To appear in Proceedings, First IEEE International Symposium on Requirements Engineering, San Diego, California, 4-6 January 1993

Domain Modelling with Hierarchies of Alternative Viewpoints

Steve Easterbrook

School of Cognitive and Computing Sciences, University of Sussex,
Falmer, Brighton, BN1 9QH, UK. <e-mail: Easterbrook@cogs.susx.ac.uk>

Abstract

Domain modelling can be used within requirements engineering to reveal the conceptual models used by the participants, and relate these to one another. However, existing elicitation techniques used in AI adopt a purely cognitive stance, in that they model a single problem-solving agent, and ignore the social and organisational context. This paper describes a framework for representing alternative, conflicting viewpoints in a single domain model. The framework is based on the development of a hierarchy of viewpoint descriptions, where lower levels of the hierarchy contain the conflicts. The hierarchies can be viewed in a number of ways, and hence allow the participants to develop an understanding of one another's perspective. The framework is supported by a set of tools for developing and manipulating these hierarchies.

1 Introduction

Domain modelling has an important role in requirements engineering. A domain model can form the basis for developing a specification, but more importantly, it provides a focus for the analysts' understanding of the design task. There are a number of techniques for eliciting conceptual models of a domain, many of which were developed in AI. However, it is not clear how these techniques might be adapted for requirements engineering.

There is clearly a relationship between domain models and specifications. For example, it is common to talk of a "knowledge level" description of an AI system, to abstract away from the implementation. However, it is not this relationship in which we are interested. Rather, we regard domain modelling as an exercise in externalising conceptual models of the domain used by participants in the software design process. By externalising these models, participants share them with one another, and develop an understanding of each other's perspectives. This provides a basis for communication between disparate communities, while reducing the chance of misunderstandings.

This paper presents an approach that allows many different viewpoints to be combined into a single domain model without necessarily resolving conflicts between them. Conflicts are treated an important part of the domain, and need to be represented and understood.

1.1 A socio-cognitive stance

For a software engineering team to work together effectively, careful co-ordination is required. This implies that members of the team must have a degree of shared understanding of the task. Achieving and maintaining this shared understanding can be difficult, given the complexity of the task. To a certain extent a shared understanding is established through the production of key documents during the software process, which define the task through a series of publicly examinable specifications.

However, misunderstandings, breakdowns of co-ordination, and conflicts still occur in the software process. Part of the problem is dealing with the sheer amount of information involved, and the changeability of that information. A recent field study of the behavioural aspects of software design [1] identified three major problem areas: the thin spread of application domain knowledge; fluctuating and conflicting requirements; and breakdowns in communication and co-ordination.

Abstractions help the software team to cope, but also introduce new problems. An abstraction represents a particular perspective, suppressing detail that is irrelevant to that perspective. If the assumptions on which an abstraction is based are not articulated, it may be difficult to understand and evaluate the abstraction. Robinson & Bannon [10] suggest that disparities between designers' goals and the way systems are used are often due to 'ontological drift': as abstractions pass through the different sub-groups of an organisation they are interpreted in terms of that particular community's set of meanings, which frequently do not map onto other groups' sets of meanings. Consequently, as the design progresses through the organisation, it is subjected to differing analyses and interpretations.

We argue that problems of ontological drift can be tackled using a process of collaborative domain modelling to develop shared understanding between communities. In particular, we wish to apply the techniques developed in knowledge acquisition for eliciting conceptual models. However, there are some important differences between the

concerns of requirements engineering and knowledge acquisition. Current research in the latter focuses on models of problem solving behaviour, with the goal of implementing systems that demonstrate such behaviour. Emphasis is placed on imitation of an expert's performance. No attempt is made to model the social and organisational setting in which the behaviour is embedded, as is consistent with a purely cognitive stance. On the other hand, much of software engineering deals with systems that support human activities, and hence an understanding of the social and organisation setting is crucial.

We take a 'socio-cognitive' stance, by which we mean we are interested in the interaction of cognitive and social activities, including issues of shared understanding, and the relationship of mental representations with their social and cultural settings. Instead of modelling a single problem solving agent, we are modelling an organisation. Knowledge needs to be elicited from many different sources, and hence we need to deal with many different viewpoints, and the inevitable conflicts between them. In effect, our domain model will encompass a number of different conceptual models, representing different viewpoints and different roles within an organisation.

2 Related Work

The need to deal with multiple views is central to requirements engineering, and a number of approaches have emerged. For example, CORE [13] introduced the notion of *viewpoints*. These represent components of the system and its environment, and can be organisational, human, software, or hardware. Areas of authority for each viewpoint are precisely defined. There is no redundancy, and hence no overlaps between viewpoints. Differences can remain in the expected interaction between viewpoints, which are ironed out in later steps of the method. Effectively, the viewpoints are not used to represent different views of the world, but together represent a single consistent view of the many agents in the world.

Our approach to requirements engineering demands that we support alternative views, to be compared and merged collaboratively. Fickas reviews some of the most promising approaches [5], including Gradual Elaboration [8], in which a small number of types of step are available to build a specification incrementally, Parallel Development [4], in which partial specifications are developed separately according to different development concerns, and merged at a later stage, and Knowledge-Based Critiquing [6] 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 collaboration, with the merge process acting as a focus for conflict resolution, forcing the analyst to document the interaction between different aspects of the specification. However, it is not clear how the merge operations should best be carried out.

One approach to easing the integration of separate specification components is through tools that support negotiation. Robinson [11] uses a single arbitrator to evaluate preferences expressed by various perspectives, and to guide the search for new solutions. A single domain model is used, in which needs are expressed as goals, and perspectives associate different values with these goals. Integration involves searching for novel combinations of proposals, which increase the satisfaction of all perspectives. A joint outcome space is used to identify an ideal, but probably unachievable combination of perspectives, to stimulate consideration of other combinations that come close to this ideal. By contrast, Easterbrook [3] presents a domain-independent technique for resolving conflicts through a process of finding and classifying correspondences between alternative domain descriptions.

Finkelstein has formalised the notion of a viewpoint as having the following components [7]:

– 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 abstracts away from the people involved, allowing one person to have several viewpoints (i.e. several areas of concern), and one viewpoint to represent several people (i.e. where people share an area of concern).

These approaches all provide the ability to model different viewpoints. Further work is needed to clarify how conflicts can be represented. A major issue is the need to establish common ground between viewpoints. The participants must have enough common ground to communicate; such common ground is needed before conflict can be expressed and recognised. In many of the above models, the common ground is assumed: in parallel elaboration the common ground is the initial specification from which the separate developments proceed; in Robinson's approach, it is the shared domain model. The formalised viewpoints model makes no 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.

3 Representing Multiple Viewpoints

In [2], we describe a model of requirements elicitation from multiple viewpoints, where a viewpoint is a self-consistent description of an area of knowledge with an

identifiable originator. Here we concentrate on problems of identifying and distinguishing separate viewpoints, and building viewpoints to represent them. We present *Analyser*, an object-oriented tool for manipulating hierarchies of related viewpoints, where descendant viewpoints represent areas of uncertainty and conflict.

Analyser does not provide a method for elicitation of requirements, but simply provides an environment in which the elicited knowledge can be represented and manipulated. The viewpoints use any 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. The system makes no prescription for the requirements specification, but concentrates on the domain modelling aspects of requirements engineering.

4 Identifying Viewpoints

Before the acquisition process begins, there is little indication of what the relevant viewpoints are, and yet it is desirable to know which viewpoint an item belongs to when it is elicited. Most attempts to provide intelligent support for requirements definition make use of a pre-existing domain knowledge base (e.g. [9]). In general, however, the prior existence of such a knowledge base cannot be assumed: requirements elicitation *is* the initial exploration of a new application. The knowledge needed to distinguish viewpoints is unlikely to be available prior to elicitation of the knowledge embodied in the viewpoints.

Furthermore, distinctions between viewpoints might only become apparent in retrospect. Our viewpoints do not correspond to people. We regard a viewpoint as a description of the world from a particular angle. More precisely, it represents the context in which a role is performed. Not only is it not always clear which viewpoint a person is using at any one time, but viewpoints can be shared by groups of people or organisations. This can make it difficult to identify an appropriate viewpoint for any particular item.

4.1 Evolving Viewpoints

Rather than create an extensive set of viewpoints beforehand, the analyst evolves them as distinctions between them become clear. 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 when new distinctions are discovered.

As a starting point, an initial set of viewpoints corresponding to the people with whom the analyst interacts is used. All assertions that a particular person makes are collected as a viewpoint, to represent that person's conceptual model. As an example, for a library system, the analyst may initially talk to two people: a librarian and a user. Two viewpoints would be created, to keep their contributions apart. As the elicitation continues, other people may need to be interviewed, and new viewpoints are added to represent them.

We noted that viewpoints do not correspond to people. If an inconsistency arises within a viewpoint, then either a mistake has been made, or the originator has been using several incompatible perspectives. In this case the viewpoint is split into several 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 below. In general, 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 *Analyser* system copes with conflicts by creating families of viewpoints to handle them. New viewpoints are descendants of the original viewpoint, so that they inherit the original description. Conflicting items are placed in different descendants, so that individually, each remains self-consistent. Any items that are consistent with both descendants remain in the original viewpoint, to be inherited. The process is illustrated in figure 1.

When choosing which items to move to the descendants to isolate the conflict, there are often several combinations to choose from. For example, in figure 1 it would

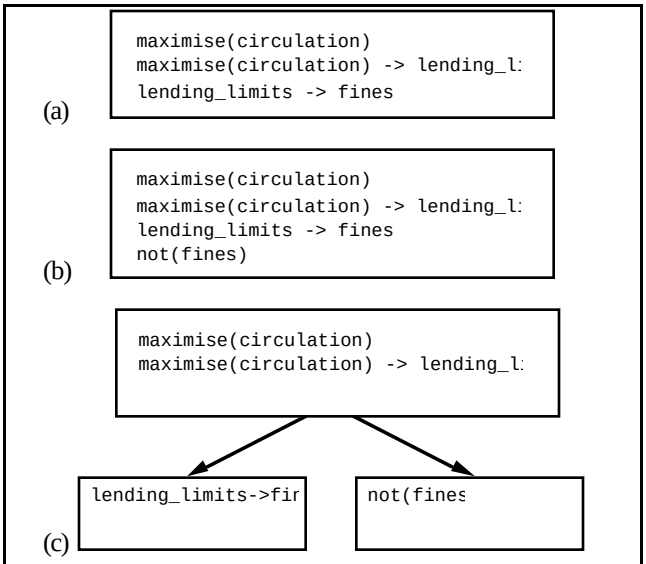

Figure 1: Evolving viewpoints. (a) A viewpoint representing a librarian. (b) The librarian adds the assertion 'not(fines)'. (c) A family of viewpoints is created, to handle the inconsistency.

Viewpoint Splitting Algorithm

1) Test for inconsistencies if the new item (**A**) were added to a viewpoint. However, **A** is not added to the viewpoint yet.
2) If there are no inconsistencies, add **A** to the viewpoint.
3) If there is an inconsistency, then:

i) Create a descendant, **X**, to contain **A**. **A** is the motivating statement for **X**.

ii) Create another descendant, **Y**.

iii) If **not(A)** is in the original viewpoint, then move it to **Y**. Record **not(A)** as the motivating statement for **Y**, whether or not **Y** contains it.

iv) If any items in the original viewpoint are inconsistent with **Y**, move them to **X**.

v) If any items in the original viewpoint are inconsistent with **X**, move them to **Y**.

Figure 2: The algorithm for splitting viewpoints.

be sufficient to separate any pair of statements. As the viewpoint was (by definition) consistent before the newest item was added, the new item can be considered to have caused the inconsistency. Hence the newest item is always chosen for one descendant, and the system attempts to identify the closest conflicting item for the other. In this case it is the last step in the inference chain which generated the inconsistency. The new item that caused the inconsistency, and its negation, are the *motivating statements* of the two descendants respectively. The algorithm for splitting viewpoints is given in figure 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. At some later date, a choice might be made, and the descendants merged into a single consistent description. The remainder of the viewpoint remains consistent, and can still be used for inference. An alternative interpretation is that there are now two separate viewpoints that happen to share some areas of description: each is composed of the union of one descendant 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 viewpoints can be regarded as having a single global ancestor, which holds any consensus information. This global viewpoint can be regarded as shared knowledge.

4.2 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

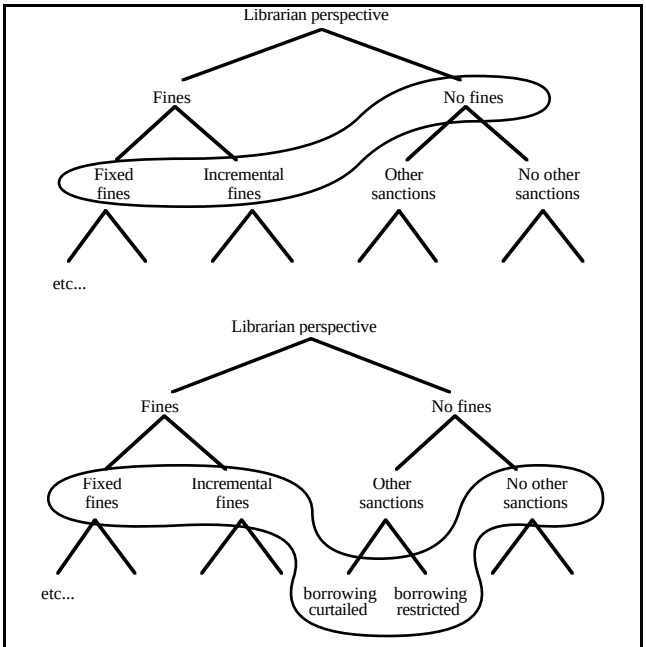

Figure 3: A possible hierarchy of librarian viewpoints, concerning the question of fines. The diagram shows two possible sets of active viewpoints.

conflict requiring a decision, then each choice can be explored in more detail, uncovering further conflicts. For example, consider the librarian viewpoint. 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 3). The discussion of these latter issues does not pre-suppose a decision has been taken about whether to have fines, and the viewpoint that excludes fines is still part of the model.

This example also serves to illustrate that the set of relevant viewpoints 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 domain, we may wish to ignore the conflict over fines, and consider there to be a single librarian viewpoint (i.e. the common ancestor in figure 3). At some point, another part of the model may depend on whether we have fines, and so it would be useful to consider there to be two librarian viewpoints, one that wants fines and one that doesn't. At a greater level of detail still, we might consider there to be several viewpoints, representing the different types of fines. Clearly the set of relevant viewpoints varies according to the level of detail needed (Figure 3).

Analyser supports this process of information hiding

by keeping track of which set of viewpoints are *active*. Active viewpoints are displayed in full. Inactive viewpoints are not displayed, but their presence is indicated by flagging the disputed part of their parent viewpoint. By selecting this flagged part, the user can ask to view the descendants, and add them 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. When descendants become active, their common parent is no longer considered as a single viewpoint, and so is no longer active. Once active, viewpoints can be de-activated in favour of their parent.

The inheritance structure implies that the higher an item 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 ought to 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 details are explored. For example, in figure 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 only occurred when considering what would happen if there were no fines. In fact, 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.

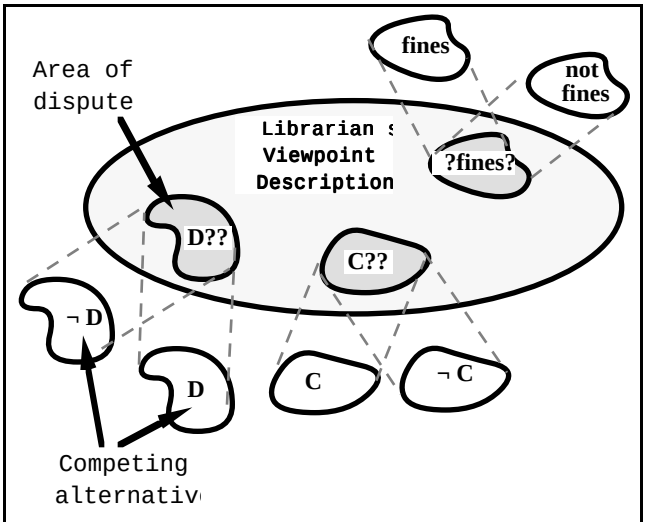

Figure 4: A single viewpoint may contain several areas of uncertainty. The alternatives are the descendants of the viewpoint.

There are many ways to structure the set of viewpoints into hierarchies. The example in figure 1 illustrated that it can be hard to determine which item(s) are in dispute. It is correspondingly hard to determine how viewpoint splits might 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.

A viewpoint description with several different sets of descendants can be regarded as a description containing several areas of uncertainty. For each of those areas, several options might exist (see figure 4). Furthermore, some options might also contain areas of uncertainty. This view of the situation applies when the original agent is active. However, if we make some of its descendants active, the case isn't quite so simple. As each descendant inherits the whole of the parent description, it inherits any other 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 (figure 5).

4.3 Placing Items

Initially, when the viewpoints represent people, there is no difficulty determining to which viewpoint each piece of information belongs. However, as the viewpoint hierarchies develop, the simple relationship between people and viewpoints breaks down. While each viewpoint has a specific originator, an 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.

All the viewpoints corresponding to a particular originator will be in a single hierarchy, having a single ancestor. The viewpoints in the hierarchy all inherit from this ancestor, and so any new item by that originator could be placed in this top-level viewpoint. If the new item is consistent with all previous items from the same originator, the new item 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. During elicitation, most items will be related to their immediate

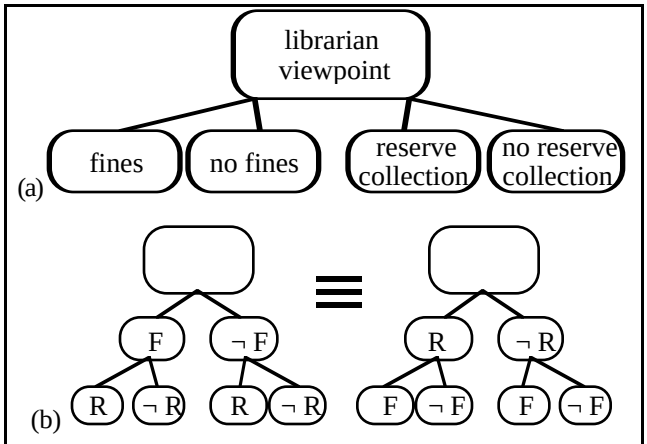

Figure 5: A descendant inherits from its parent other (unrelated) conflicts in that parent. If the parent is active, it appears to have several sets of descendants (a). If one pair of descendants is active, each member will inherit the other pair (b). The two hierarchies in (b) are equivalent, but are displayed differently.

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 a default for any new items.

The new item 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 item may conflict with one of the descendants, as the descendants contain more detail. Rather than check all the descendants immediately, checking is deferred. As the viewpoint to which the item was added is active, none of its descendants can be. Hence, the descendants are flagged as possibly inconsistent, to be checked when (and if) they are made active.

If the new item 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 detail than the ones below, so the system moves up the hierarchy. The highest viewpoint with which the item conflicts is the one that is split. Figure 6 illustrates this process. Figure 7 describes what happens to existing descendants of the viewpoint to which the item is added.

We noted that the item that caused a viewpoint to be split is recorded as the motivating statement for the new descendant. If it is retracted or modified, it may remove the conflict, making it possible to re-unite the descendants.

4.4 Functionality of Viewpoint Creation Tools

Analyser 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 usually identified either by the originator's name, or the name of a role played by the originator. Viewpoints that are created automatically are identified by their motivating statements. These viewpoints can be renamed by the analyst, if they appear to represent identifiable perspectives.

Commands are provided to create and rename viewpoints, list the active viewpoints and to display the contents of a particular viewpoint. Active viewpoints can

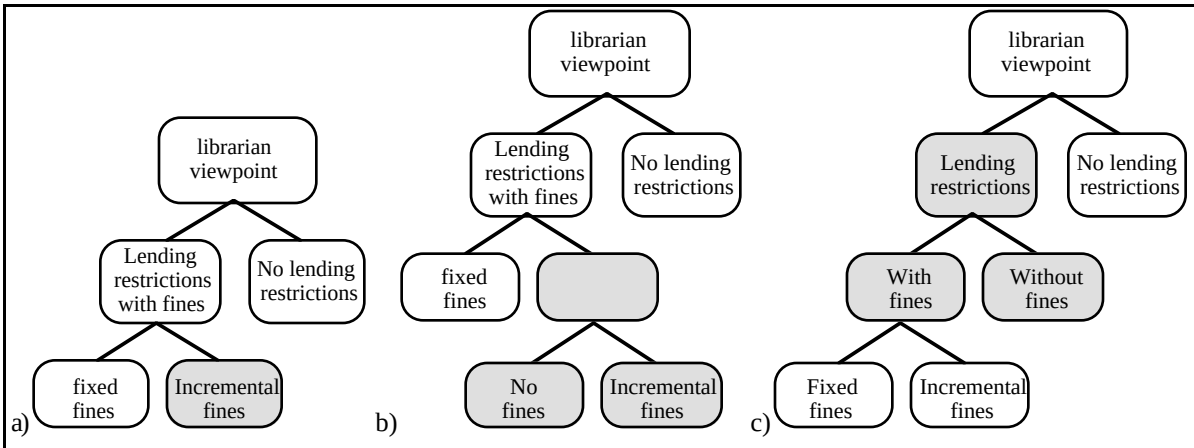

Figure 6: Splitting viewpoints within a hierarchy. The originator was elaborating the shaded viewpoint in (a) when she reflected "of course, we may not need fines". This conflicts with "incremental fines". If the shaded viewpoint was split (b), there would still be a conflict with "lending restrictions with fines" inherited from above. The system works up the hierarchy, to isolate the conflict. In this case the "lending restrictions" viewpoint is split (c).

Inheritance Rules for New Descendants

1) If no previous descendants exist, the usual algorithm (figure 2) is used (a, b).

2) If the item is inconsistent with the viewpoint, then it follows that it is inconsistent with all descendants. In this case, two new descendants, **X** and **Y** are created. **X** will contain the new item, while **Y** represents the status quo. Any previous descendants become descendants of **Y** (c).

3) If the item is consistent with the viewpoint, it might be inconsistent with some existing descendants. For each family, test whether the new item is consistent with each descendant. The following situations are possible:

i) The new item is consistent for all existing descendants. In this case it can be added directly to the original viewpoint (d).

ii) The new item is inconsistent with all existing descendants. In this case rule 2 above applies (c)

iii) The new item is consistent with some descendants and not others. If only one descendant in each pair is inconsistent with the new item, it is placed in the alternative to this descendant (e, f). Otherwise, two new descendants, **X** and **Y** are created, as in rule 2. Pairs that are consistent with the new item become descendants of **X**; any that are both inconsistent become descendants of **Y** (g).

Figure 7: Rules for creating descendants. (Letters refer to the examples in figure 8).

be selected to be de-activated, in favour of an ancestor. In this case any siblings are also removed from the list (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 expanded by activating the descendant viewpoints. In this case the original viewpoint is removed from the list of active viewpoints. Note that when a viewpoint is displayed, all the items inherited from ancestor viewpoints are also shown.

5 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 an inference engine is provided for any representation languages used. 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, which are stored with the inference rules. In this way, the kinds of conflict tested for in each representation can be varied as desired. For example, the rules for detection of conflict might be based on detection of logical inconsistencies (as in the examples above),

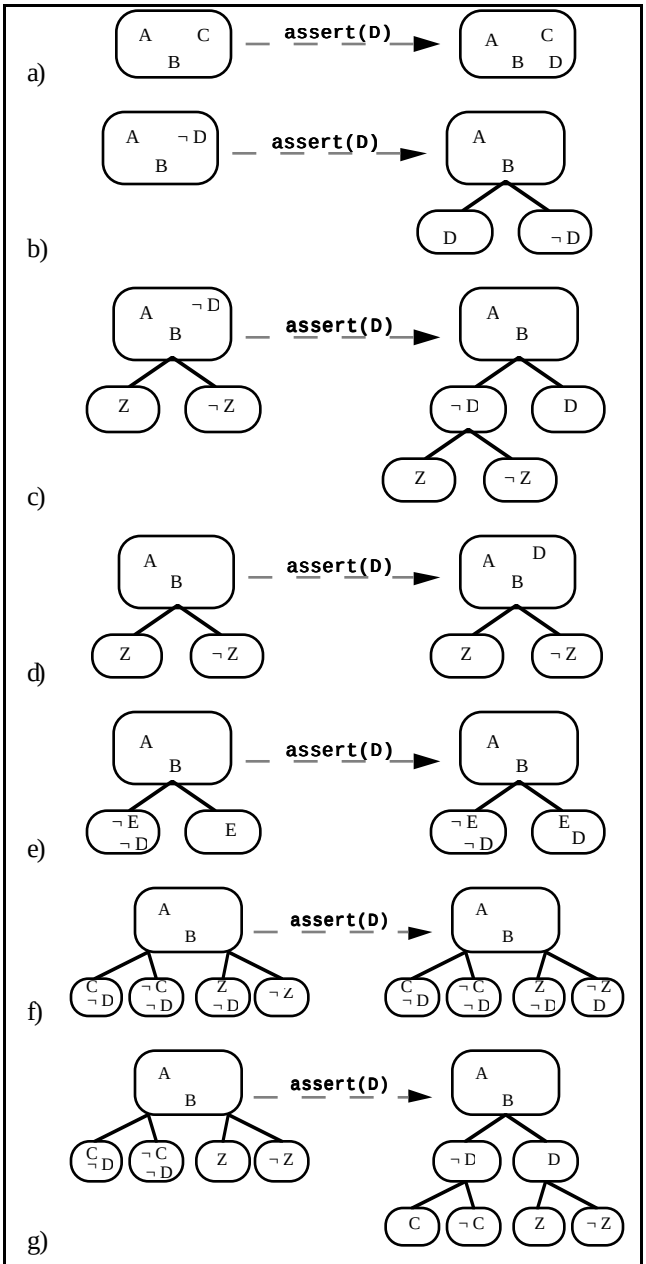

Figure 8: Examples of descendent creation. Note that when a new item is added to one descendant, all other families automatically inherit the split that contains the new addition, as shown in figure 5.

together with tests for clashes of terminology.

New representations can be added to the system by adding a viewpoint containing inference rules, 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* supports first order predicate calculus, dataflow diagrams, and state

transition diagrams. The predicate calculus is supported with a simplified set of rules for detecting conflicts, by generating contradictions through the application of rules such as modus ponens. The graphical representations include validity constraints in their conflict detection rules. For example, in a state transition diagram, a state with two identically labelled transitions from it is treated as a conflict.

6 Discussion

We presented an approach to domain modelling which facilitates the identification and elaboration of viewpoints, and introduces a method of representing conflicting knowledge explicitly, using hierarchies of viewpoints. Each viewpoint contains a description in some suitable representation, and has a unique originator. 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. *Analyser* currently exists as a prototype implementation.

6.1 Remaining Problems

One problem we have not addressed is how to recognise and handle the use of different terminology. This is a difficult problem when combining contributions from many people [12]. There are, however, some mitigating factors in *Analyser*. For instance, we assume that to a certain extent participants will recognise instances of mismatching terminology, either during translation into a structured representation, or when the viewpoints are presented back to them. 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 or semi-formal representations involves the formulation of a suitable ontology. 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 is unlikely to be any pre-existing common ontology. However, the viewpoints must use the same terms, for comparisons, and to allow communication between participants. In fact, there does not need to be a shared ontology, as long as correspondences between terms can be found. The conflict resolution model described in [3] addresses these problems in more detail.

Finally, we have glossed over the relationship between inconsistency and conflict. Conflicts are detected if the rules of a representation scheme are broken, or an inconsistency is generated. However, there are conflicts which might not surface in this way. For example, *Analyser* is unable to detect conflicts between a person's goals unless they generate 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.

References

[1] Curtis, B., H. Krasner, & N. Iscoe (1988) A Field Study of the Software Design Process for Large Systems. *Communications of the ACM*, 31 (11).

[2] Easterbrook, S. M. (1991a) *Elicitation of Requirements from Multiple Perspectives*. PhD Thesis, University of London.

[3] Easterbrook, S. M. (1991b) Resolving Conflicts Between Domain Descriptions with Computer-Supported Negotiation. *Knowledge Acquisition: An International Journal*, Vol 3, pp 255-289.

[4] 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.

[5] Fickas, S., (1987) *Automating the Specification Process*. Technical Report No. CIS-TR-87-05, Dept of Computer and Information Science, University of Oregon, Eugene, OR.

[6] Fickas, S., & P. Nagarajan (1988) Being Suspicious: Critiquing Problem Specifications. *Proceedings, Seventh AAAI National Conference on AI*, p19-24.

[7] Finkelstein, A. C. W., M. Goedicke, J. Kramer, & C. Niskier (1989) ViewPoint Oriented Software Development: Methods and Viewpoints in Requirements Engineering. *Proceedings, Second Meteor Workshop on Methods for Formal Specification*, Springer-Verlag.

[8] Goldman, N. (1982) Three Dimensions of Design. *Proceedings, Second AAAI National Conference on AI*.

[9] 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.

[10] Robinson, M. & L. Bannon (1991) Questioning Representations. In L. Bannon, M. Robinson & K. Schmidt (eds) *Proceedings of the Second European Conference on Computer-Supported Cooperative Work: ECSCW-91*. Dordrecht: Kluwer.

[11] Robinson, W. N. (1990) Negotiation Behaviour During Multiple Agent Specification: A Need for Automated Conflict Resolution. *Proceedings, International Conference on Software Engineering*.

[12] Shaw, M. L. G., & B. R. Gaines (1988) A Methodology for Recognising Consensus, Correspondence, Conflict, and Contrast in a Knowledge Acquisition System. *Proceedings, Third Knowledge Acquisition Workshop*, Banff, November 1988.

[13] Systems Designers (1985) CORE: the Method. CORE manual issue 1.0, Systems Designers Scientific, Pembroke House, Camberley, Surrey, UK