N., and McGettrick, A. D., (eds) 1986, "Software Specification Techniques", Addison Wesley.
- Balzer, R., Goldman, N., and Wile, D., 1978, "Informality in Program Specifications", IEEE Transactions on Software Engineering, Vol SE-4, No 2, p94-102. Reprinted in Rich, C., and Waters, R. C., 1986, "Readings in Artificial Intelligence and Software Engineering", Morgan Kaufmann.
- Barstow, D., 1984, "A Perspective on Automatic Programming", The AI Magazine, Spring 1984. Reprinted in Rich, C., and Waters, R. C., 1986, "Readings in Artificial Intelligence and Software Engineering", Morgan Kaufmann.
- Barstow, D., 1987, "Artificial Intelligence and Software Engineering", Proceedings, Ninth International Conference on Software Engineering, p200.
- Bell, D. E., Keeney, R. L., and Raiffa, H., (eds), 1977, "Conflicting Objectives in Decisions", J. Wiley & Sons, NY.
- Bjorner, D., 1987, "On the Use of Formal Methods in Software Development", Proceedings, Ninth International Conference on Software Engineering.
- Blamey, S., 1986, "Partial Logic", in Gabbay, D., and Guenther, F., (eds) 1986, "Handbook of Philosophical Logic, Volume III: Alternatives to Classical Logic", D. Reidel Publishing Co.
- Blum, B. I., 1985, "On How We Get Invalid Systems", Proceedings, Third IEEE International Workshop on Software Specification and Design, London, p20-21.

- Boehm, B. W., 1981, "Software Engineering Economics", Prentice-Hall, Englewood Cliffs, NJ.
- Boose, J. H., 1986, "Expertise Transfer for Expert System Design", Elsevier, Amsterdam.
- Boose, J. H., 1989, "A Survey of Knowledge Acquisition Techniques and Tools for Knowledge-Based Systems", *Knowledge Acquisition: an International Journal*, Vol 1, No 1.
- Boose, J. H., 1990, "Knowledge Acquisition Tools, Methods, and Mediating Representations", in Motoda, H., Mizoguki, R., Boose, J. H., and Gaines, B. R., (eds) 1990, "Knowledge Acquisition for Knowledge Based Systems (Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, JKAW-90)", Tokyo: IOS Press.
- Borgida, A., Greenspan, S., and Mylopoulos, J., 1985, "Knowledge Representations as the Basis for Requirements Specifications", *IEEE Computer*, April 1985, p82-90. Reprinted in Rich, C., and Waters, R. C., 1986, "Readings in Artificial Intelligence and Software Engineering", Morgan Kaufmann.
- Brooks, F. P., 1975, "The Mythical Man-Month: Essays on Software Engineering", Reading MA: Addison-Wesley.
- Brown, R., 1988, "Group Processes: Dynamics within and between Groups", Oxford: Basil Blackwell Ltd.
- Burstall, R. M., and Goguen, J. A., 1981, "An Informal Introduction to Specifications Using CLEAR", in Gehani, N., and McGettrick, A. D., (eds) 1986, "Software Specification Techniques", Addison Wesley.
- Burton, M., and Shadbolt, N., 1987, "Knowledge Engineering", Tech Report No 87-2-1, Dept of Psychology, University of Nottingham.
- Carbonell, J. G., 1985, "Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition", Technical Report No. CMU-CS-85-115 Dept of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.
- Carter, R., Martin, J., Mayblin, B., and Munday, M., 1984, "Systems, Management and Change: A Graphic Guide", Harper and Row, London.
- Cleaves, D. A., 1987, "Cognitive Biases and Corrective Techniques: Proposals for Improving Elicitation Procedures for Knowledge-Based Systems", in Gaines, B. R., and Boose, J. H., (eds) 1988, "Knowledge Acquisition for Knowledge Based Systems, Vol 1", Academic Press.
- Compton, P., and Jansen, R., 1989, "A Philosophical Basis for Knowledge Acquisition", Proceedings, Third European Workshop on Knowledge Acquisition for Knowledge Based Systems (EKAW-89), Paris, July 1989.
- Conklin, J., 1986, "A Theory and Tool for Co-ordination of Design Conversations", Tech. Report No. STP-236-86, Micro-electronics and Computer Technology Corporation (MCC), Austin, Texas.
- Conklin, J., 1987, "A Survey of Hypertext", Tech. Report No STP-356-86, Micro-electronics and Computer Technology Corporation (MCC), Austin, Texas. Reprinted in Greif, I., (ed) 1988, "Computer-Supported Co-operative Work: A Book of Readings", Morgan Kaufmann, San Mateo, CA.
- Conklin, J., 1989, "Design Rationale and Maintainability", Proceedings of the Twenty-Second Annual IEEE International Conference on System Sciences, Vol 2, Hawaii.
- Conklin, J., and Richter, C., 1985, "Support for Exploratory Design", Proceedings, AIAA Conference on Computers in Aerospace (also available as Tech. Report No. STP-117-85 Micro-electronics and Computer Technology Corporation (MCC), Austin, Texas).
- Cordingley, E. S., 1989, "Knowledge Elicitation Techniques for Knowledge-Based Systems", in Diaper, D., (ed) 1989, "Knowledge Elicitation: Principles, Techniques and Applications", Ellis Horwood.

- Cunningham, R. J., Finkelstein, A. C. W., Goldsack, S., Maibaum, T. S. E., and Potts, C., 1985, "Formal Requirements Specification - The FOREST Project", Proceedings, Third IEEE International Workshop on Software Specification and Design, London.
- Curtis, B., Krasner, H., and Iscoe, N., 1988, "A Field Study of the Software Design Process for Large Systems", Communications of the ACM, Vol 31, No 11.
- Davis, R., 1979, "Interactive Transfer of Expertise: Acquisition of New Inference Rules", Artificial Intelligence 12, p121-157.
- De Bono, E., 1985, "Conflicts: A Better Way to Resolve Them", Penguin Books.
- DeKleer, J., 1986, "An Assumption-based TMS", Artificial Intelligence, Vol 28, No 2, p127-162.
- DeMillo, R. A., Lipton, R. J., and Perlis, A. J., 1977, "Social Processes and Proofs of Theorems and Programs", Proceedings, 4th International Conference on Principles of Programming Languages. Reprinted in Communications of the ACM, Vol 22, No 5, p271-280.
- Deutsch, M., 1973, "The Resolution of Conflict", Yale University Press, New Haven.
- Diederich, J., 1987, "Knowledge-Based Knowledge Elicitation", Proceedings, Tenth International Joint Conference on Artificial Intelligence (IJCAI-87), p201-204.
- Dietterich, T. G., and Michalski, R. S., 1983, "A comparative Review of Selected Methods for Learning from Examples", in Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., (eds) 1983, "Machine Learning: An Artificial Intelligence Approach", Morgan Kaufmann Publishers Inc., Los Altos, CA.
- Dreyfus, H. L., and Dreyfus, S. E., 1986, "Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer", Macmillan Inc., New York.
- Dubois, E., 1989, "A logic of Action for Supporting Goal-Oriented Elaborations of Requirements", Proceedings, Fifth IEEE International Workshop on Software Specification and Design, Pittsburg, Penn.
- Dubois, E., and Hagelstein, J., 1987, "Reasoning on Formal Requirements: A Lift Control System", Proceedings, Fourth IEEE International Workshop on Software Specification and Design, Monterey, CA., April 3-4, 1987.
- Durfee, E. H., and Lesser, V. R., 1986, "Incremental Planning to Control a Blackboard-based Problem Solver", Proceedings, Fifth AAAI National Conference on AI, p58-64.
- Durfee, E. H., Lesser, V. R., and Corkill, D. D., 1987, "Co-operation through communication in a Distributed Problem-Solving Network", in Huhns, M. N., (ed) 1987, "Distributed Artificial Intelligence", Morgan Kaufmann Publishers Inc, Los Altos CA.
- Dyer, M. G., 1983, "In-Depth Understanding", The MIT Press, Cambridge, Mass.
- Easterbrook, S. M., 1989, "Distributed Knowledge Acquisition as a Model for Requirements Elicitation", Proceedings, Third European Workshop on Knowledge Acquisition for Knowledge Based Systems (EKAW-89), Paris, July 1989.
- Easterbrook, S. M., 1990, "What is Hypertext?", in Gillman, P., (ed), "Text Retrieval: The State of the Art", Taylor Graham, London.
- Easterbrook, S. M., 1991, "Handling Conflict Between Domain Descriptions With Computer-Supported Negotiation", To appear, Knowledge Acquisition: An International Journal.
- Engelbart, D. C., 1963, "A Conceptual Framework for the Augmentation of Man's Intellect", in P. Howerton (ed) "Vistas in Information Handling", Vol 1, Spartan Books. Reprinted in Greif, I., (ed) 1988, "Computer-Supported Co-operative Work: A Book of Readings", Morgan Kaufmann, San Mateo, CA.

- Erman, L. D., and Lesser, V. R., 1975, "A Multi-Level Organization for Problem Solving using Many Diverse, Co-operating Sources of Knowledge", Proceedings, Fourth International Joint Conference on AI, p483-489.
- Feather, M. S., 1987, "The Evolution of Composite System Specifications", Proceedings, Fourth IEEE International Workshop on Software Specification and Design, Monterey, CA., April 3-4, 1987.
- Feather, M. S., 1989a, "Constructing Specifications by Combining Parallel Elaborations", IEEE Transactions on Software Engineering, Vol 15, No 2, Feb 1989, p198-208.
- Feather, M. S., 1989b, "Detecting Interference when Merging Specification Evolutions", Proceedings, Fifth IEEE International Workshop on Software Specification and Design, Pittsburg, Penn.
- Fetzer, J., 1988, "Program Verification: The Very Idea", Communications of the ACM, Vol 31, No 9.
- Fickas, S., 1987a, "Automating the Specification Process", Technical Report No. CIS-TR-87-05, Dept of Computer and Information Science, University of Oregon, Eugene, OR.
- Fickas, S., 1987b, "Automating the Analysis Process: An Example", Proceedings, Fourth IEEE International Workshop on Software Specification and Design, Monterey, CA., April 3-4, 1987.
- Fickas, S., and Nagarajan, P., 1988, "Being Suspicious: Critiquing Problem Specifications", Proceedings, Seventh AAAI National Conference on AI, p19-24.
- Fickas, S., Collins, S., and Olivier, S., 1987, "Problem Acquisition in Software Analysis: A Preliminary Study", Technical Report No CIS-TR-87-04, Dept of Computer and Information Science, University of Oregon, Eugene, OR.
- Finkelstein, A. C. W., 1987, "Reuse of Formatted Specifications", IEE Software Engineering Journal, September 1987, pp186-197.
- Finkelstein, A. C. W., and Fuks H., 1989, "Multi-Party Specification", Proceedings, Fifth IEEE International Workshop on Software Specification and Design, pp185-195.
- Finkelstein, A. C. W., and Potts, C., 1985, "Evaluation of Existing Requirements Extraction Strategies", FOREST Project Report R1, Dept of Computing, Imperial College of Science and Technology, 180 Queens Gate, London SW7 2AZ.
- Finkelstein, A. C. W., and Potts, C., 1987, "Building Formal Specifications Using Structured Common Sense", Proceedings, Fourth IEEE International Workshop on Software Specification and Design, Monterey, CA., April 3-4, 1987.
- Finkelstein, A. C. W., Finkelstein, L. and Maibaum, T. S. E., 1990, "Engineering-In-The-Large: Software Engineering and Instrumentation", Proceedings, UK IT '90, pp 1-8, Peter Peregrinus.
- Finkelstein, A. C. W., Fuks, H., Niskier, C., and Sadler, M., 1987, "A Dialogue Framework for Software Development", Research Report No 87/15, Dept of Computing, Imperial College of Science and Technology, 180 Queensgate, London SW7.
- Finkelstein, A. C. W., Goedicke, M., Kramer, J., and Niskier, C., 1989, "ViewPoint Oriented Software Development: Methods and Viewpoints in Requirements Engineering", Proceedings, Second Meteor Workshop on Methods for Formal Specification, Springer-Verlag, LNCS.
- Finkelstein, L., and Finkelstein, A. C. W., 1983, "Review of Design Methodology", IEE Proceedings, Vol 130, Pt A, No 4, June 1983.
- Fisher, R., and Ury, W., 1981, "Getting to Yes: Negotiating Agreement Without Giving in", Hutchinson.

- Gaines, B. R., 1987, "An Overview of Knowledge Acquisition and Transfer", in Gaines, B. R., and Boose, J. H., (eds) 1988, "Knowledge Acquisition for Knowledge Based Systems, Vol 1", Academic Press.
- Gaines, B. R., 1989, "Design Requirements for Knowledge Support Systems", Proceedings, Fourth AAAI Knowledge Acquisition For Knowledge-Based Systems Workshop, Banff, October 1989.
- Galbraith, J. R., 1977, "Organizational Design", Addison-Wesley, MA.
- Gehani, N. H., 1982, "Specifications: Formal and Informal - A Case Study", Software Practice and Experience, Vol 12, p433-444. Reprinted in Gehani, N., and McGettrick, A. D., (eds) 1986, "Software Specification Techniques", Addison Wesley.
- Giddings, R. V., 1984, "Accommodating Uncertainty in Software Design", Communications of the ACM, Vol 27, No 5, p428-434.
- Ginsberg, M. L., 1987, "Decision Procedures", in Huhns, M. N., (ed) 1987, "Distributed Artificial Intelligence", Morgan Kaufmann Publishers Inc, Los Altos CA.
- Goguen, J. A., 1981, "More Thoughts on Specification and Verification", ACM SIGSOFT, Vol 6, No 3, p38-41. Reprinted in Gehani, N., and McGettrick, A. D., (eds) 1986, "Software Specification Techniques", Addison Wesley.
- Goldman, N., 1982, "Three Dimensions of Design", Proceedings, Second AAAI National Conference on AI.
- Goldstein, I. P., and Bobrow, D. G., 1984, "A Layered Approach to Software Design", in D. Barstow, H. Shrobe, and E. Sandewall, (eds) "Interactive Programming Environments", McGraw-Hill, p387-413.
- Green, C., Luckham, D., Balzer, R., Cheatham, T., and Rich, C., 1983, "Report on a Knowledge-Based Software Assistant", Tech. Report No RADDC-TR-83-195, Rome Air Development Centre. Reprinted in Rich, C., and Waters, R. C., 1986, "Readings in Artificial Intelligence and Software Engineering", Morgan Kaufmann.
- Greenberg, S., 1989, "A Survey of Computer Supported Cooperative Work", Draft Report, Alberta Research Council, Calgary, Canada.
- Gross, N., McEachern, A. W., and Mason, W. S., 1958, "Role Conflict and its Resolution", reprinted in B. J. Biddle and E. J. Thomas (eds) "Role Theory: Concepts and Research", 1966, J. Wiley & Sons.
- Halpern, J. Y., and Moses, Y., 1984, "Knowledge and Common Knowledge in a Distributed Environment", Proceedings, Third ACM Symposium on Principles of Distributed Computing, p50-61.
- Harrison, W., 1987, "RPDE3: A Framework for Integrating Tool Fragments", IEEE Software, Nov 1987.
- Hayes-Roth, B., and Hewett, M., 1985, "Learning Control Heuristics in BB1", Technical Report No. STAN-CS-85-1036, Dept of Computer Science, Stanford University, Stanford, CA.
- Huhns, M. N., (ed) 1987, "Distributed Artificial Intelligence", Morgan Kaufmann Publishers Inc, Los Altos CA.
- Johnson, N. E., 1989, "Mediating Representations in Knowledge Elicitation", in Diaper, D., (ed) 1989, "Knowledge Elicitation: Principles, Techniques and Applications", Ellis Horwood.
- Johnson, P., Johnson, H., and Russell, F., 1988, "Collecting and Generalising Knowledge Descriptions from Task Analysis Data", ICL Technical Journal, Vol 6, No 1, May 1988, p137-155.

- Jones, W. P., 1989, "Bringing Corporate Knowledge into Focus with CAMEO", Proceedings, Fourth AAAI Knowledge Acquisition For Knowledge-Based Systems Workshop, Banff, October 1989.
- Kaplan, S. M., 1989, "COED: Conversation-Oriented Software Environments", (University of Illinois at Urbana-Champaign).
- Keeney, R. L., and Raiffa, H., 1976, "Decisions with Multiple Objectives: Preferences and Value Tradeoffs", J. Wiley & Sons, NY.
- Kerth, N. L., 1987, "The Use of Multiple Specification Methodologies on a Single System", Proceedings, Fourth IEEE International Workshop on Software Specification and Design, Monterey, CA., April 3-4, 1987.
- Kidd, A., and Sharpe, W., 1987, "Goals for Expert Systems Research: An Analysis of Tasks and Domains", Proceedings, Expert Systems 1987, Brighton.
- Kolodner, J. L., 1984, "Towards Understanding of the Role of Experience in the Evolution from Novice to Expert", in Coombs (ed) "Developments in Expert Systems", Academic Press, Computers and People Series.
- Konolige, K., and Nilsson, N. J., 1980, "Multi-Agent Planning Systems", Proceedings, First AAAI National Conference on Artificial Intelligence, p138-142.
- Kramer, J., Ng, K., Potts, C., and Whitehead, K., 1987, "Tool Support for Requirements Analysis", Tech. Report No DoC 87/3, Dept of Computing, Imperial College of Science and Technology, 180 Queensgate, London SW7.
- Kramer, J., Ng, K., Potts, C., Finkelstein, A. C. W., Khan, B., Whitehead, K., Chinnick, S., Brown, G., and Galley, D., 1987, "TARA: Tool Assisted Requirements Analysis", Tech. Report No DoC 87/18, Dept of Computing, Imperial College of Science and Technology, 180 Queens Gate, London SW7 2AZ.
- Lehman, M. M., 1980, "Programs, Life Cycles, and Laws of Software Evolution", Proceeding of the IEEE, Vol 68, No 9, p1060-1076.
- Lehman, M. M., 1987, "Process Models, Process Programs, Programming Support", Proceedings, Ninth International Conference on Software Engineering.
- Lehman, M. M., 1990, "Uncertainty in Computer Application is Certain (Software Engineering as a Control)", Draft Report, Dept of Computing, Imperial College London. Extracts appear in Communications of the ACM, May 1990, p584-586.
- Leishman, D., 1988, "An Annotated Bibliography of Works on Analogy", Research Report No R/88/03/03, Computer Science Dept., University of Calgary, Alberta, Canada.
- Leishman, D., 1989, "A Principled Analogical Tool, Based on Evaluations of Partial Correspondences Over Conceptual Graphs", (MSc Thesis) Research Report No. 89/355/17, Computer Science Dept., University of Calgary, Alberta, Canada.
- Lenat, D. B., 1975, "Beings: Knowledge as Interacting Experts", Proceedings, Fourth International Joint Conference on AI, p126-133.
- Lenat, D. B., Guha, R. V., Pittman, K., Pratt, D., and Shepherd, M., 1990, "Cyc: Towards Programs with Common Sense", Communications of the ACM, Vol 33, No 8, Aug 1990, p30-49.
- Levinson, S. C., 1983, "Pragmatics", Cambridge University Press.
- Littman, D. C., 1987, "Modeling Human Expertise in Knowledge Engineering: Some Preliminary Observations", in Gaines, B. R., and Boose, J. H., (eds) 1988, "Knowledge Acquisition for Knowledge Based Systems, Vol 1", Academic Press.

- London, P. E., and Feather, M. S., 1982, "Implementing Specification Freedoms", in Rich, C., and Waters, R. C., 1986, "Readings in Artificial Intelligence and Software Engineering", Morgan Kaufmann.
- Lowe, D. G., 1985, "Co-operative Structuring of Information: The Representation of Reasoning and Debate", *International Journal of Man-Machine Studies*, Vol 23, p97-111.
- Lubars, M. D., 1987, "Schematic Techniques for High-Level Support of Software Specification and Design", *Proceedings, Fourth IEEE International Workshop on Software Specification and Design*, Monterey, CA., April 3-4, 1987.
- Luce, D. L., and Raiffa, H., 1957, "Games and Decisions: Introduction and Critical Survey", J. Wiley & Sons, NY.
- Maarek, Y. S., and Berry, D. M., 1989, "The use of Lexical Affinities in Requirements Extraction", *Proceedings, Fifth IEEE International Workshop on Software Specification and Design*, Pittsburg, Penn.
- Maibaum, T. S. E., 1986, "A logic for the Formal Requirements Specification of Real-time / Embedded Systems", Alvey FOREST Project Deliverable Report No 3, GEC Research Labs, Marconi Research Centre, Great Baddow, Chelmsford.
- Marcus, S., 1987, "Taking Backtracking With a Grain of SALT", *International Journal of Man-Machine Studies*, Vol 26, No 4, p383-398.
- Meyer, M. A., and Booker, J. M., 1989, "A Practical Program for Handling Bias in Knowledge Acquisition", *Proceedings, Fourth Knowledge Acquisition For Knowledge-Based Systems Workshop*, Banff, October 1989.
- Michalski, R. S., 1986, "Understanding the Nature of Learning: Issues and Research Directions", in Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., (eds) 1986, "Machine Learning: An Artificial Intelligence Approach, Volume II", Morgan Kaufmann Publishers Inc., Los Altos, CA.
- Michalski, R. S., and Baskin, A. B., 1983, "Integrating Knowledge Representations and Learning Capabilities in an Expert System: The ADVISE system", *Proceedings, Eighth International Joint Conference on Artificial Intelligence*, p256-258.
- Michalski, R. S., and Chilausky, R. L., 1980, "Knowledge Acquisition by Encoding Rules vs. Induction from Examples", *International Journal of Man-Machine Studies*, Vol 12, No 1.
- Michie, D., 1982, "Experiments on the Mechanisation of Game Learning 2: Rule Based Learning and the Human Window", *BCS Computer Journal*, Vol 25, No 1, p105-113.
- Miller, M. S., and Drexler, K. E., 1988, "Comparative Ecology: A Computational Perspective", in Huberman, B. A., (ed) 1988, "The Ecology of Computation", North Holland.
- Moore, C. M., 1987, "Group Techniques for Idea Building", Newbury Park, CA: Sage.
- Mostow, J., 1985, "Towards Better Models of the Design Process", *The AI Magazine*, Vol 6 No 1, Spring 1985.
- Mostow, J., 1986, "Why are Design Derivations Hard to Replay?", in Mitchell *et. al.* (eds), "Machine Learning - A Guide to the Current Research", Kluwer Academic Publishers.
- Mostow, J., and Voigt, K., 1987, "Explicit Integration of Goals in Heuristic Algorithm Design", *Proceedings, Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, p1090-1096.
- Neighbors, J. M., 1984, "The Draco Approach to Constructing Software from Re-usable Components", *IEEE Transactions on Software Engineering*, Vol 10, No 5, p564-574. Reprinted in Rich, C., and Waters, R. C., 1986, "Readings in Artificial Intelligence and Software Engineering", Morgan Kaufmann.

- Nii, H. P., 1986a, "Blackboard Systems (part 1): The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures", The AI Magazine, Summer 1986.
- Nii, H. P., 1986b, "Blackboard Systems (part 2): Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective", The AI Magazine, Conference 1986.
- Norman, D. A., 1986, "Cognitive Engineering", in Norman, D. A., and Draper, S. W., 1986, "User Centered System Design: New Perspectives on Human-Computer Interaction", Lawrence Erlbaum Associates.
- Osterweil, L., 1987, "Software Processes are Software too", Proceedings, Ninth International Conference on Software Engineering.
- Partridge, D., 1978, "A Philosophy of 'Wicked' Problem Implementation", Proceedings, AISB/GI Conference on Artificial Intelligence, Hamburg.
- Patchen, M., 1970, "Models of Co-operation and Conflict: A Critical Review", Journal of Conflict Resolution, Vol 14, No 3, Sept 1970.
- Petre, M., and Winder, R., 1988, "Issues Governing the Suitability of Programming Languages for Programming Tasks", in D. M. Jones and R. Winder, (eds) "People and Computers IV (Proceedings of the Fourth Conference of the BCS Human-Computer Interaction Specialist Group)", Cambridge University Press.
- Rapoport, A., (ed) 1974a, "Game Theory as a Theory of Conflict Resolution", D. Reidel Publ. Co., Dordrecht, Holland.
- Rapoport, A., 1974b, "Prisoner's Dilemma - Recollections and Observations", in Rapoport, A., (ed) 1974, "Game Theory as a Theory of Conflict Resolution", D. Reidel Publ. Co., Dordrecht, Holland.
- Regoczei, S., and Hirst, G., 1989, "Sortal Analysis with SORTAL, a Software Assistant for Knowledge Acquisition", Proceedings, Fourth AAAI Knowledge Acquisition For Knowledge-Based Systems Workshop, Banff, October 1989.
- Reubenstein, H. B., 1990, "Automated Acquisition of Evolving Informal Descriptions", Ph.D. Thesis, Tech. Report No AI-TR 1205, MIT Artificial Intelligence Laboratory, Cambridge, MA.
- Reubenstein, H. B., and Waters, R. C., 1989, "The Requirements Apprentice: An Initial Scenario", Proceedings, Fifth IEEE International Workshop on Software Specification and Design, Pittsburg, Penn.
- Rich, C., 1985, "The Layered Architecture of a System for Reasoning About Programs", Proceedings, Ninth International Joint Conference on Artificial Intelligence (IJCAI-85).
- Rich, C., Waters, R. C., and Reubenstein, H. B., 1987, "Towards a Requirements Apprentice", Proceedings, Fourth IEEE International Workshop on Software Specification and Design, Monterey, CA., April 3-4, 1987.
- Robbins, S. P., 1974, "Managing Organizational Conflict: A Nontraditional Approach", Prentice Hall, NJ.
- Robbins, S. P., 1989, "Organizational Behaviour: Concepts, Controversies, and Applications", (fourth edition) Prentice Hall, NJ.
- Robinson, W. N., 1989, "Integrating Multiple Specifications Using Domain Goals", Proceedings, Fifth IEEE International Workshop on Software Specification and Design, Pittsburg, Penn.
- Robinson, W. N., 1990, "Negotiation Behaviour During Multiple Agent Specification: A Need for Automated Conflict Resolution", To appear, ICSE-90.

- Rosenschein, J. S., 1985, "Rational Interaction: Co-operation Among Intelligent Agents", Ph.d. Thesis, Report No STAN-CS-85-1081, Dept of Computer Science, Stanford University, Stanford, CA.
- Rosenschein, J. S., and Genesereth, M. R., 1985, "Deals Among Rational Agents", Proceedings, Ninth International Joint Conference on Artificial Intelligence, p91-99.
- Rychener, M. D., 1980, "Approaches to Knowledge Acquisition: The Instructable Production System Project", Proceedings, First AAAI National Conference on Artificial Intelligence, p228-230.
- Schoenmakers, W. J., 1986, "A problem in Knowledge Acquisition", SIGART Newsletter No 95, p56-57.
- Schuler, D., 1988, "AI and Hypertext in Support of Negotiation", in Bernstein, M., (ed) 1988, "Proceedings, AAAI-88 Workshop on AI and Hypertext: Issues and Directions".
- Scott, B., 1988, "Negotiating: Constructive and Competitive Negotiations", Paradigm Publishing, London.
- Searle, J. R., 1969, "Speech Acts: An essay in the Philosophy of Language", Cambridge University Press.
- Shalin, V. L., Wisniewski, E. J., Levi, K. R., and Scott, P. D., 1990, "A Formal Analysis of Machine Learning Systems for Knowledge Acquisition", in Gaines, B. R., and Boose, J. H., (eds) 1990, "Machine Learning and Uncertain Reasoning (Knowledge Based Systems Vol 3)", Academic Press.
- Shaw, M. L. G., and Gaines, B. R., 1987, "Techniques for Knowledge Acquisition and Transfer", International Journal of Man-Machine Studies, Vol 27, p251-280
- Shaw, M. L. G., and Gaines, B. R., 1988, "A Methodology for Recognising Consensus, Correspondence, Conflict, and Contrast in a Knowledge Acquisition System", Proceedings, Third Knowledge Acquisition For Knowledge-Based Systems Workshop, Banff, November 1988.
- Shaw, M. L. G., and Gaines, B. R., 1989, "Knowledge Acquisition: Some Foundation, Manual Methods and Future Trends", Proceedings, Third European Workshop on Knowledge Acquisition for Knowledge Based Systems (EKAW-89), Paris.
- Shaw, M. L. G., and Woodward, J. B., 1989, "Mental Models in the Knowledge Acquisition Process", Proceedings, Fourth Knowledge Acquisition For Knowledge-Based Systems Workshop, Banff, October 1989.
- Simon, H. A., 1983, "Why Should Machines Learn?", in Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., (eds) 1983, "Machine Learning: An Artificial Intelligence Approach", Morgan Kaufmann Publishers Inc., Los Altos, CA.
- Sloman, A., 1985, "Why We Need Many Knowledge Representation Formalisms", in Bramer, M. A., (ed) 1985, "Research and Development in Expert Systems (Proceedings, 4th Technical Conference of the BCS specialist group on Expert Systems, 1984)", Cambridge University Press.
- Sloman, M. S., and Moffett, J. D., 1988, "Management Domains", Draft Report, Dept of Computing, Imperial College of Science and Technology, 180 Queens Gate, London SW7 2AZ.
- Sommerville, I., 1989, "Software Engineering", Third Edition, Addison Wesley.
- Stamper, R., Althans, K., and Backhouse, J., 1988, "MEASUR: Method for Eliciting, Analysing and Specifying User Requirements", Proceedings, IFIP WG 8.1 Conference on Computerized Assistance During the Information Systems Life Cycle, North Holland.

- Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., and Suchman, L., 1987, "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings", *Communications of the ACM*, Vol 30, No 1. Reprinted in Greif, I., (ed) 1988, "Computer-Supported Co-operative Work: A Book of Readings", Morgan Kaufmann, San Mateo, CA.
- Strauss, A., 1978, "Negotiations: Varieties, Contexts, Processes and Social Order", Jossey-Bass Publishers, San Francisco, CA.
- Swartout, W., and Balzer, R., 1982, "On the Inevitable Intertwining of Specification and Implementation", *Communications of the ACM*, Vol 25, No 7, p438-440. Reprinted in Gehani, N., and McGettrick, A. D., (eds) 1986, "Software Specification Techniques", Addison Wesley.
- Sycara, K. P., 1988, "Resolving Goal Conflicts via Negotiation", *Proceedings, Seventh AAAI National Conference on AI*, p245-250.
- Systems Designers, 1985, "CORE: the Method", CORE manual issue 1.0, Systems Designers Scientific, Pembroke House, Camberley, Surrey, UK.
- Thimbleby, H. W., 1988, "Delaying Commitments", *IEEE Software*, May 1988.
- Thomas, K., 1976, "Conflict and Conflict Management", in Dunnette (ed), "Handbook of Industrial and Organizational Psychology", Rand McNally College Publ. Co.
- Turski, W. M., and Maibaum, T. S. E, 1987, "The Specification of Computer Programs", Addison-Wesley.
- Welbank, M., 1983, "A Review of Knowledge Acquisition Techniques for Expert Systems", Technical Report, Martlesham Consultancy Services, British Telecom Research Labs, Ipswich.
- Wielinga, B. J., and Breuker, J. A., 1984, "Interpretation of Verbal Data for Knowledge Acquisition", Memo 27 of the Esprit Research Project "The Acquisition of Expertise", Dept. of Social Science Informatics, Univ. of Amsterdam, Weesperplein 8, 1018 XA, Amsterdam.
- Wile, D., 1982, "Program Developments: Formal Explanations of Implementations", *Communications of the ACM*, Vol 26 No 11, p902-911.
- Winograd, T., and Flores, F., 1986, "Understanding Computers and Cognition: A New Foundation for Design", Addison-Wesley, NY.
- Woodward, J. B., 1988, "Knowledge Engineering at the Front-End: Defining the Domain", *Proceedings, Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, Nov 1988.
- Yue, K., 1987, "What Does It Mean To Say That a Specification Is Complete?", *Proceedings, Fourth IEEE International Workshop on Software Specification and Design*, Monterey, CA., April 3-4, 1987.
- Zave, P., 1982, "An Operational Approach to Requirements Specification for Embedded Systems", *IEEE Transactions on Software Engineering*, Vol SE-8, No 3, p250-269. Reprinted in Gehani, N., and McGettrick, A. D., (eds) 1986, "Software Specification Techniques", Addison Wesley.
- Zeleny, M., 1982, "Multiple Criteria Decision Making", McGraw-Hill Book Co., NY.

9 Appendix

1 Algorithm for splitting agents

In the Analyser system, each new statement added to a viewpoint is tested for inconsistencies, and if there are any, the viewpoint is split. The following algorithm is used:

- 1) Test for inconsistencies when new statement (**A**) is added to a viewpoint. This is likely to result in a number of inferences (the supposition set) being drawn from the union of **A** with the viewpoint. However, neither **A** nor any of the supposition set are added to the viewpoint yet.
- 2) If there are no inconsistencies then add **A** to the viewpoint, and add the supposition set to the viewpoint's list of suppositions.
- 3) If there is an inconsistency, then:
 - i) Create a sub-viewpoint to contain **A**. This is the motivating statement for this sub-viewpoint.
 - ii) Create another (empty) sub-viewpoint. Any previous sub-viewpoints of the original viewpoint now become descendants of this second sub-viewpoint.
 - iii) If **not(A)** is in the original viewpoint, then move it to the second sub-viewpoint. Otherwise add **not(A)** as a *supposition* in the second sub-viewpoint. This is the motivating statement for this sub-viewpoint.
 - iii) For each sub-viewpoint, if any statements in the original viewpoint are inconsistent with a sub-viewpoint, move them to the other sub-viewpoint.

2 Rules for creating new sub-viewpoints

The algorithm given above does not adequately handle the some situations where the viewpoint to which the new statement is added already has some sub-viewpoints. In this case, the following rules are used:

- 1) If no sub-viewpoint exists, the above algorithm is used (Fig A.1: a & b).
- 2) If the statement is inconsistent with the viewpoint, then it follows that it is inconsistent with all descendants. In this case, the two new viewpoints described in the algorithm above, and any sub-viewpoints of the original viewpoint now become descendants of the second new sub-viewpoint (Fig A.1: c).
- 3) If the statement is consistent with the viewpoint, it is still possible that it is inconsistent with some of the descendants. For each family of sub-viewpoints, test whether the new statement is consistent with each descendant. The following situations are possible:
 - i) The new statement is consistent for all descendants of the original viewpoint – in this case it can be added directly to the original viewpoint (Fig A.1: d).
 - ii) The new statement is inconsistent with all descendants of the original viewpoint – in this case the same solution as used in rule (2) above applies (Fig A.1: c)

iii) The new statement is consistent with some descendants and not with others – if there is only one sub-viewpoint in each pair with which the new statement is inconsistent, it is placed in the alternative to this sub-viewpoint (Fig A.1: e & f). Otherwise, the two new sub-viewpoints are created as in rule 2 above. Any pairs which are both consistent with the new statement become descendants of the first new sub-viewpoint; any that are both inconsistent become descendants of the second (Fig A.1: g).

The possibilities are shown in figure A1. Note that none of these rules generates a family of viewpoints like that shown as the starting point in figures A1(f-h). This situation can only be created by the user re-organising the families of viewpoints. The arrangement is useful where the motivating statements over which the splits occurred are entirely orthogonal. Having several splits from the same viewpoint is a shorthand for creating all possible combinations of those splits. Effectively, each sub-viewpoint inherits not just the contents of the original viewpoint, but any other splits it might have (Fig A.2). Given this, it is important to note that when a new statement is added to one sub-viewpoint, it is incorrect to add it to any other families too, as they automatically inherit the split which contains the new addition (Fig A.1: h).

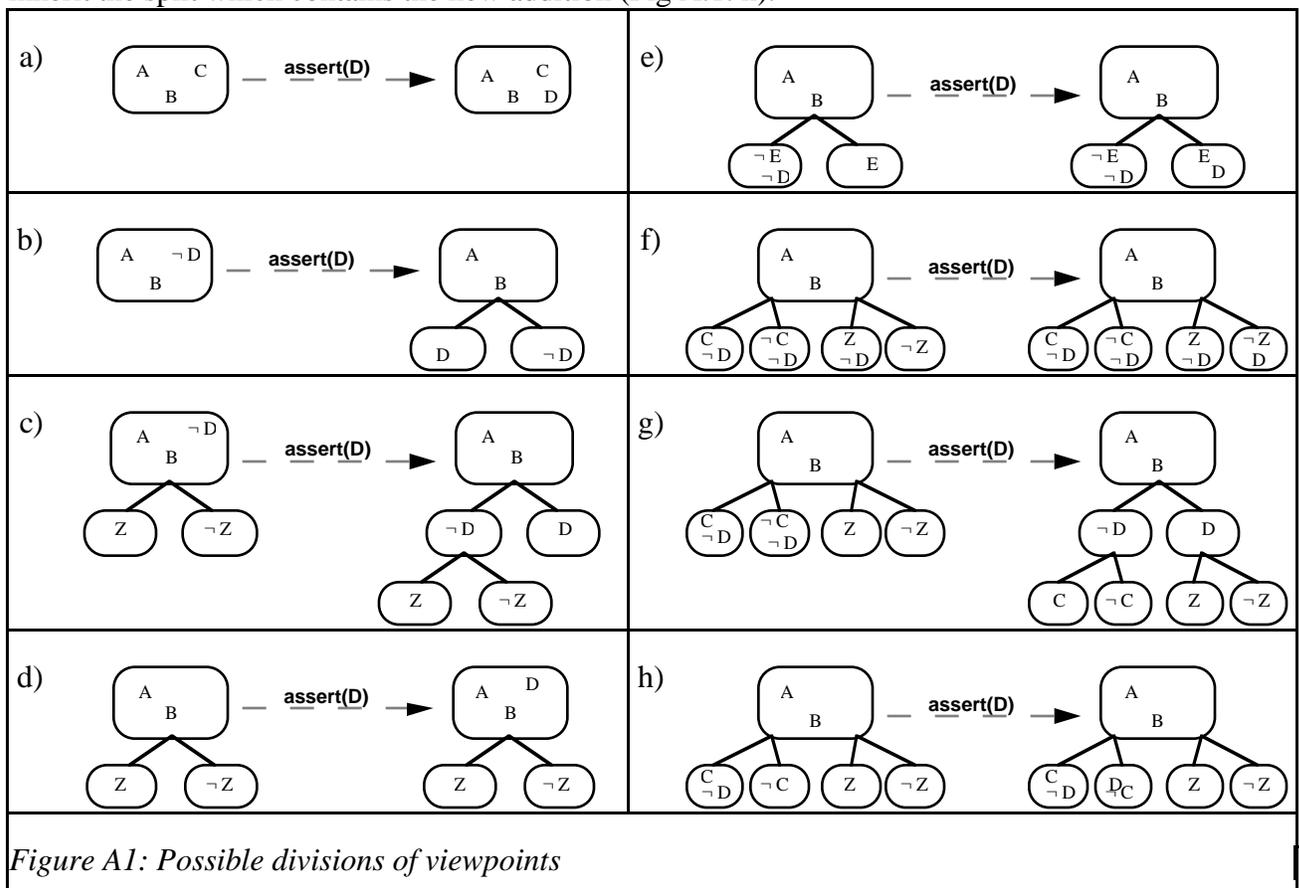


Figure A1: Possible divisions of viewpoints