Validating Inconsistent Requirements Models using Graph-based Abduction

TIM MENZIES
University of British Columbia, Canada
STEVE EASTERBROOK
University of Toronto, Canada
BASHAR NUSEIBEH
Imperial College of Science Technology and Medicine, UK
and
SAM WAUGH
Defence Science and Technology Organisation Air Operations Division, Australia

Multiple viewpoints are often used in requirements engineering to facilitate traceability to stakeholders, to structure the requirements process, and to provide richer modeling by incorporating multiple conflicting descriptions. Toleration of inconsistency is a key advantage. However, inconsistency introduces considerable extra complexity when reasoning about requirements. This complexity may limit our ability to maintain a lazy approach to consistency management when we want to validate requirements models. In this paper, we describe a series of experiments with graph-based abduction for multiple world reasoning over inconsistent requirements models. Our abductive algorithm, HT4, sorts an inconsistent model into a number of consistent worlds that support answers to specific queries about the model. This approach avoids the trivialization that occurs in classical deductive inference in the presence of inconsistency. Experiments with this approach reveal that surprisingly few worlds are generated. This result is robust over a range of different models, different amounts of data available from the domain, and different modeling primitives for representing time. To explore the implications of this result, we developed a second algorithm, HT0, that extracts a single world at random from an inconsistent model. Experimentally, HT0 runs fast even for very large models, and supports most of the queries addressed by HT4. The paper discusses possible reasons for this finding, and the implications it has for software engineering in general. We conclude that it is not difficult to support reasoning in the presence of inconsistency even in large models.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms:
Additional Key Words and Phrases: Abductive inference, inconsistency, multiple world reasoning, viewpoints

This work was partially supported by NASA through cooperative agreement #NCC 2-979, partially by UK EPSRC through funding for project MISE (GR/L 55964), and partially by the Australian Defence Science Technology Organisation. The article is a revised and expanded version of a paper presented at the Fourth IEEE International Symposium on Requirements Engineering.
Authors' addresses: T. Menzies, Department of Electrical and Computer Engineering, University of British Columbia, 2356 Main Mall, Vancouver, B.C. Canada V6T 1Z4, tim@menzies.com; S. Easterbrook, Department of Computer Science, University of Toronto, 6, King's College Rd, Toronto, Ontario, Canada, M5S 3H5, sme@cs.toronto.edu; B. Nuseibeh, Department of Computing, Imperial College, 180 Queens Gate, London, UK, SW7 2BZ, ban@doc.ic.ac.uk; S. Waugh, Defence Science and Technology Organisation Air Operations Division, PO Box 4331, Melbourne, Australia, 3001, sam.waugh@dsto.defence.gov.au;

1. INTRODUCTION

Requirements Engineering (RE) is the process of identifying high-level project goals and refining them into software specifications. This process is complicated by the inconsistent and incomplete nature of the information available early in the software life cycle. Different stakeholders often have very different perspectives on the requirements. They use different vocabulary, they talk about different aspects of the problem, they have different ways of structuring their descriptions, and they may have conflicting goals. For these reasons, information gathered from different stakeholders can be difficult to consolidate. It can even be hard just to distinguish which things the various stakeholders agree about, and which things they disagree about.

In viewpoints-based RE, an emphasis is placed on capturing separately descriptions of the viewpoints of different stakeholders, and on identifying and resolving conflicts between them [Finkelstein et al. 1994; Easterbrook 1991b]. Darke and Shanks [1996] note that "if different perceptions of the same problem domain can exist, then it may not always be possible, or desirable, to develop a single integrated viewpoint [that] attempts to satisfy the needs of all stakeholders". Reasoning about multiple viewpoints will be increasingly important as software development focuses more on the development of product families, on building software by integrating commercial-off-the-shelf packages, on distributed development over the web, and on software targeted at globally distributed users with very different needs and contexts of use.

A key advantage to the use of viewpoints is that inconsistencies between viewpoints can be tolerated [Easterbrook and Nuseibeh 1996]. Toleration of inconsistent viewpoints is beneficial for three different aspects of requirements engineering:

—Stakeholder buy-in and traceability. By capturing separately different stakeholder viewpoints during elicitation, stakeholders can identify their contributions, and requirements can be traced back to a source.

—Structuring the process. By permitting parallel development of separate 'workpieces', with no hard constraint on consistency between them, the analysis and specification process can be distributed amongst a team of developers.

—Structuring the descriptions. Richer requirements models can be obtained by separating out different concerns, employing multiple problem structures, and delaying resolution of conflicts.

Most existing approaches to requirements modeling and validation assume that a consistent model is needed; such methods provide little or no support for managing inconsistencies. Tools that overcome this limitation by providing explicit support for identifying, tracking and resolving inconsistencies are emerging [Easterbrook and Nuseibeh 1996; Grundy et al. 1998; Robinson and Pawlowski 1999]. However, there is disagreement over how long inconsistency can and should be tolerated during software development. For example, van Lamsweerde *et al.* [1998] concentrate on resolving inconsistency at a very early stage by analyzing and resolving divergences between stakeholder goals, while Nuseibeh *et al.* [2000] argue that some inconsistencies are never resolved, even in an operational system.

In order to determine how long inconsistency should be tolerated during development, we need to examine the costs. At first sight, the costs of tolerating inconsistency seem high. For example, assume that our requirements model can be expressed as a set of sentences in some suitable logic. Determining whether an arbitrary set of sentences is consistent is an NP-complete problem, and hence global consistency will be expensive to test in large models. Classical reasoning is of limited use for analyzing inconsistent models because the presence of a single contradiction results in trivialization: anything follows from $A \wedge \neg A$. This makes inconsistent requirements models impossible to validate because we cannot refute them. We could overcome this by sorting an inconsistent model into maximally consistent subsets (which we will refer to as 'worlds'). Unfortunately, this sorting is also an NP-complete problem. Gabow *et al.* [1976] showed that the worst case time complexity for generating a consistent pathway through a space containing incompatible pairs is exponential ($O(2^N)$).

Despite this complexity, practical systems can be built to do reasoning over inconsistent models for reasonable sized problems. In the original ViewPoints framework [Finkelstein et al. 1994] the natural structure of the modeling process is exploited so that each viewpoint becomes a self contained specification tool. Inconsistencies are contained at the boundaries between viewpoints, permitting local reasoning with each viewpoint. Inter-viewpoint consistency is explored using a set of heuristic checks. Recently we have explored more rigorous approaches, such as the use of labeled quasi-classical logic [Hunter and Nuseibeh 1998], abductive inference [Menzies 1996a; Nuseibeh and Russo 1999], and multi-valued logics [Chechik et al. 2000]. For the experiments described in this paper we used a particular form of abductive inference known as graph-based abductive validation.

But if we permit inconsistent models to be constructed during elicitation, we will eventually want to validate them. We can build analysis tools that tolerate inconsistency, but in so doing we incur a significant cost in complexity. At what point should we attempt to combine the multiple viewpoints into a single consistent model? The goal of this paper is to address that question.

We proceed as follows. We first provide an overview of the use of viewpoints and inconsistency management in requirements engineering (Section 2). We then introduce the use of abductive inference, present our graph-based abductive validation tool, HT4, and show how it can be used for handling conflicting viewpoints during requirements modeling (Section 3). We then describe two experiments in which we mutated an initial domain model to obtain a range of conflicting viewpoints, and then used HT4 to validate the multiple viewpoints against an existing data set (Section 4). The first experiment shows that, at least in the domain studied, inconsistent models exhibited far less indeterminacy than we expected. The second experiment indicates that multiple world reasoning may be unnecessary. This finding led to the development of HT0, a simplified version of HT4. Whereas HT4 searches all possible worlds, HT0 is a randomized search of some of the worlds. While HT4 is slow ($O(2^N)$), HT0 ran very fast ($O(N^2)$) even for very large models ($N = 20,000$ clauses). Based on these experiments, we conclude that reasoning with inconsistent viewpoints is far simpler than we had thought (section 5). Hence, we need not rush to combine stakeholder's viewpoint into a possibly premature and artificial unified view, even for large requirements problems.

2. VIEWPOINTS AND INCONSISTENCY IN RE

Viewpoints have been used in Requirements Engineering as a structuring technique, to cope with complexity during requirements elicitation and modeling. Unfortunately, different authors have used the term 'viewpoint' for widely different things. For example viewpoints have been used to characterize entities in a system's environment [Kotonya and Sommerville 1992], to characterize different classes of users [Ross 1985], to distinguish between stakeholder terminologies [Stamper 1994], and to partition the requirements process into loosely coupled workpieces [Nuseibeh et al. 1994]. Darke & Shanks [1996] provide an excellent survey and comparison.

A common theme across most of these usages is that 'viewpoints' provide a technique for partitioning a large quantity of information collected from many different sources. The information is collected in coherent, but overlapping chunks ('viewpoints'). Because viewpoints can overlap, there is the potential for inconsistency [Spanoudakis et al. 1999]. However, the inconsistencies between viewpoints can be dealt with separately from the task of describing and elaborating each viewpoint. It is this toleration of inconsistency that distinguishes viewpoints from other problem structuring techniques.

Most of the work on viewpoints has emphasized the benefit they offer during requirements *elicitation*. Viewpoints can be identified with stakeholders, with classes of users, with individual analysts, and so on, to address the multiple perspectives problem [Finkelstein et al. 1994]. Each viewpoint owner is then free to describe her contribution using whatever notation and problem decomposition she chooses, and to focus on the aspects that matter most to her.

In this paper, we are interested in what happens to those inconsistent viewpoints once they have been elicited. How can we build requirements models that incorporate inconsistent information from multiple viewpoints, and how can we validate these models? Of course, building models in requirements engineering is not an end in itself: models are useful only if they help us validate our understanding of the requirements, and if they help us communicate that understanding to others.

To understand why most RE methods assume inconsistencies should be removed before models can be validated, we can compare requirements validation with theory-building in science. Many requirements engineers adopt a logical positivist approach – essentially the belief that there is an objective world that can be modeled by building a consistent body of knowledge grounded in empirical observation [Nuseibeh and Easterbrook 2000]. In RE, this is the view that the requirements describe some objective problem that exists in the world, and that validation is the task of making sufficient empirical observations to check that the problem has been captured correctly. Popper's observations on the limitations of empirical observation clearly apply: scientific theories can never be proved correct through observation, they can only be refuted [Popper 1963]. For RE, this suggests that validation should proceed by devising experiments to attempt to refute the current statement of requirements [Jackson 1995]. Descriptions that are not refutable are therefore of little use, which explains the need for consistency; an inconsistent theory is not refutable using classical reasoning because everything follows from it.

Just as logical positivism was severely criticized in the latter part of the twentieth century, so requirements methods that adopt this stance have been questioned. For

example, take the Kuhnian idea that observation is not value-free, but rather is theory-driven, and biased by the current paradigm [Kuhn 1962]. For requirements engineers, this implies that the methods and tools they use dominate the way that they see and describe problems. In the extreme case, this can shift the problem of validating requirements statements to a problem of convincing stakeholders that the chosen representation for requirements models is appropriate [Goguen and Linde 1993]. There is a growing recognition in RE that usually the task is not to describe some pre-existing problem in the world; more often the task involves negotiation and consensus building among stakeholders who have conflicting goals [Easterbrook 1991b; van Lamsweerde et al. 1998; Boehm et al. 1998; Damian et al. 2000].

The use of viewpoints to capture overlapping, inconsistent information is a pragmatic attempt to address these criticisms. For example, individual viewpoints can be validated separately by the stakeholders that 'own' them. Viewpoints also allow inconsistencies to be tolerated in an evolving specification. This is important, as the inconsistencies often indicate areas of uncertainty, where more stakeholder input is needed [Nuseibeh et al. 2000]. Unfortunately, this does not solve the problem of validating our entire set of viewpoints: existing approaches still assume that eventually we need to resolve all the inconsistencies as a basis for validation.

In this paper, we propose and test an alternative approach that allows us to validate a merged set of viewpoints without first resolving the inconsistencies.

3. ABDUCTIVE REASONING OVER VIEWPOINTS

To validate requirements models that incorporate inconsistent information from multiple viewpoints, we need to be able to reason about the models. Specifically, we would like to be able to formally challenge the model [Rushby 1995]: we express the requirements model formally, and then challenge it by finding properties that should hold if the model is valid. The properties against which the model is tested are drawn from the domain [Easterbrook et al. 1998], for example from empirical data, from expert opinion, or from stakeholder's goals. In this section we describe how abductive inference can be used for this type of validation.

3.1 Classical Abductive reasoning

Informally, abduction is an inference technique that finds the set of assumptions needed to draw a specific conclusion [O'Rourke 1990]. Imagine that we have a domain theory[1], D, along with some observed facts, I, about the domain. *Deductive Inference* allows us to derive some conclusions, O, from D and I, such that

$$I, D \vdash O \tag{1}$$

I and O can be thought of as inputs and outputs for a deductive reasoning engine with D as its knowledge base. *Inductive Inference* allows us to derive a domain theory, D, given lots of examples of I and O. *Abductive Inference* allows us to identify which I would be needed in order to derive a given O from the domain theory, D. More specifically, we may know *some* of the inputs and wish to know

[1]We will use the term *domain theory* as synonymous with the more common software engineering term *domain model*, as we intend both the logic and scientific notion of a 'theory'.

what additional inputs are needed to derive particular outputs. We refer to the additional inputs as assumptions, A, so the inference task becomes: given I, D, and O, find a minimal A such that:

$$I, D, A \vdash O \tag{2}$$

and, because Equation 2 is trivially true in any classical deductive logic if the antecedents are inconsistent, we need to check:

$$I, D, A \nvdash \bot \tag{3}$$

Like inductive inference, abductive inference is under-constrained. There may be many possible choices for A, so the results need to be assessed using a plausibility operator [Bylander et al. 1991].

Abductive inference is closely related to deductive inference; the main difference is in terms of proof strategy. This can be seen more clearly when we examine soundness and completeness. Abduction uses the standard set of deductive inference rules to construct proofs, hence abductive proofs are sound (with respect to a given semantics) for any logic in which deductive inference is sound. To consider completeness, note that Equation 2 can be re-written as

$$I, D \vdash A \rightarrow O \tag{4}$$

For reasoning over consistent theories (i.e. when Equation 3 holds), abductive inference is complete for any logic in which the deductive inference rules are complete: if we restrict ourselves to the case where $A = \mathit{true}$, then Equation 4 reduces to Equation 1, which is exactly the deductive case[2]. However, for inconsistent theories, we use a paraconsistent form of abductive inference, described in the next section, for which completeness and soundness with respect to the classical consequence relation $\vdash$ do not apply[3].

Abductive inference has been used for a wide range of tasks [Menzies 1996b], including identifying missing assumptions, and generating explanations for given observations. In this paper, we concentrate on its use to validate inconsistent domain models formed by merging multiple viewpoints. The next two sections describe how abductive inference can be used for inconsistent domain theories, and how we use it as a validation tool. We will then present our graph based abductive validation tool, HT4, and show how it can be used to validate inconsistent viewpoints.

3.2 Paraconsistent abductive reasoning

Abductive inference can be used for inconsistent domain theories in the following way. Firstly, note that if D contains contradictions, then Equation 3 is unsatisfiable

[2] and hence deductive inference can be viewed as just a special case of abductive inference!

[3] Soundness and completeness of paraconsistent reasoning can be defined in terms of a non-classical consequence relation, such as credulous consequence, $\mid\sim$, which is defined over maximally consistent subsets of an inconsistent theory [Makinson 1994]. However, such considerations are beyond the scope of this paper.

for any A. So instead of using the entire theory, we look for sets of assumptions a_j that satisfy:

$$i_j, d_j, a_j \vdash o_j \tag{5}$$

for $d_j \subset D$, $i_j \subset I$, $o_j \subset O$, and to avoid trivialization, we require

$$i_j, d_j, a_j \nvdash \bot \tag{6}$$

Operationally, an abductive algorithm searches for proofs of elements of O, using the structure of the domain theory to guide the search. Each time a candidate proof, i_k, d_k, a_k, of elements of O, is found, we check whether the terms used in d_k, a_k are consistent with any world d_j, a_j that we have already found; if so, we add the terms used in the new proof to d_j, a_j, if not we create a new world to contain d_k, a_k. In this way we develop a set of maximally consistent subsets of D, which we call *worlds*. For a plausibility operator to evaluate each a_j, we could simply measure $|o_j|$, and prefer those worlds that cover the more elements of O.

In effect, this form of abductive inference adds paraconsistent reasoning to an existing (non-paraconsistent) logic. Paraconsistent reasoning permits some contradictions to be true, without then entailing that all contradictions be true [Mortensen 1995]. Classical deductive logic is not paraconsistent, because anything follows from a contradiction, including all other possible contradictions. Paraconsistent logics are interesting because they avoid *trivialisation* in the presence of inconsistency[4]; for this reason they appear to have useful applications for reasoning about inconsistent specifications [Hunter and Nuseibeh 1998].

To show that abductive inference is paraconsistent, we need to distinguish between theories and theory presentations. A *theory presentation* is a subset of the sentences of a theory, which, when closed under deduction, forms the theory. In a non-paraconsistent logic, if a given theory presentation contains a contradiction, then the corresponding theory is trivial. However, given such a theory presentation, our abductive inference procedure constructs only those proofs that do not contain contradictions. These are then sorted into worlds, each of which is consistent. Hence, the *abductive* closure of an inconsistent theory presentation is actually a set of theories, each of which contains no contradictions. We have therefore allowed true contradictions in a theory presentation, without then entailing that all contradictions be true in the corresponding theory.

Abductive inference does not by itself form a paraconsistent *logic*, because we do not change the basic deduction rules; rather we achieve the same effect by restricting the form of the proofs that are constructed. Hence we describe this as *paraconsistent reasoning* rather than a paraconsistent logic.

3.3 Validating inconsistent theories

In this paper we use abductive inference for validating domain theories. Abductive inference is ideally suited to this, as it allows us to collect empirical observations,

[4]for an accessible introduction to paraconsistent logics, see the Stanford Encyclopedia of Philosophy at `http://plato.stanford.edu/entries/logic-paraconsistent/`

express them as pairs $\langle I, O \rangle$, and present them to an abductive inference engine to be tested against the theory, D. If there is no set of assumptions, A, that satisfies Equation 2, then our data refutes the theory. If we do find different sets of assumptions, and at least one of them is *plausible*, then we have failed to refute the theory. Note that deductive inference is less suited to this task, because Equation 1 may not hold if the theory is incomplete, and in Requirements Engineering we expect our domain theories to be incomplete.

This approach was first proposed by Feldman and Compton [1989], then generalized and optimized by Menzies [1996c]. Abductive validation has found a large number of previously unseen errors in scientific theories taken from internationally refereed publications [Menzies and Compton 1997]. The errors had not previously been detected and had escaped peer review prior to publication.

The approach works equally well for inconsistent theories. We use the paraconsistent abductive reasoning described above to generate multiple worlds from the inconsistent theory for a given pair $\langle I, O \rangle$. If we do not find any world that contains proofs for all of O, then the pair $\langle I, O \rangle$ is not supported by the theory - we have found a refutation. Inconsistent theories exhibit more indeterminacy than consistent theories [Clancy and Kuipers 1997], although our experiments below indicate there is less indeterminacy that one might expect. The greater the indeterminacy, the less likely it is that we will be able to refute the theory; there is greater chance that at least one world will support the data. However, contrast this with deductive validation, with which we can *never* refute an inconsistent theory.

The idea of refuting an inconsistent theory makes sense if we view each inconsistency as a disagreement between the people that developed the theory (our stakeholders). We may still be able to find data that contradicts the things that the stakeholders do agree on.

Note that we can also use the approach to tell us whether the theory is consistent *with respect to a given data set*. If we do not generate multiple worlds for some given query $\langle I, O \rangle$, the domain theory is consistent for the portion of the domain covered by $\langle I, O \rangle$. Conversely, if we do generate multiple worlds, then the query $\langle I, O \rangle$ represents a *boundary condition*, i.e. a condition for which the inconsistency does matter.

Our definition of a boundary condition is similar to that of van Lamsweerde et al. [1998], who use it when reasoning about inconsistencies between stakeholder goals. Briefly, they define a *divergence* as an inconsistency between goals G_1,...,G_n, in the presence of some boundary condition B (assuming a domain theory, D), such that:[5]:

$$D, B, \wedge_{1 \leq i \leq n} G_i \vdash \bot \tag{7}$$

$$\forall j (D, B, \wedge_{i \neq j} G_i \nvdash \bot) \tag{8}$$

However, this definition focuses only on divergences between the stakeholder goals, G_i, not within the domain theory itself. In practice, we have found that domain theories themselves may be inconsistent, especially when constructed from

[5]There is a third condition concerning the feasibility of B, which we will not concern ourselves with here.

multiple viewpoints. Our approach allows us to determine whether a given data set $\langle I, O \rangle$ forms a boundary condition with respect to the domain theory itself – i.e. it is a condition that exercises a latent inconsistency in the domain theory. We believe this to be much more useful than insisting that we eliminate all inconsistencies from the domain theory before it can be used for reasoning.

3.4 HT4 - An abductive validation tool

Our abductive validation tool, HT4, uses graph-based abduction. We assume that our domain theory can be represented as a directed graph, where each node is a term that can be assigned a value, and each edge is an inference step that allows us to propagate the values. Our sets of inputs, outputs and assumptions are sets of assignments of values to some of the nodes of the graph. In principle, we can represent many different types of modeling scheme used in requirements engineering in this way:

—the nodes might be predicates taking the values true or false, and the edges are entailments;

—the nodes might be states represented as state vectors, and the edges represent next-state relations;

—the nodes might be soft goals taking a range of values representing degree of satisfaction, and the edges represent influence relationships between goals (e.g. see Figure 1).

For the examples in this paper, we will use the latter, as this modeling scheme has been widely used in the early stages of Requirements Engineering (see for example the softgoal framework of Mylopoulos *et al.* [1999]).

HT4 is a *graph-based abductive validation* algorithm [Menzies and Compton 1997; Menzies 1996c] that allows us to perform inference on an inconsistent theory, without trivialization. Graph-based abductive validation builds explanations (worlds) for each query $\langle I, O \rangle$ from a domain theory represented as a graph. The pairs $\langle I, O \rangle$ represent specific behaviors that we would like the theory to cover. In other words, each query asks: does the graph contain paths from the given set of inputs, I, to the given set of outputs, O?

In an inconsistent domain theory, there may be paths through the graph that allow us to draw contradictory conclusions: for a given set of inputs, we can derive conflicting values for other nodes depending on the path(s) taken. HT4 deals with this by finding all possible paths from outputs back to inputs across the directed graph, and treating each maximally consistent subset of these paths as a separate *world*. If two paths contain contradictory value assignment for some of their nodes, they are stored in separate worlds. HT4's plausibility operator simply evaluates each world generated according to its coverage of the set of queries.

3.5 HT4 for reasoning over viewpoints

Firstly, note that we distinguish *viewpoints* from *worlds*. Viewpoints correspond to natural division of requirements information into overlapping chunks, perhaps representing different stakeholders, different notations, or different aspects of the domain. We use the term *worlds* to denote maximally consistent subsets of a theory generated during analysis. That is, viewpoints are used to preserve traceability to

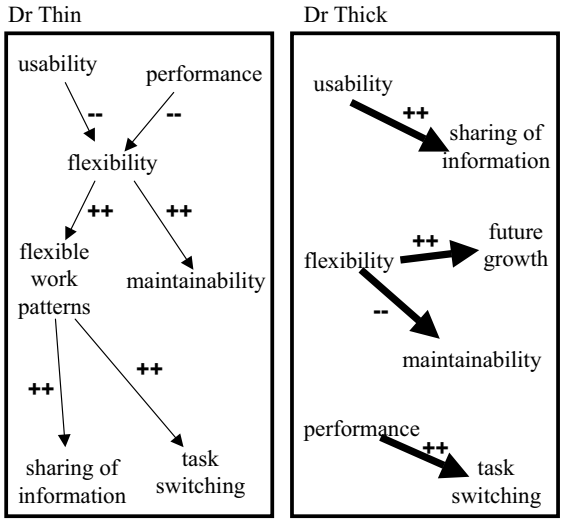

Fig. 1. Soft-goal viewpoints from two experts: adapted from [Mylopoulos et al. 1999]. Dr. Thick's and Dr. Thin's ideas are shown in thick and thin lines respectively.

stakeholders' contributions, while worlds are used purely to facilitate reasoning. If several viewpoints are consistent they can be covered by a single world; conversely if a single viewpoint contains inconsistencies, it may be divided into several worlds.

To illustrate how HT4 can be used to reason with conflicting viewpoints, consider the following example. A requirements engineer has interviewed two domain experts, Dr. Thick and Dr. Thin, to create the two viewpoints shown in Figure 1. The two viewpoints represent the experts' views of the inter-relationship of the goals relating to the development of a new CASE tool.

These viewpoints are softgoal graphs recorded in the QCM notation [Menzies and Compton 1997]. Each node represents a variable that can take three possible values: *up*, *down* or *steady*. There are two types of dependencies between nodes, as follows. An edge labelled ++ indicates a direct connection between goals; for example Dr. Thin would explain *flexible work patterns* being *up* (or *down*) using *flexibility* being *up* (or *down* respectively). An edge labeled −− indicates an inverse connection between goals; for example Dr. Thick would explain *maintainability* being *up* (or *down*) using *flexibility* being *down* (or *up* respectively).

Note that our doctors hold some of the same views, but focus on different aspects of the system. Note also that our doctors disagree on the connection between *flexibility* and *maintainability*. Dr. Thin holds the standard view that future change requests are best managed via a flexible system. Dr. Thick takes the opposite view, saying that when developers work in very flexible environments, their bizarre alterations confuse the maintenance team.

In effect, the viewpoints represent overlapping fragments of a domain theory. We can query these viewpoints to find out how we might effect certain changes. For example, we could ask whether it is possible to increase our ability to move between tasks ("task switching"=up) and increase future growth ("future growth"=up) while reducing the amount of documentation shared across the development team

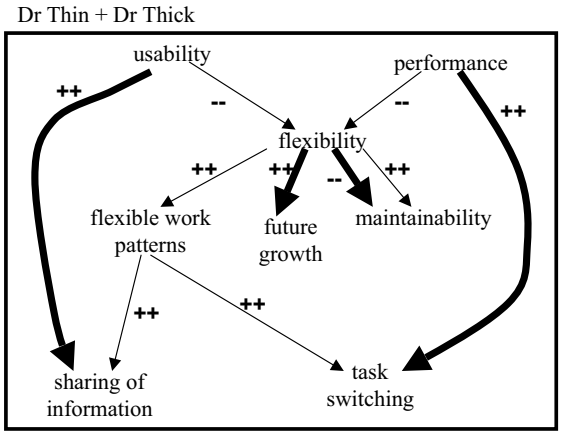

Fig. 2. Union of the viewpoints of Figure 1.

("sharing of information"=down).

We can assess the two viewpoints in Figure 1 in a number of ways. For example, we can measure how useful the viewpoints are in answering queries related to the requirements analysis. More importantly, we can partially validate the viewpoints against historical data, or data observed in similar domains: if one viewpoint explains more of the observed behaviors than another, we can regard it as "more valid". However, selecting one of these viewpoints in preference to the other may not yield the best solution. It is unlikely that, for example, Dr. Thick is totally correct and Dr. Thin is totally wrong. It may be preferable to combine portions of Dr. Thick's and Dr. Thin's viewpoints. Hence, we first merge the two viewpoints to create a single (inconsistent) domain theory, Figure 2. This combined space will be explored, looking for portions that explain our set of observed behaviors.

To demonstrate how graph-based abductive validation allows us to validate an inconsistent model, imagine that we have an observed case where *performance=up* and *usability=down* led to *task switching=up, future growth=up* and *sharing of information=down*. We can present this as a query to the domain theory in Figure 2, with appropriate inputs and outputs. There are five proofs `P` across Figure 2 that can reach the outputs from the inputs:

`P.1`: *performance=up, task switching=up*

`P.2`: *usability=down, flexibility=up, flexible work patterns=up, task switching=up*

`P.3`: *usability=down, flexibility=up, future growth=up*

`P.4`: *usability=down, sharing of information=down*

`P.5`: *performance=up, flexibility=down, flexible work patterns=down, sharing of information=down*

Note that these proofs contain contradictory assumptions; e.g. *flexibility=up* in `P.2` and *flexibility=down* in `P.5`. When we sort these proofs into maximal subsets that contain no contradictory assumptions, we arrive at the two worlds shown in Figure 3. Note that world #1 covers all the outputs in the query while world #2 only covers two-thirds of the outputs.

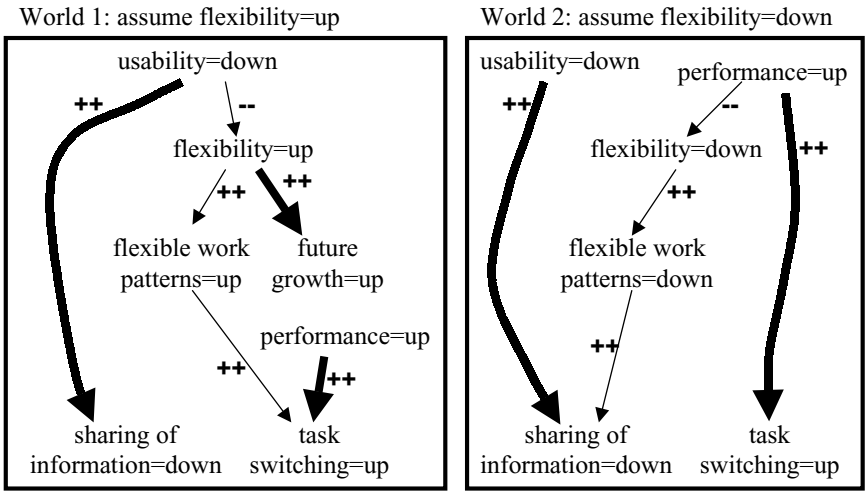

Fig. 3. Worlds from Figure 2. World #1 contains the proofs that do not contradict *flexibility=up*; i.e. `P.1, P.2, P.3, p.4`. World #2 contains the proofs that do not contradict *flexibility=down*; i.e. `P.1, P.4, P.5`.

The use of viewpoints in requirements engineering is geared towards gaining stakeholder buy-in and facilitating discussion as much as it is about selecting the best model. Hence, this abductive approach does not offer automatic support for resolving conflicts between different experts. However, it does support the automatic generation of reports describing the relative merits of the ideas of Dr. Thick and Dr. Thin as follows:

—Our query does not refute the theory, because we found a world (world #1) that contains proofs for all of O.

—Our query is a boundary case that reveals an inconsistency. Hence, with respect to the our observed case, inputs $\langle$*performance=up, usability=down*$\rangle$ and outputs $\langle$*future growth=up, sharing of information=down, task switching=up*$\rangle$, our doctors' views are inconsistent.

—We can measure the utility of different worlds. This can be achieved through a variety of scoring functions for generated worlds. For example, we might give world #1 a higher score than world #2 because world #1 covers all the outputs. More sophisticated scoring functions might give higher scores to worlds that contain multiple reasons for believing each output.

Note how abduction can guide the viewpoint owners to a point of collaboration, despite having conflicting viewpoints. Rather than focus on the obvious dispute (the effects of flexibility on maintenance), abduction can show the requirements engineer how to validate other portions of the viewpoints using domain data during the analysis process.

In summary, abductive reasoning builds worlds from the union of the viewpoints of different stakeholders. This framework will be used below to assess the utility of multiple world reasoning for conflicting viewpoints. In particular, we will assess

whether generating multiple worlds provides greater expressive power than a single world chosen at random from those generated.

3.6 Applying HT4 in requirements analysis

Before we present our experiments in the cost and benefits of multiple world reasoning in RE, we need to consider the broader issues surrounding our use of graph based abductive validation. Although we have described a particular example using abduction to reason about softgoal graphs, we expect that the approach generalizes to a wide variety of modeling notations, with the following limitations:

—We assume a vocabulary shared by all viewpoint owners. For example, when the domain experts in the example above talk about "flexibility", we assume they both mean the same thing. In practice, conflicts over the use of vocabulary are a pervasive and complex problem. Effort can be expended early in the requirements process to establish a common ontology, but this does not guarantee to prevent terminological clashes, because of ontological drift [Easterbrook 1995]. Often, such problems are first revealed as inconsistencies between viewpoints [Easterbrook and Nuseibeh 1996]; the analysis we have described above may help to pinpoint terminological clashes. For methods of repairing terminological clashes see, for example, the repertory grid research of Gaines and Shaw [1989; 1997].

—We assume that there exists a dataset that can be used to validate the viewpoints, which is endorsed by the entire community. The dataset may be a historical case library, or a set of observations taken during the RE process. Our approach is of limited use if the viewpoint owners disagree about the validity of the dataset.

—We assume a scoring function where the *worth* of a world is along the lines of *what percent of the observed behaviors is found in that world?* Further, our scoring system assumes there is a uniform distribution of *utilities* across the queries. That is, it measures worth by the number of behaviors explained, and ignores the fact that some behaviors may be more important than others. This may be an incorrect assumption in some requirements processes. The differences in utilities of stakeholders' goals may be crucial when differences between their viewpoints are considered.

—We assume that the different viewpoints are expressed in the same notation.

Despite these limitations, our abductive approach has a number of advantages. Firstly, unlike previous approaches for analyzing viewpoints, we don't assume that individual viewpoints are internally consistent and we don't need to label information during the analysis according to the viewpoint that it is from. Recalling the above example, abduction can handle inconsistencies within the viewpoint of a single expert. Further, this approach can check whether the disagreement between viewpoints matter for different kinds of analysis: if they don't generate different worlds for given queries when they are combined, then they are not in conflict as far as those queries are concerned.

Secondly, graph-based abductive validation is not the Justification-based Truth Maintenance System (JTMS) [Doyle 1979] approach used in other conflict recognition and management systems (e.g. [Rich and Feldman 1992]). A JTMS searches for a single set of beliefs. Hence, by definition, a JTMS can only represent a single

world at any one time. Our approach is more like the Assumption-based Truth Maintenance Systems (ATMS) [DeKleer 1986]. An ATMS maintains all consistent belief sets. We believe that an ATMS approach is better suited to conflict management in requirements engineering, since the different belief sets (worlds) are available for reflection.

Thirdly, one striking feature of other systems that support multiple-worlds (e.g. CAKE [Rich and Feldman 1992], TELOS [Plexousakis 1993]) is their implementation complexity. Rich and Feldman especially comment on the complexity of their heterogeneous architecture [Rich and Feldman 1992]. We have found that it is easier to build efficient implementations [Menzies 1996a; 1996c] using the above graph-based approach than using purely logical approaches.

Fourthly, these tools do not suffer from the restrictions of other tools. For example, while earlier tools such as Synoptic [Easterbrook 1991a] only permit comparisons of two viewpoints, our approach can compare N viewpoints.

Lastly, the approach is simple enough that we can perform experiments on the utility of multiple world reasoning under different circumstances. The remainder of this paper describes such experiments.

4. EXPERIMENTS WITH ABDUCTION

Our experiments are concerned with the costs of multiple world reasoning for analysing requirements models. Such analysis is important to determine at what point during the development process we need to resolve conflicts between requirements. If the costs of multiple world reasoning are too high, we should attempt to resolve any inconsistencies immediately they arise; requirements conflicts should be handled early so that a single consistent requirements model can be developed and maintained. If on the other hand, the costs are low, we can delay resolution of inconsistencies, allowing greater freedom during requirements modeling to entertain divergent views and to delay design decisions.

The first experiment explores how many worlds are generated during our abductive reasoning. Researchers into qualitative models often comment on the indeterminacy of such models (the generation of too many worlds). Clancy and Kuipers [1997] suggest that qualitative indeterminacy is the major restriction to the widespread adoption of qualitative reasoners. Our first experiment set out to explore whether this is a problem for our graph-based abductive reasoner.

The second experiment explores the expressive power of multiple worlds. One of the reasons for maintaining multiple viewpoints during requirements modeling is the assumption that allowing inconsistencies between stakeholders delivers greater expressive power. In other words, a single consistent model would be less able to cover all the requirements our stakeholders care about. Our second experiment compared the expressive power of multiple worlds with a single world chosen at random.

4.1 Experimental setup

The experiments were set up as follows. We chose as an example domain a set of quantitative equations describing a fisheries system taken from Bossel [1994]. We developed an initial domain model (Figure 4) to represent these equations, then mutated it using a series of mutators to generate multiple (conflicting) viewpoints.

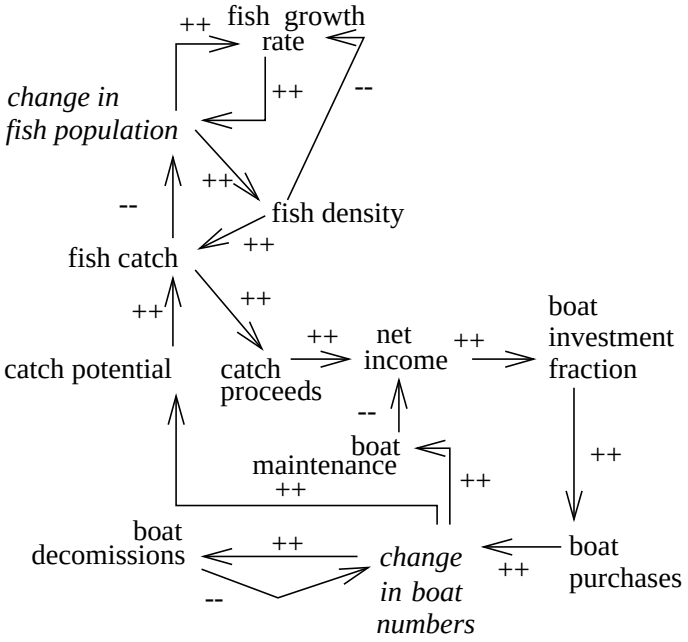

Fig. 4. The fisheries model. Adapted from [Bossel 1994] (pp135-141).

We then combined these viewpoints in various ways to generate a series of inconsistent models to be used as our experimental treatments. Data from the original fisheries equations was used as validation data for the abductive reasoning. We will now explain these treatments in more detail.

Firstly, note that our initial model (Figure 4) is similar to the softgoal graphs of Figure 1, with one significant complication. The variables *change in boatNumbers* and *change in fishPopulation* explicitly model rates of change over time; we therefore need to handle time in the modeling language. The original published data from the quantitative equations offered state assignments at each year. To handle this, we copied the qualitative model once for every time tick (year) in the simulation. That is, variables like *fishCatch* were copied to become *fishCatch@1*, *fishCatch@2*, etc. Variables at time i were connected to variables at time $i+1$ using a particular *temporal linking policy*, which we discuss below.

The mutators we used were as follows:

—Mutator 1 corrupted the edges on the original fisheries model. This mutator selects N links at random in the fisheries model and flipped the annotation ($++$ to $--$ and vice versa). There are 17 edges in the fisheries model. Note that as the number of edges mutated increases from 0 to 17, the mutated model becomes less and less like the original model.

—Mutator 2 added edges to the fisheries model. The original model has 12 nodes and 17 edges (fanout=17/12=1.4). This mutator added 0, 5, 10, 15, 20, 25 or 30 new edges at random (checking each time that the added edges did not exist already in the model). That is, the model fanout was mutated from 1.4 to

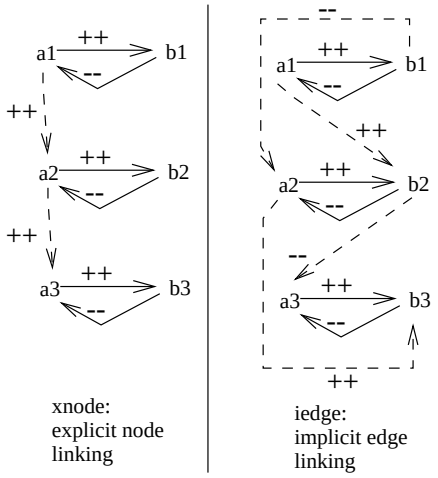

Fig. 5. *Direct(A,B)* and *inverse(B,A)* renamed over 3 time intervals using different temporal linking policies. Dashed lines indicate time traversal edges.

(17+30)/12=3.9.

—Mutator 3 changed the amount of validation data available to our graph-based abduction. The complete set of Bossel equations provides values for all variables at all time points. However, if we provide 100% of this data to the abductive inference procedure, no inconsistency can occur (because every variable is assigned). Mutator 3 threw away some of the data to produce data sets with 0,10,..,90 percent of the variables unmeasured (denoted as U percent unmeasured).

—Mutator 4 changed how the variables were connected across time steps. The XNODE temporal linking policy connects all the explicitly-marked temporal variables from time i to time $i+1$; e.g. *change in boatNumbers=up@1* to *change in boatNumbers=up@2*. Note that there are only two explicit temporal variables in fisheries. It was thought that, since the number of connections were so few, this could artificially restrict world generation. Hence, an alternative temporal linking policy was defined which made many cross-time links. The IEDGE temporal linking policy took all edges from A to B in the fisheries model and connected $A@i$ to $B@i+1$. XNODE and IEDGE can be compared as follows. Consider a model with two variables, A and B, with a direct connection from A to B, and an inverse connection from B to A. Figure 5 shows how the XNODE and IEDGE linking policies expand this model over three time steps.

The above mutators were used to create experimental treatments as follows. For statistical validity, the following procedure was repeated 20 times:

(1) Mutator 4 generated two copies of the original model, using the IEDGE and XNODE linking policies.

(2) Mutator 1 corrupted 0 to 17 of the edges in each model. This produced $20 \times 2 \times 18 = 720$ models.

(3) Mutator 2 added 0, 5, 10, 15, 20, 25 or 30 edges to each model, giving $20 \times 2 \times 7 = 280$ models.

(4) Mutator 3 provided different amounts of domain data for each of the above models, using values of $U = 0, 10, .., 90$. This produced $(720+280)\times 10 = 10,000$ models.

The original Bossel equations were then used to generate 105 pairs of inputs and outputs for our abductive validation process, producing $10,000 \times 105 = 1,050,000$ runs. In our modeling terms, each run represents a validation step: does the (mutated) model capture a particular behavior described in the original equations?

In summary, generated treatments contained (i) a range of different models (ranging from correct to very incorrect); (ii) models with different fanouts, (iii) different amounts of data available from the domain; (iv) different temporal linking policies.

4.2 Experiment #1

For the first experiment, we merely counted the number of worlds generated. If we generate a large number of worlds, it indicates our reasoning process involves too much indeterminacy. Too many worlds means too many possible answers, making the abductive inference virtually useless. Curiously, and contrary to the experience of Clancy, Kuipers, Kakas, *et al.*, graph-based abductive validation exhibits very little indeterminacy.

The results are shown in Figure 6. The upper graph gives the results for the models generated by mutator 1, for different numbers of edges corrupted; the lower graph gives the results for models generated by mutator 2 for different numbers of edges added.

Note the low number of worlds generated. Our reading of the literature (e.g. [Kakas et al. 1998; Clancy and Kuipers 1997]) led us to expect far more worlds than those observed here (maximum=5). Also, note the *hump* shape in all the results graphs. As we decrease the amount of data available, there is less information available to constrain indeterminacy. Hence, initially, less data means more worlds. However, after some point (around 50 percent unmeasured), another effect dominates and the number of worlds decreases.

To explain this effect, we must distinguish between set-covering abduction and consistency-based abduction. HT4 uses set-covering; i.e. the only thing added to worlds are literals that are found on pathways between inputs and outputs. Consistency-based approaches (e.g. the ATMS [DeKleer 1986]) adds to worlds all literals consistent with proof pathways, even if those literals are not required for building that proof. World-generation is a function of the number of conflicting assumptions made by the reasoner. As the percentage of unmeasured variables increases, the size of the input and output sets decreases. In consistency-based abduction, this has no effect on the number of assumptions made since consistency-based abduction offers assumptions for all variables. However, set-covering abduction make fewer assumptions since it adds fewer literals to worlds. Hence, for low-assumption policies (e.g. set-covering abduction), world-generation is reduced when the amount of data from the domain is reduced.

Also, note that only one of our two different temporal linking policies (IEDGE) generated the multiple worlds that we expected. In other words, the generation of multiple worlds is extremely sensitive to the choice of modeling constructs, and some constructs will not generate multiple worlds. Our initial reaction was that if

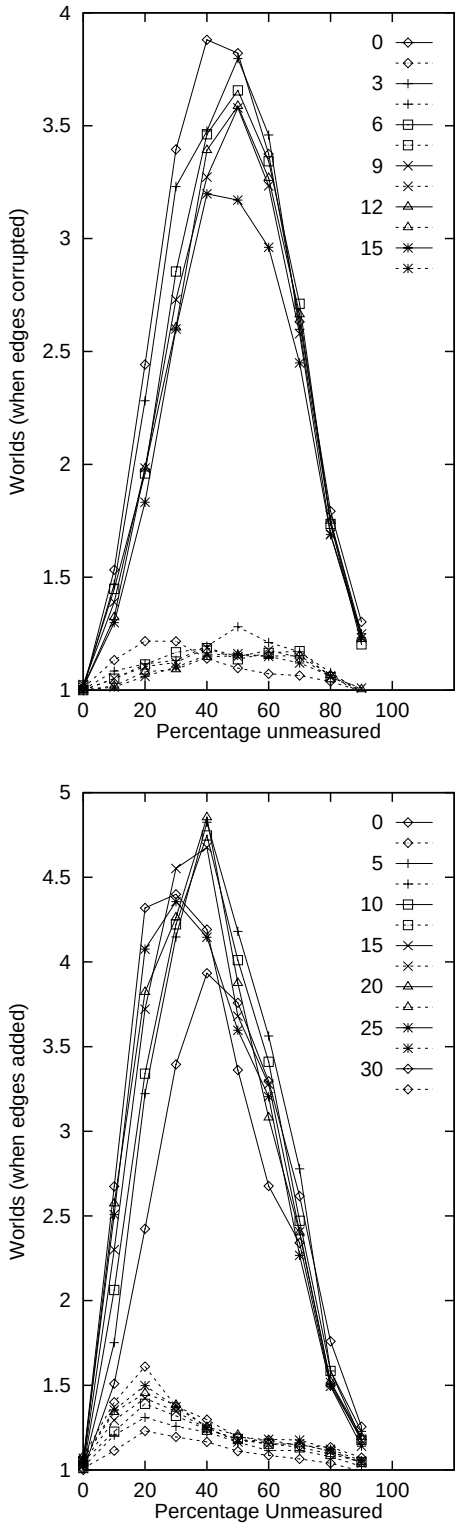

Fig. 6. Results from Experiment 1, showing the number of worlds generated. Results for IEDGE (solid lines) and XNODE (dashed lines) differ dramatically.

XNODE does not generate multiple worlds, then it is less expressive as a modeling language, and would be poorer at capturing all the behaviors in the original data. However, our next experiment contradicted this interpretation.

4.3 Experiment #2

For the second experiment, we explored whether models that generated multiple worlds were more expressive. We modified the graph-based abductive validation procedure, so that instead of returning the world(s) that explained the most number of outputs, we returned any single world, chosen at random. The results of that one-world abduction run were compared to the results gained from full multiple-world abduction. For this experiment, we used the same set of treatments as was used in experiment 1; i.e. another 1,050,000 runs.

The results are shown in Figure 7. In these graphs, the percentage of the input-output pairs that were covered in the worlds is shown on the y-axis (labeled *percent explicable*). For multiple-world abduction, the maximum percentage is shown; i.e. this is the most explanations that the model can support. For one-world abduction, the percent of the one-world (chosen at random) is shown. Note that, at most, many-world reasoning was ten percent better than one-world reasoning (in the IEDGE graph for U=40 and 10 edges corrupted). The average improvement of many-world reasoning over one-world reasoning was 5.6 percent. That is, in millions of runs over thousands of models, there was very little difference seen in the worlds generated using one-world and multiple-world abduction.

5. SIMPLER REQUIREMENTS CONFLICT EXPLORATION

The results of experiment 1 demonstrate that our graph based abductive validation does not suffer the extreme problems of indeterminacy of other abductive approaches. This is important because it strengthens our belief in the utility of graph-based abduction. If we generate many different worlds, each supporting different conclusions, our ability to refute the models is in doubt. This conclusion is supported by the gradients on the graphs in Figure 7: as we mutate the models to be less like the original, we increase the amount of data that cannot be explained in the model. Our validation procedure is clearly useful for inconsistent requirements models.

The results of experiment 2 seem counter-intuitive. Although each world represents a different way of resolving the inconsistencies, there was very little difference in the number of queries covered. In other words, each different world seems to give the pretty much the same answers. How can this be? Are our results distorted by our choice of case study or choice of scoring function? Our results are based on mutations of a single small model, *fisheries*. Perhaps an analysis of larger, more intricate models, would offer different conclusions? While we acknowledge this possibility, we note fisheries was just the initial model that seeded our mutators. Thousands of variants on fisheries were constructed, many of which were more complicated than fisheries (recall the first mutator added edges into the model).

Nevertheless, it is always appropriate to question the generality of experimental results. Methodologically speaking, in order to generalize our experiments, we need to isolate some principle that could apply to other models. We have identified one possible underlying principle, that we call funnel theory.

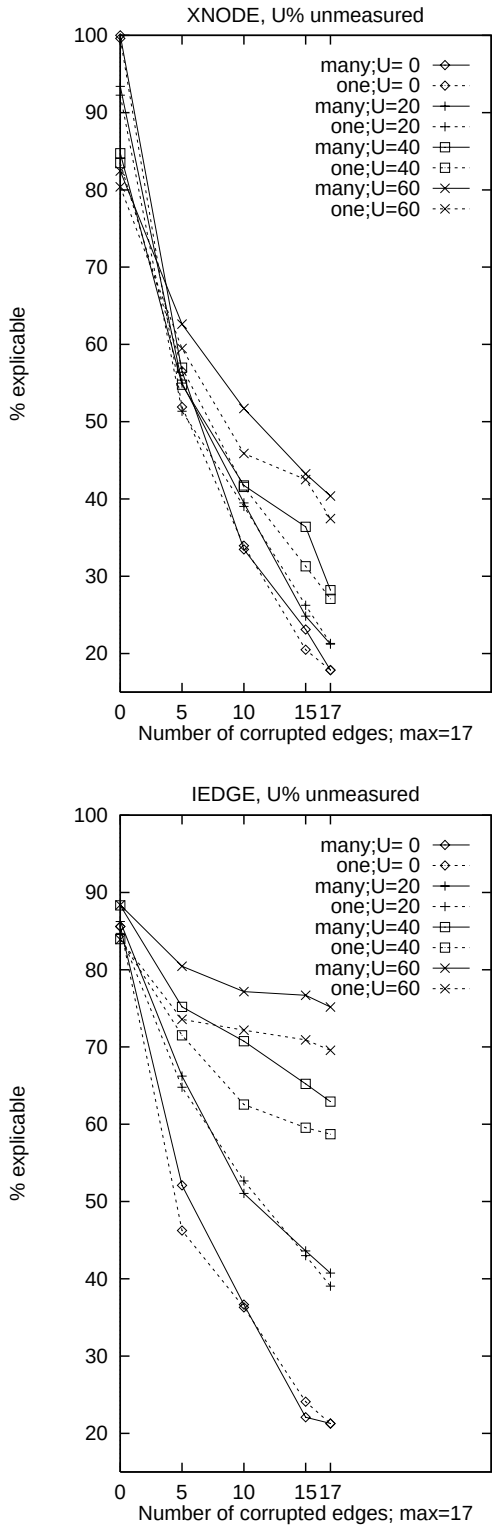

Fig. 7. Results of experiment 2, showing multiple-world abduction (solid line) vs one-world abduction (dashed line).

5.1 Funnel Theory

According to funnel theory, the pathways within our requirements models contain *very narrow funnels*; i.e.

—Most pathways converge to the same points.

—Inference outside the narrow funnels quickly runs down into the funnels

We can see part of such a narrow funnel around *fish catch* variable of Figure 4. Without *fish catch*, our model divides into 2 sub-networks. That is, all inferences from (e.g.) *boat purchases* to (e.g.) *fish growth rate* must pass through the funnel of *fish catch*.

Narrow funnels have two interesting properties that simplify conflict resolution:

(1) Narrow funnels dictate how arguments must be resolved around the funnel. Dr. Thick and Dr. Thin may argue for days on the nature of the inferences around *fish catch*. However, all that discussion is irrelevant if, to achieve some query, then (e.g.) *catch potential* must encourage *fish catch*.

(2) Narrow funnels let us ignore certain disagreements. Consider two arguments: one around a narrow funnel and another very peripheral to that funnel. The funnel argument might be resolved quickly (see the last point). Further, we need not spend much time on the peripheral argument since it is likely that most pathways will never use that peripheral part of the model.

Assuming that funnel theory is correct, the presence of the *fish catch* funnel in Figure 4 suggests that it is not surprising that the fisheries models generated so few worlds.

Elsewhere Menzies and Cukic have conducted an extensive literature review showing that funnel-like behaviour has been seen in numerous cases in software engineering and knowledge engineering [Menzies and Cukic 2000]. Further, there is some mathematical evidence to suggest that we should routinely expect funnels when searching indeterminate spaces [Menzies et al. 2000].

5.2 HT0

Funnel theory suggests that we can often rely on one world reaching as many goals as multiple worlds. HT0 [Menzies and Michael 1999] is a single-world abductive inference engine that exploits this "one world is usually enough" property.

Recall that HT4 build all worlds from maximal consistent subsets of all the possible pathways. Instead of building all worlds, HT0 builds them one at a time. When assumptions are required, HT0 takes random choices. HT0 terminates when the reached outputs for world W_i are about the same as the maximum reached outputs for worlds $W_1..W_{i-1}$.

HT0 executes over horn-clauses. Note that Figure 4 can simply be expressed in such horn-clauses; e.g. *fish density* encouraging *fish catch* could be encoded as:

```
t(fishCatch,up) :-
    t(fishDensity,up).
t(fishCatch,down) :-
    t(fishDensity,down).
```

```
 1 INPUT:  Theory,Inputs, Outputs,  File, N1,N2,N3
 2 OUTPUT: File
 3 (<A,Max>:=readBest(File)) or (A:=[];Max:=0)
 4 Facts := Inputs + Outputs;
 5 N1 times repeat
 6 { N2 times repeat
 7   { Covered := [];
 8     T1 := burn(Facts,copy(T))
 9     for Output in Outputs do
10     { if   <T2,A1>:=thrashBurn(Output,Inputs,T1,A)
11        then {add Output to Covered
12              A:= A1; T1:= T2;
13     } }
14     if    size(Covered) > Max
15     then  {Max := size(Covered);
16             append(File,<A,Max,Covered>)
17             if Max=100% then goto :stop}
18     Outputs:=permute(Outputs-Covered)+permute(Covered)
19     A:=change(first(N3,mostUsed(A)));
20   }
21   A := []; Outputs := permute(Outputs);
22 } :stop
```

Fig. 8. HT0

The HT0 algorithm is shown in Figure 8. In that figure, square brackets denote ordered sets and the `Permute` function randomly shuffles set order. A persistent store of old runs is maintained in `File`. If `File` already exists, then the best assumptions found to date are retrieved; else they are initialized (line 3). `N1, N2, N3` control the number of searches performed (the variable `Nall` will be used to denote `N1+N2+N3`). `N1-1` times, HT0 clears any old assumptions and randomly permutes the order of the outputs (line 21). Then, `N2` times, HT0 tries to prove each output in order (line 10). `ThrashBurn` is a depth-first search from *Output* to any member of *Inputs* across T_1. T_1 is generated (at line 8) from T by `burn`ing away all variable assignments inconsistent with known `Facts` (technically, this is node consistency [Mackworth 1977]). As `thrashBurn` searches, if new assumptions are found, they are added to A_1. When a horn clause is accessed, its sub-goals are *thrashed*; i.e. re-arranged randomly. This randomizes the direction of the depth first search from this point on. Also, when a new assumption is made, contradictory assumptions are *burnt* away (i.e. removed via node consistency). The burning and the discovery of new assumptions creates T_2 and A_1 respectively. Lines 11,12 arrange that if $Output_i$ is explained, T_2 and A_1 are used for the subsequent searches for $Output_j$ $(i < j)$. That is, searches for $Output_j$ explore a smaller space than $Output_i$. Note that if $Output_i$ is explained and $Output_j$ is not, then the system does not backtrack to find other pathways to $Output_i$. However, we may get another chance to explain $Output_j$ since the next time through lines 7-19, we permute the order in which we explore the outputs (see line 18). Note, in line 18, when we reset the output order, we move the uncovered outputs to the front of the output list; i.e.

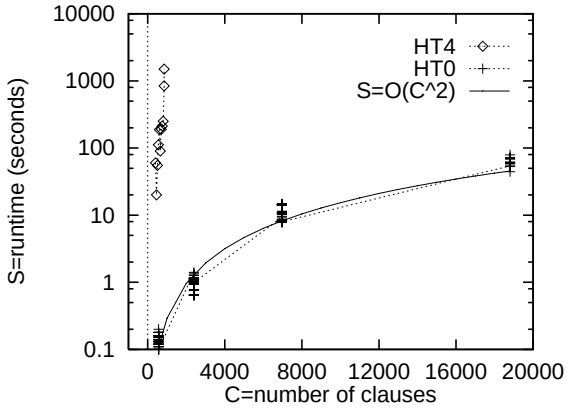

Fig. 9. Runtimes.

Model	Clauses	Literals	$\frac{sub-goals}{clause}$	$\frac{clauses}{literals}$
T1 [Smythe 1989]	558	273	1.6	2
Random 1	2390	688	1.6	3.5
Random 2	6961	1540	1.7	4.5
Random 3	18803	3394	1.7	5.5

Fig. 10. Some models processed by HT0

next time through we give priority to things we could not prove this time through. The only other feature of note is line 19. The `N3`-th most used assumptions are changed so that the next time through lines 7-19, the proofs for *Outputs* are forced into other parts of the model.

HT0 can be used as an anytime conflict exploration algorithm. Assuming that each explanation supplies less information than the one before, HT0 could run while the analyst traced the `Max` covered value. At anytime, the best `Max` found to date would be available. Also, at anytime, running the system for longer would explore different parts of the model and (potentially) could find better worlds.

Real world and artificially generated models were used to test HT0. A real-world model of neuroendocrinology [Smythe 1989] with 558 clauses containing 91 variables with 3 values each (273 literals) was copied X times. Next, Y% of the variables in one copy were connected at random to variables in other copies. In this way, the models Random1, Random2, and Random3 (see Figure 10) were built using `Y=40`. When executed with `Nall` varied from 1 to 50, the $O(N^2)$ curve of Figure 9 was generated. We conclude that HT0 was $O(N^2)$ in these experiments since the R^2 for an $O(N^2)$ curve fit to the HT0 data was 0.98 while the R^2 for $O(N)$, $O(N^3)$, $O(e^N)$ were all < 0.82.

In a result consistent with the HT4 experiments, no increase in queries covered was detected above `Nall=5`. That is, (1) the anytime nature of HT0 may not be required since (2) what explanations HT0 can find, it seems to find very quickly.

6. SUMMARY AND CONCLUSIONS

In exploring the utility of multiple viewpoints, we have found it useful to distinguish between use of multiple viewpoints during elicitation and their use during modeling and analysis. For the former, viewpoints can be used to represent different stakeholder's contributions, and to provide traceability back to an authority for each piece of information [Gotel and Finkelstein 1997]. For the latter, viewpoints can be used to model and analyze inconsistent information. Viewpoints offer a number of other benefits for requirements modeling, including the use of multiple representation schemes, multiple problem structures, and the ability to partition the modeling process itself. However, if there is no inconsistency, then these benefits are essentially presentation issues: the same benefits could be achieved by taking projections and translations of a single, consistent model. That is not to say that such issues are trivial, but rather that it is the handling of inconsistency that makes viewpoints truly interesting for requirements analysis.

In this paper, we examined the question of modeling and analyzing inconsistent models created from multiple viewpoints, and in particular whether is is possible to validate inconsistent models. From our theoretical and experimental work with graph-based abduction, we draw two general conclusions: (i) that it *is* feasible to reason with and validate inconsistent requirements models, and (ii) that to do so is *much* cheaper than the initial complexity analysis led us to believe. These results are important because they mean that stakeholders need not be rushed into some premature and artificial unification of their differing views.

Firstly, we showed that if we move away from purely classical deductive reasoning, we can perform sound reasoning on inconsistent models. We described an abductive inference approach that allows us to validate an inconsistent model against domain data. The abductive reasoner only generates proofs that do not contain inconsistencies, sorting the possible proofs into consistent worlds. Effectively, we add a form of paraconsistent reasoning to an existing non-paraconsistent deductive logic. Despite the theoretical complexity of our reasoning algorithm, we showed that it can be applied to reasonable sized problems.

More importantly, our experimental results showed that frequently we do not need to incur the extra complexity of multiple world reasoning. Multiple worlds reasoning is only useful if the worlds are truly different. We have explored these issues using our abductive framework. Abduction can check if some explicitly named viewpoints are truly different: if the combined viewpoints don't generate multiple worlds in response to particular a particular query, then they are not truly different *with respect to that query.*

Experimentally, we have shown here that for a range of problems (different models ranging from correct to very incorrect, different fanouts, different amounts of data available from the domain, different temporal linking policies) multiple world reasoning can only generate marginally better results than one-world reasoning (ten percent or less). Hence, in the domain explored by these experiments, there is little or no value in fully exploring all the possible worlds. Further, if our explanation of this effect via funnel theory is correct, we expect that the result generalizes to many other domains. For domains where "one world is usually enough", we offer HT0 as a very simple conflict exploration tool. HT0 runs fast ($O(N^2)$), even for

very large models ($N = 20,000$).

Note that the result in experiment 2, and our exploration of HT0 show that one *world* often explains as much of the data as multiple worlds. This is not the same as saying that any *viewpoint* does the same. Our 'worlds' are extracted from multiple viewpoints merged together without resolving inconsistencies. Depending on how they are used, viewpoints tend to cover particular areas of interest (sub-domains). One world, extracted from multiple viewpoints, may contain data from all the viewpoints, and therefore may cover all of the sub-domains. We cannot therefore just discard other viewpoints in favor of one selected at random.

Our results allow us to extend the use of viewpoints beyond their demonstrated benefits for requirements elicitation. We can use multiple viewpoints as a way of maintaining multiple overlapping descriptions at any stage of software development, without worrying about resolving inconsistencies between them. Not only can we validate such inconsistent models against domain data, but in many cases, the inconsistencies have little impact on the validity of the overall model. We explained this counter-intuitive result via funnel theory. Our future work will explore whether funnel theory applies to other domains.

ACKNOWLEDGMENTS

We would like to thank Axel van Lamsweerde, Bojan Cukic, Tony Bonner, Marsha Chechik, Ric Hehner, Alessandra Russo, Albert Lai, Benet Devereux, Victor Petrovykh, and Christopher Thompson-Walsh for their detailed comments on earlier versions of this paper.

REFERENCES

Boehm, B., Egyed, A., Port, D., Shah, A., Kwan, J., and Madachy, R. 1998. A stakeholder win-win approach to software engineering education. *Annals of Software Engineering 6*, 295–321.

Bossel, H. 1994. *Modeling and Simulation*. A.K. Peters Ltd.

Bylander, T., Allemang, D., Tanner, M., and Josephson, J. 1991. The Computational Complexity of Abduction. *Artificial Intelligence 49*, 25–60.

Chechik, M., Easterbrook, S., and Petrovykh, V. 2000. "Model-Checking Over Multi-Valued Logics". (submitted for publication).

Clancy, D. and Kuipers, B. 1997. Model decomposition and simulation: A component based qualitative simulation algorithm. In *Proceedings, AAAI National Conference on Artificial Intelligence, AAAI-97*.

Damian, D., Eberlein, A., Shaw, M., and Gaines, B. 2000. Using different communication media in requirements negotiation. *IEEE Software 17,* 3 (May/June), 28–36.

Darke, P. and Shanks, G. 1996. Stakeholder viewpoints in requirements definition: A framework for understanding viewpoint development approaches. *Requirements Engineering 1,* 2, 88–105.

DeKleer, J. 1986. An Assumption-Based TMS. *Artificial Intelligence 28*, 163–196.

Doyle, J. 1979. A truth maintenance system. *Artificial Intelligence 12*, 231–272.

Easterbrook, S. 1991a. Elicitation of requirements from multiple perspectives. Ph.D. thesis, Imperial College of Science Technology and Medicine, University of London. Available from `http://www.cs.toronto.edu/~sme/papers/`.

Easterbrook, S. 1991b. Handling conflicts between domain descriptions with computer-supported negotiation. *Knowledge Acquisition 3*, 255–289.

Easterbrook, S. 1995. Coordination breakdowns: why groupware is so difficult to design. In *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences*. 191–199.

Easterbrook, S., Lutz, R., Covington, R., Kelly, J., Ampo, Y., and Hamilton, D. 1998. Experiences using lightweight formal methods for requirements modeling. *IEEE Transactions on Software Engineering 24,* 1 (January), 4–14.

Easterbrook, S. and Nuseibeh, B. 1996. Using viewpoints for inconsistency management. *BCS/IEE Software Engineering Journal 11,* 1 (January), 31–43.

Feldman, B., Compton, P., and Smythe, G. 1989. Hypothesis Testing: an Appropriate Task for Knowledge-Based Systems. In *4th AAAI-Sponsored Knowledge Acquisition for Knowledge-based Systems Workshop, Banff, Canada.*

Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., and Nuseibeh, B. 1994. Inconsistency handling in multi-perspective specification. *IEEE Transactions on Software Engineering 20,* 8, 569–578.

Gabow, H., Maheshwari, S., and Osterweil, L. 1976. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering 2*, 227–231.

Gaines, B. and Shaw, M. 1989. Comparing the conceptual systems of experts. In *International Joint Conference on Artificial Intelligence, (IJCAI '89).* 633–638.

Goguen, J. and Linde, C. 1993. Techniques for requirements elicitation. In *1st IEEE International Symposium on Requirements Engineering (RE'93).* 152–164.

Gotel, O. and Finkelstein, A. 1997. Extended requirements traceability: Results of an industrial case study. In *International Symposium on Requirements Engineering (RE'97)* (January 6-10, 1997). 169–178.

Grundy, J., Hosking, J., and Mugridge, W. B. 1998. Inconsistency management for multiple-view software development environments. *IEEE Transactions on Software Engineering 24,* 11, 960–981.

Hunter, A. and Nuseibeh, B. 1998. Managing inconsistent specifications: Reasoning, analysis and action. *ACM Transactions on Software Engineering and Methodology 7,* 4, 335–367.

Jackson, M. 1995. *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices.* Addison Wesley.

Kakas, A., Kowalski, R., and Toni, F. 1998. The role of abduction in logic programming. In *Handbook of Logic in Artificial Intelligence and Logic Programming 5*, C. H. D.M. Gabbay and J. Robinson, Eds. Oxford University Press, 235–324.

Kotonya, G. and Sommerville, I. 1992. Viewpoints for requirements definition. *IEE Software Engineering Journal 7*, 375–387.

Kuhn, T. 1962. *The Structure of Scientific Revolutions.* Cambridge Press.

Mackworth, A. 1977. Consistency in Networks of Relations. *Artificial Intelligence 8*, 99–118.

Makinson, D. C. 1994. General patterns in nonmonotonic reasoning. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. Gabbay, C. Hogger, and J. Robinson, Eds. Vol. 3. Oxford University Press, 35–110.

Menzies, T. 1996a. Applications of abduction: Knowledge level modeling. *International Journal of Human Computer Studies 45*, 305–355.

Menzies, T. 1996b. Applications of abduction: Knowledge level modeling. *International Journal of Human Computer Studies.*

Menzies, T. 1996c. On the practicality of abductive validation. In *Proceedings of the European Conference on AI (ECAI'96).*

Menzies, T. and Compton, P. 1997. Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models. *Artificial Intelligence in Medicine 10*, 145–175.

Menzies, T. and Cukic, B. 2000. Adequacy of limited testing for knowledge based systems. *International Journal on Artificial Intelligence Tools (IJAIT).* (to appear).

Menzies, T., Cukic, B., Singh, H., and Powell, J. 2000. Testing indeterminate systems. International Symposium on Software Reliability Engineering (ISSRE-2000).

Menzies, T. and Michael, C. 1999. Fewer slices of pie: Optimising mutation testing via abduction. In *11th International Conference on Software Engineering and Knowledge Engineering (SEKE'99)* (June 17-19).

Mortensen, C. 1995. *Inconsistent Mathematics.* Kluwer Academic.

Mylopoulos, J., Cheng, L., and Yu, E. 1999. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM 42,* 1 (January), 31–37.

Nuseibeh, B. and Easterbrook, S. 2000. Requirements engineering: A roadmap. In *Proceedings of International Conference on Software Engineering (ICSE-2000).*

Nuseibeh, B., Easterbrook, S., and Russo, A. 2000. Leveraging inconsistency in software development. *IEEE Computer 33,* 4, 24–29.

Nuseibeh, B., Kramer, J., and Finkelstein, A. 1994. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering 20,* 10, 760–773.

Nuseibeh, B. and Russo, A. 1999. Using abduction to evolve inconsistent requirements specifications. *Austrialian Journal of Information Systems 7,* 1.

O'Rourke, P. 1990. Working notes of the 1990 spring symposium on automated abduction. Tech. Rep. 90-32, University of California, Irvine, CA. September 27, 1990.

Plexousakis, D. 1993. Semantical and ontological considerations in telos: a language for knowledge representation. *Computational Intelligence 9,* 1 (February).

Popper, K. 1963. *Conjectures and Refutations: The Growth of Scientific Knowledge.* Routledge and Kegan Paul.

Rich, C. and Feldman, Y. 1992. Seven layers of knowledge representation and reasoning in support of software development. *IEEE Transactions on Software Engineering 18,* 6 (June), 451–469.

Robinson, W. and Pawlowski, S. 1999. Managing requirements inconsistency with development goal monitors. *IEEE Transactions on Software Engineering 25,* 6, 816–835.

Ross, D. 1985. Applications and extensions of sadt. *IEEE Computer 18*, 25–34.

Rushby, J. 1995. Formal methods and their role in the certification of critical systems. Tech. Rep. SRI-CSL-95-1, Computer Science Laboratory, SRI International, Menlo Park, CA. Mar.

Shaw, M. 1997. *WebGrid: a WWW PCP Server.* Knowledge Systems Institute, University of Calgary, `http://Tiger.cpsc.ucalgary.ca/WebGrid/WebGrid.html`.

Smythe, G. 1989. Brain-hypothalmus, Pituitary and the Endocrine Pancreas. *The Endocrine Pancreas.*

Spanoudakis, G., Finkelstein, A., and Till, D. 1999. Overlaps in requirements engineering. *Automated Software Engineering 6,* 2, 171–198.

Stamper, R. 1994. Social norms in requirements analysis: an outline of measur. In *Requirements Engineering: Social and Technical Issues*, M. Jirotka and J. Goguen, Eds. Academic Press, 107–139.

van Lamsweerde, A., Darimont, R., and Letier, E. 1998. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering 24,* 11, 908–926.