A Model for Decision Maintenance in the WinWin Collaboration Framework[1]

Prasanta Bose

Center for Software Engineering
Department of Computer Science
University of Southern California

(bose@sunset.usc.edu)

Accepted for 10th KBSE Conference, November 1995

Abstract

Cost-effective engineering and evolution of complex software must involve the different stakeholders concurrently and collaboratively. The hard problem is providing computer support for such collaborative activities. The WinWin approach being developed and experimented at the USC Center for Software Engineering provides a domain independent solution for the stakeholders to cooperate in the requirements engineering phase of the software lifecycle. The key ideas in the WinWin approach and its support are: i) defining a win-win process for obtaining requirements through collaboration and negotiation, ii) defining a decision rationale model using a minimal set of conceptual elements, such as win conditions, issues, options and agreements, that serves as an agreed upon ontology for collaboration and negotiation defined by the winwin process, and, iii) defining a support framework, based on manipulation of explicit representation of the decision rationale and reasoning about it.

A major problem confronted in the WinWin framework is aiding decision coordination - coordinating the decision making activities of the stakeholders. A key element in supporting decision coordination is decision maintenance. As decisions undergo evolution, the effects of such changes on existing decision elements must be determined and the decision structure appropriately revised. This paper presents an approach to addressing the problem of supporting decision maintenance. The key ideas involve a) defining an extended ontology for decision rationale, that models the WinWin decision space and their states, b) formally describing a theory based on that ontology that specify conditions for states to hold, and c) defining an agent that utilizes the theory to determine revisions and coordinate with other agents to propagate revisions in a distributed support framework.

1 Introduction

In today's rapidly evolving technology and competitive marketplace - it has become critical that the design of complex software involve the users, designers, maintainers, and

1.This research is sponsored by the Advanced Research Projects Agency (ARPA) through Rome Laboratory under contract F30602-94-C-0195 and by the Industrial Affiliates of the Center for Software Engineering.

other stakeholders concurrently in the initial stages of design when requirements and system design decisions are made [4, 5, 6, 9]. The challenge is providing collaboration support that aid in such decision making. As noted by Fred Brooks, in his article “No Silver Bullets” [1], significant gains in software productivity are likely to be based on tools for aiding organization and design rather than traditional software engineering approaches. Research in the area of design rationale and models of argumentation and their support [10, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23] can be applied to meet such a challenge. The primary focus of most of these models have been the recording of the design decisions and their rationale. To facilitate such recording, the models typically make distinctions on the issue (topic/goal/question) focusing the decision, the options (position/alternative/claims) and the argument for the position. The support for such models typically involve use of hypermedia support frameworks.

In concurrent engineering of software, apart from recording the design rationale, additional problems arise from the need to collaborate and coordinate in the decision making process involving the different stakeholders as well as deal with decisions that span both requirements for the software and its design[2]. The decision rationale is no longer a static record of decisions made, that only serves for documentation purposes but rather an active model of the requirement decisions, the design decisions and trade-offs that constantly evolve with changes in requirements, rationale, and etc.

An adequate model of the decision rationale to aid in concurrent engineering of software must therefore address four important questions: i) How individual stakeholder objectives and their interactions get modeled ii) How options, tradeoffs, and agreements on requirements and design decisions get modeled and how the space of such decisions get collaboratively explored. iii) How changes in the decision rationale entities, e.g. objectives, rationale, etc., and their effects on other decisions get modeled. Such a model is critical to addressing the problem of decision maintenance required to coordinate the decision making of the involved stakeholders, and iv) How requirements and design decisions get modeled and their relationships.

Any solution to the above problems based on formal structuring of the entities [9] leads to complexity in the corresponding support system and consequent cognitive overload on the part of the users. At the other extreme, completely unstructured representations (example use of natural language) leads to intractability in the support mechanisms.

This paper briefly describes the WinWin approach that makes use of semi-structured representations to addressing the first two questions and presents the decision rationale model used in such an approach. We then focus on the problem of decision maintenance in the context of WinWin and present a solution for the problem based on an ontological description of the decision rationale, and a formal theory based on that ontology that aids in determining effects of changes in WinWin objects.

2. From here onwards we use the term decision rationale to include decisions on requirements and design.

2 The WinWin Approach

The WinWin approach [7] is aimed at addressing collaboration for the requirements engineering phase of the software life-cycle. The three key ideas in the approach are:

- **WinWin Spiral process.** The process [6] consists of two main steps: i) Identifying stakeholders and their win conditions ii) Creating win-win relationships collaboratively and negotiating to confront win-lose and lose-lose situations. A key aspect of the process is that it introduces economic, product quality and risk considerations into the decision making steps and introduces tradeoff exploration into the process to address risks and conflicts. The process identifies the domain independent conceptual elements that forms the agreed upon ontology of decisions for collaboration and negotiation and also identifies the domain taxonomy, as a given, for providing a shared understanding of the domain. The taxonomy is used as a domain-specific reference model with respect to which stakeholders express their win conditions, issues and agreements.
- **Decision rationale model.** The decision space that gets collaboratively explored in a WinWin process is modeled using four main conceptual objects: i) Win Condition - capturing the desired objective of the individual. ii) Issue - capturing the conflict between win conditions and their associated risks and uncertainties. iii) Option - capturing a strategy for resolving an issue. iii) Agreement - capturing the agreed upon set of conditions which satisfy stakeholder win conditions and also define the system objectives. The ontology also defines a set of relations between these objects. Figure 1, shows a typical abstract structure of the decision rationale in terms of the above entities and the link types denoting the relations between them. As shown in the figure, an issue (I) is related to one or more win conditions (W_x and W_y) through the *involves* relation. An option (O) for resolving an issue (I) is related to the issue through the *addresses* relation. An agreement (Ag_i) based on an option choice (Op) is related to Op through the *adopts* relation. The agreement (Ag_i) has a *covers* relation with a win condition and a *replaces* relation to any previous agreement it substitutes.

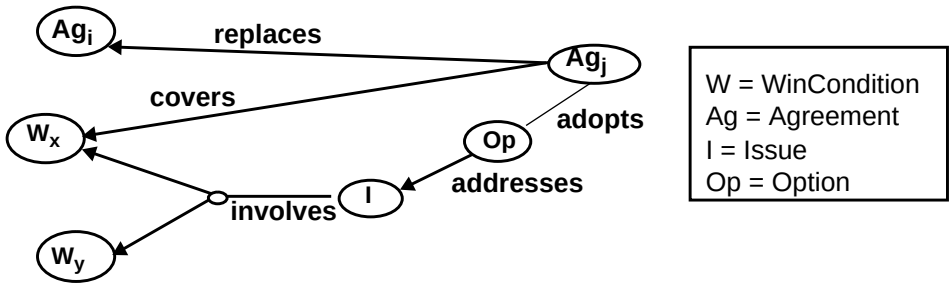

FIGURE 1. The WinWin decision objects and relations between them.

- **WinWin support framework**. Figure 2 shows a schematic diagram of the support framework that implements the WinWin concept of collaboration. As shown in the figure, each stakeholder is associated with a WinWin client. The clients communicate via the WinWin Router. A stakeholder interacts with the WinWin client support system interface to define their individual win conditions, raise issues, suggest options and draft and vote on agreements. The WinWin objects created or revised by the

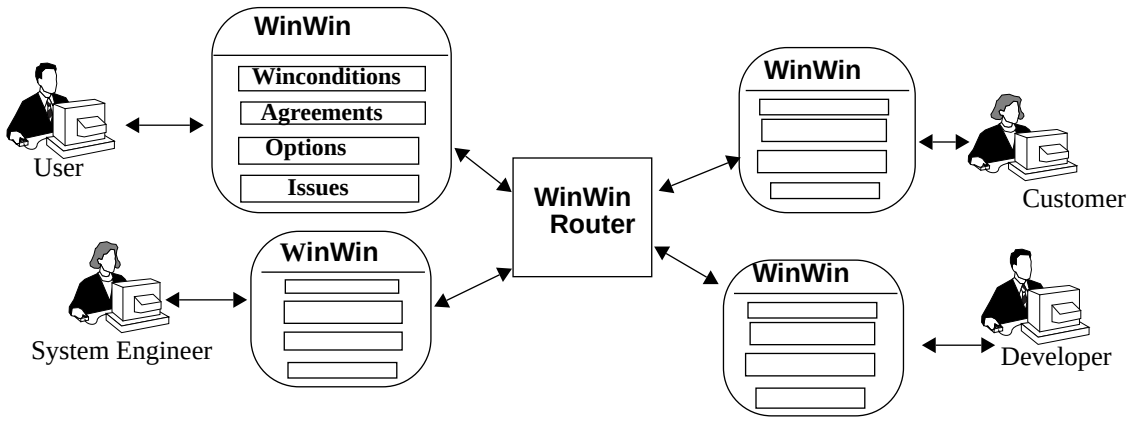

FIGURE 2. A Schematic diagram of the WinWin architecture.

stakeholder are recorded in a local database by the WinWin client. Any update operation performed by a stakeholder are noted by the client and used to notify other clients and update their object base using the WinWin Router. Hence each client, in essence keeps a copy of all the WinWin objects (winconditions, issues, etc.) - all of which may not be modifiable by the associated stakeholder. The issues raised are used to focus negotiation between stakeholders. WinWin also provide tools to support the negotiation activity [7]. Issue resolution leads to negotiated changes in win conditions. When all issues have been resolved, agreements are drafted and closed. The closed points of agreement, in concert with backup reference documents, then define the system requirement specifications.

2.1 Decision Maintenance Problem in WinWin

Given the above WinWin framework for collaboration, let us consider an example scenario that describes the two main services that the WinWin support system must provide to aid collaboration:

1) *Decision rationale structure capture*: Figure 3.(i) shows the evolving WinWin decision structure based on objects and named links, as a win condition, W, on software portability is entered by a stakeholder, an issue, I, on efficiency is raised, an option, O, based on using portability layers is considered by the stakeholders and finally the issue being resolved by an agreement, A, to restrict portability requirements to the interface layer. The WinWin decision structure must be explicitly captured since it forms the basis for aiding goal directed collaboration and negotiation.

2) *Decision state maintenance*. Figure 3.(ii) shows the corresponding changes in the decision state and the associated dependencies. As shown in the figure, addition of a win condition, W, on portability followed by addition of an unresolved issue, I, on efficiency leads to the win condition W, being at-issue. Resolving the issue, via an

option leads to an agreement causing the win condition to be covered and considered to be in a valid state[3]. Here the decision state undergoes revision with decision update

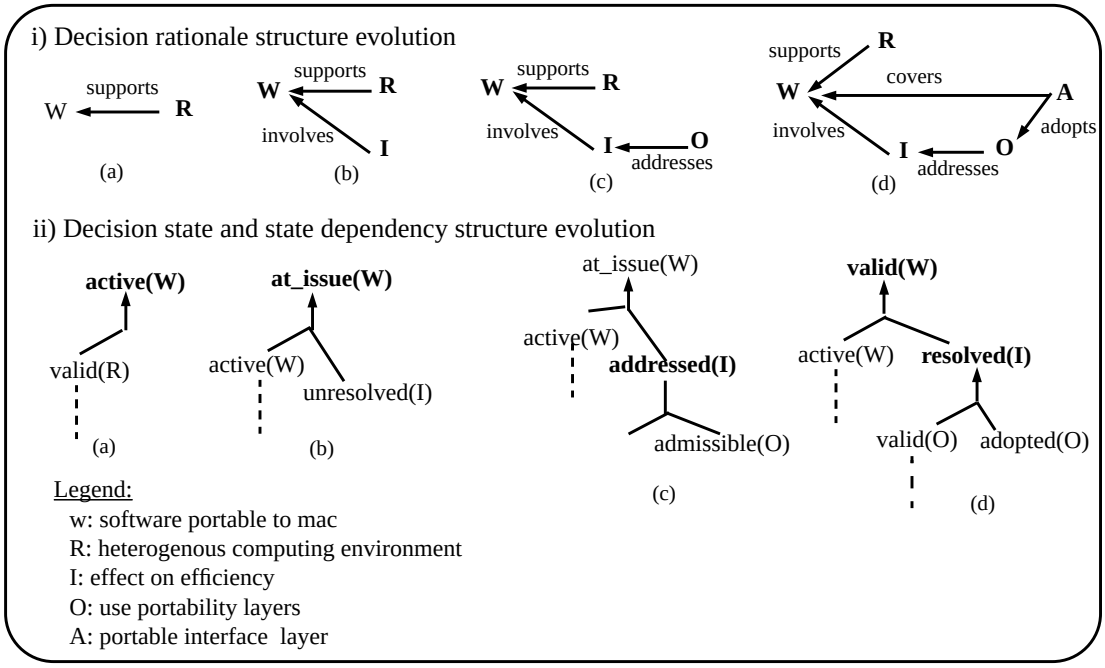

FIGURE 3. Decision rationale evolution with changes in decisions.

and aiding such state revision is key to supporting decision coordination.

The first service is provided in the the support framework discussed at the beginning of Section 2. The WinWin system aids in defining the objects and the directed named links in Figure 3.(i) by using a set of schema templates (for details, please refer to [7]).

The second service requires automated decision maintenance and is not supported in the system discussed above. In the rest of this paper we present a model for automated decision maintenance in the WinWin framework. The model makes use of an extended and formal description of the ontology for decision rationale and a theory based on that ontology that specify the conditions for states of the elements in the ontology. We then consider an extended WinWin support framework, where decision maintenance service is provided via coordination of a set of agents that use the theory to infer effects of a change and propagate its effects.

3. It is important to note that the WinWin object state distinctions play a crucial role in determining and coordinating the decision-making activities of the different stakeholders involved in the WinWin process. A coarse-grain distinction based on for example IN/OUT states would fail to capture the intermediate steps in the decision making process - that correspond to issue identification, option generation, issue resolution, and other steps.

3 Ontology of WinWin Decision Rationale

As discussed in section 2, the WinWin approach identifies four basic object types that capture a stakeholder's win conditions, issues, options and agreements. In order to consider the effect of changes in the justification or rationale for a win condition, rationale for an issue, and rationale for choice or rejection of an option, the base set of WinWin objects is extended to consider reason as a first class object. A generic constraint object is also introduced to aid in capturing structure where available. Constraints are used to elaborate win conditions, options and agreements. A constraint captures assertions that specify domain specific restrictions on the requirements and the design[4].

The extended ontology has the following elements:.

- Wincondition: encapsulates problem domain specific software functional and non-functional objectives and desired process constraints (e.g. schedule and cost)
- Reason: encapsulates domain specific arguments for a wincondition, arguments for the decision made pertaining to choice of an option, or rejection of the option and reason for an issue raised
- Issue: encapsulates domain specific issues that involve the winconditions, options and agreements;
- Option: encapsulates a domain specific choice for resolving an issue
- Agreement - Encapsulates the decision based on negotiated option
- Constraint - the constraints imposed on the objectives and design solution

Apart from these object types, there are two generic objects - strings and constraints.The constraint object is used to model restrictions and bindings on product and process parameters. Relations are used in specifying object attributes and link types between objects. Unary predicates are used to define the states of objects.

Figure 3 shows a portion of the ontology pertaining to the wincondition, issue, option and the agreement object. Each object type is defined by a set of tuples, where each tuple consists of the slot name denoting the relation name with a single or multiple cardinality and a typed value field that ranges over an object or enumerated value set. For example, in the win condition object W, the relation comment is a one-to-many function between win conditions and the set of all strings. Similarly the relation defined by the adopted-by slot in an option is a one-to-one function between the options and agreements. The state slot of each object is a one-to-many function from the object to an enumerated set of unary predicate constants. For example, a win condition can be in both active and at_issue state. The theory defining the state predicates is discussed in the following section.

4.The restrictions are expressed over the elements of the reference domain model and design solution model as represented using a reference domain and solution taxonomy in WinWin [6]. Such constraints can be used to identify issues caused by logical inconsistencies between them.

a) Wincondition(W):

Slot name	Cardinality	Value-Type
<name>	single	string
<Objective>:	single	strings
<comment>:	multiple	string
<sub-winc>:	multiple	win condition
<state>:	multiple	predicate
<rationale>:	multiple	reason
<covered-by>:	single	agreement
<involved-in>	single	issue
<replaces>	single	wincondition

State predicates: {active, covered, at-issue, retracted, valid, reduced}

b) Option(O)

Slot name	Cardinality	Value-Type
<name>	single	string
<body>:	single	string
<adopted-by>:	single	agreement
<assertion>:	multiple	constraint
<pros>:	single	string
<cons>:	single	string
<rationale>:	multiple	reason
<rejection>:	multiple	reason
<involved-in>:	multiple	issue
<state>:	multiple	predicate

State predicates: {admissible, valid, adopted, at_issue, rejected, retracted}

c) Issue(I):

Slot name	Cardinality	Value-Type
<name>	single	string
<body>:	single	string
<involves>:	multiple	{W, O, A}
<rationale>:	multiple	reason
<addressed-by>:	multiple	option
<state>:	multiple	predicate

State predicates: {resolved, unresolved, non-resolvable, addressed, retracted}

d) Agreement(A)

Slot name	Cardinality	Value-Type
<name>:	single	string
<body>:	single	string
<commitments>:	multiple	constraint
<covers>:	multiple	{W, A}
<involved-in>:	multiple	issue
<adopts>:	single	option
<state>:	multiple	predicate
<replaces:>	single	agreement

State predicates: {open, valid, committed, at_issue, replaced, retracted}

FIGURE 4. A subset of the WinWin ontology for Decision rationale.

3.1 Theory Defining State Predicates

The states of the objects in the WinWin decision structure database are impacted by the updates that a stakeholder makes to states of objects and relations between the objects. The updates are captured via changes in the slot values of existing objects or creation of new objects. Now the problem is determining the effects of the updates and their propagation through the database. Determining the effects of such changes can be relegated to the stakeholder based on a shared understanding of an informal description of the states. Such an approach leads to computational overload on the stakeholders when the decision rationale database becomes complex and hence is not scalable. An alternative approach, that we adopt, is to make explicit the theory defining the states[5]- that can then be exploited to infer effects of changes by an automated agent. Table 1 shows part of the theory that defines how states of objects depend on relations among elements of the

5. Note that we are assuming that the states and their definition is part of the agreed upon domain independent ontology for collaboration. The basis for the choice of the states and their definitions is presented in [8].

TABLE 1. Theory defining the states of a subset of the winwin objects.

i) Win Condition(I)

active(W) $\Leftrightarrow$ [[$\forall$R:reason, rationale(W, R) Λ valid(R)] V told("active(W)", Wdb)] $\Lambda \neg$ retracted(W)
at_issue(W) $\Leftrightarrow$ active(W) Λ [$\exists$ I:issue, involved-in(W, I) $\rightarrow$ unresolved(I)]
replaced(W) $\Leftrightarrow\exists$ W':wincondition, replaced-by(W, W') Λ valid(W') $\Lambda \neg$ retracted(W)
valid(W) $\Leftrightarrow \neg$ replaced(W) Λ active(W)
Λ [["I:issue[involved-in(W, I) $\rightarrow$ resolved(I)] Λ covered(W)]
V [reduced(W) $\rightarrow \forall$W':wincondition, subwinc(W, W') Λ valid(W')]]
covered(W) $\Leftrightarrow \exists$ A:agreement, covered-by(W, A) Λ committed(A)
retracted(W) $\Leftrightarrow$ told("retracted(W)", Wdb) V [$\exists$ R:reason, rationale(W,R) Λ invalid(R)]
V [$\exists$ I:issue, involved-in(W, I) $\rightarrow$ non-resolvable(I)]
V [$\exists$ W':wincondition, replaces(W, W') $\rightarrow$ retracted(W')]
V [$\exists$ W':wincondition, replaces(W, W') $\Lambda \exists$A: agreement, covered-by(W,A)
$\rightarrow$ retracted(A)]

ii) Issue(I)

unresolved(I) $\Leftrightarrow \neg$retracted(I)
Λ [$\neg \exists$ O:option[addressed-by(I, O) Λ admissible(O) $\Lambda \neg$ rejected(O)]
V $\exists$ O:option [addressed-by(I, O) Λ valid(O) $\Lambda \neg$ adopted(O)]
addressed(I) $\Leftrightarrow \exists$ O:option, addressed-by(I, O) Λ admissible(O) $\Lambda \neg$ rejected(O)
resolved(I) $\Leftrightarrow \exists$ O:option, addressed-by(I, O) Λ valid(O) Λ adopted(O)
nonresolvable(I) $\Leftrightarrow \forall$ O:option [addressed-by(I, O) Λ rejected(O)]
retracted(I) $\Leftrightarrow \forall$ X [involves(I, X) Λ retracted(X)]

iii) Option(O)

admissible(O) $\Leftrightarrow$ told("admissible(O)", Wdb)
suboptimal(O) $\Leftrightarrow$ [$\exists$ I:issue, involved-in(O, I) Λ unresolved(I)] V
[$\exists$ R:reason, rationale(O, R) $\Lambda \neg$ valid(R)]
optimal(O) $\Leftrightarrow$ [$\forall$ I:issue, involved-in(O, I) Λ valid(I) $\rightarrow$ resolved(I)] Λ adopted(O)
adopted(O) $\Leftrightarrow \exists$A: agreement, adopted-by(O, A) L committed(A).
rejected(O) $\Leftrightarrow$ told("rejected(O)", Wdb) V [$\exists$ R:reason, rejection(O, R) Λ valid(R)]
valid(O) $\Leftrightarrow$ admissible(O) $\Lambda \neg$ rejected(O)
Λ [$\forall$R:reason, rationale(O, R) Λ valid(R)]
$\Lambda \neg$ [$\exists$ I:issue, involved-in(O, I) Λ unresolved(I)]

iv) Agreement(A)

open(A) $\Leftrightarrow$ told("open(A)", Wdb) $\Lambda \neg$ committed(A)
committed(A) $\Leftrightarrow$ told("committed(A)", Wdb) $\Lambda \neg$ retracted(A)
valid(A) $\Leftrightarrow$ [$\exists$ option:O, adopts(A, O) Λ valid(O)] Λ committed(A)
Λ [$\forall$ X:assertion, commitment(A, X) Λ valid(X)]
$\Lambda \neg$ [retracted(A) V at_issue(A)]
at_issue(A) $\Leftrightarrow$ [$\exists$ I:issue, involved-in(A, I) Λ unresolved(I)]
V [$\exists$ W:wincondition, covers(A, W) Λ at_issue(W)]
retracted(A) $\Leftrightarrow$ told("retracted(A)") V retracted(Option)
V [$\forall$ X: wincondition, covers(A, W) Λ retracted(W)]
V [$\exists$ I:issue, involved-in(A, I) Λ nonresolvable(W)]
replaced(A) $\Leftrightarrow \exists$ A':agreement, replaced-by(A, A') Λ valid(A') $\Lambda \neg$ retracted(A)

ontology and their state slot values. For example, a win condition, W, is considered to be in at_issue state iff there is some issue, I, which is linked to W through the involved-in relation and I is unresolved. The theory also captures the notion of failure to resolve an issue - hence a win condition is considered retracted if the stakeholders failed to resolve an issue (non-resolvable state of I) involving the win condition. There are other elements of the theory that aims to capture optimality of decisions and needs a more detailed discussion, beyond the scope of this paper.

3.2 State Dependency Structure

A given decision rationale structure representation in WinWin only makes explicit the object instances, their states and relations. The theory then implicitly defines a state dependency structure for the states of the objects in the structure. Figure 5 shows an example of the dependency structure for a win condition, W, on computer driven interactive interface, to be in a valid state. The valid state of the win condition is justified by the win condition being active (depends on having a valid rationale), the related issue, I, being resolved and the win condition being covered by a committed agreement. The issue I is resolved due to the option O, addressing I is valid. The option O is valid since it is admissible and there is no valid reason for its rejection and their is a valid rationale for O. The implicit nature of the state dependency arises from the fact that the WinWin object representation does not maintain an explicit representation of the justification for the states.

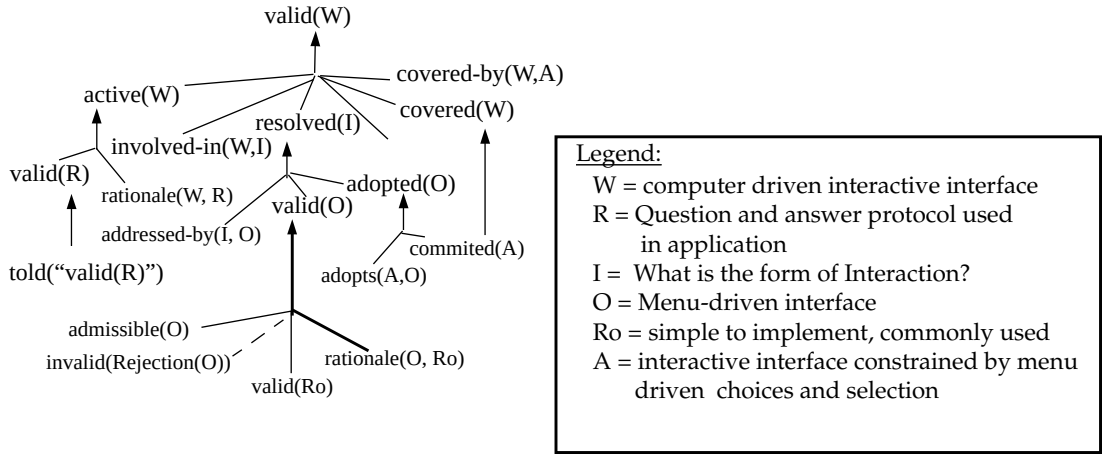

FIGURE 5. State dependency structure for a win condition to be in valid state.

4 WinWin Decision State Revision and Change Propagation

Decision maintenance for the implicit state dependency structure require revisions of the states such that the states can be justified relative to the theory and the changed decision database. The basic steps taken to achieve consistency are: i) localized inference of new states that are consistent with the theory, in the immediate neighborhood of the object where update occurs and ii) revision of the existing states for consistency with respect to

the inferred states, and propagation of the revisions through the existing dependency links in the structure.

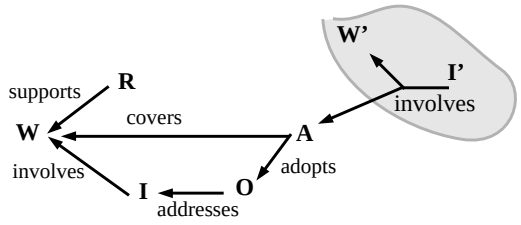

FIGURE 6. An example of a decision structure with an update.

To illustrate the above revision process - we take the example given in Figure 5 to be the initial state and consider the situation (Figure 6) where a new win condition, W' is entered and leads to an unresolved issue, I', involving the existing agreement A and W'. Using the theory on the state of agreements - the update to the agreement involved-in slot leads to inferring that the agreement is in at_issue state and is no longer valid. The agreement still remains committed and hence the rest of the states that depend on the agreement being in committed state remain unchanged. If in the course of decision making - the issue I' becomes non-resolvable then A becomes retracted. Propagating the effect of the agreement being retracted to the option (O), O is no longer adopted and becomes unresolved. Further propagation, using the theory relevant to win conditions leads to the win condition being no longer valid and being in at_issue state.

It is important to note, why truth maintenance techniques, that are well studied in the field of Artificial Intelligence [11,12] does not work here. The primary reason is that the truth maintenance techniques depend on the caching of justifications for an assertion (here the state predicate assertion). As discussed above, such justifications are only partially represented (given by the relations defining the link types) and hence the theory is required for state determination.

4.1 Decision Maintenance Service Agent

The model developed above can be used to define an automated agent that provides the decision maintenance service. In a distributed WinWin framework - such an agent will reside in each WinWin client that interacts with a specific stakeholder and the agent must coordinate with the other service agents associated with other clients to propagate the state changes.

The operation of the agent in each client site is defined by the method[6] given in Table 2. The agent is triggered by updates that are locally posted or by messages it receives to propagate a change from other client sites. For the clarity of the presentation we have ignored issues of concurrent updates and other issues that arise in agent coordination. The

6. The method assumes that there are no cycles in the decision dependency structure.

TABLE 2. The steps of the method to propagate state changes.

Method: Propagate-Change(W_x, C_x)

Input: i) an update C_x pertaining to winwin object W_x
ii) Theory, Th, defining the states
Output: revisions or messages to propagate a change

Steps:
1. Find WS, the set of objects W_i that are immediately related to W_x and includes it.
2. *Do forall* W_i *in* WS:
a) Let S = Inferred states of W_i given the relations, state of W_x and the Theory Th
b) *If* owned-locally(W_i)
then i) revise state slot to be consistent with S
ii) *If* state of W_i is revised to produce change C_i,
then call Propagate-change(W_i, C_i)
else i) let Ci = revision relevant to Wi
ii) send message = propagate-change(W_i, C_i) to the client agent that owns the object

input to the method is an update that specifies a change to a slot (the relevant slots here are ones which model relations between objects and state of an object) and the theory defining states of objects. Step 1 retrieves the set S of immediately related objects. Step 2 iterates over S and takes action to revise their states where required. It has two substeps. In substep 2 (a), the states of the related object Wi are infers their states in the context of Wx. Step 3 revises their states to be consistent with the inferred states. If the object whose state needs revision is not locally owned - the agent process sends a message to propagate change to the agent which owns it. The last step requires the WinWin router (Figure 2) to maintain a name space which identifies the agent owning a particular object.

5 Summary and Future Work

The research presented in this article identifies the problem of decision maintenance in the context of the WinWin framework for collaborative requirements engineering and presents a solution to the problem. The key ideas in the solution involve i) defining the ontology of decision rationale in WinWin ii) defining a a theory based on that ontology that specifies the conditions for decision states. iii) defining an agent that utilizes the theory to infer revisions and propagate the effect of changes via communication with other agents that support collaboration in a distributed WinWin support framework. Current work is focused on implementing these agents in an extended WinWin support framework. Some of the directions for future work are i) defining a model for aiding decision space exploration based on the existing ontology ii) domain independent model of the notion of optimality of decisions and how to provide feedback to users on lose of optimality due to changes in decisions and iii) how states of objects and the theory can be used to provide support for incremental backtracking over decisions, and iv) definition of a set of agent services for coordination and collaboration in concurrent engineering of software.

Acknowledgments. The author gratefully acknowledges the helpful comments of Dr. Barry Boehm and support of the WinWin team at the Center for Software Engineering for implementation of the WinWin system.

6 References

1. [Brooks, 1987]. F. P. Brooks Jr., "No Silver Bullet: Essence and Accidents of Software Engineering", *IEEE Computer*, September 1987, pp 10-19.
2. [Boehm, 1988]. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *Compute*r, May 1988, pp. 61-72.
3. [Boehm et. al., 1993]. B. W. Boehm, P. Bose, E. Horowitz, W. Scacchi, S. Bendifallah and A. Madni, "Next-Generation Software Processes and Their Environment Support", *Proceedings, USC Center for Software Engineering Inaugural Convocation*, June 1993.
4. [Boehm et. al., 1994a]. B. W. Boehm, P. Bose, E. Horowitz, and M-J. Lee, "Experimental Results from a Prototype Next Generation Process Support System", *TRW SIG Technology Review Journal*, Volume 2, Number 1, 1994.
5. [Boehm et. al., 1994b]. B. W. Boehm, P. Bose, E. Horowitz, and M-J. Lee, "Software Requirements As Negotiated Win Conditions", *Proceedings, 1994 International Conference on Requirements Engineering*, IEEE, April 1994.
6. [Boehm et. al., 1994c]. B. Boehm and P. Bose, "A Collaborative Spiral Software Process Model Based on Theory W" ,*3rd International Process Conference*, 1994.
7. [Boehm et. al. 1995]. B. Boehm, P. Bose, Ellis Horowitz and Ming June Lee "Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach", *IEEE Proceedings of the 17th ICSE Conference*, 1995.
8. [Bose 1995]. P. Bose "A Decision Rationale Model For Collaboration in Concurrent Engineering of Software: An Extended WinWin Model", USC-CSE Technical Report, under preparation.
9. [Cutosky, et. al 1993], Cutosky, Engelmore, Gruber, Genesereth, Mark, Tenenbaum and Weber, "PACT: An Experiment in integrating concurrent engineering systems", *Computer*, Vol. 26, No. 1, 1993.
10. [Conklin-Begeman, 1988]. J. Conklin and M. Begeman, "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," *ACM Trans. Office Info. Systems*, Oct. 1988, pp. 303-331.
11. [Curtis et. al. 1988]. B. Curtis, H. Krasner, N. Iscoe, "A Field Study of the Software Design Process for Large Systems", *CACM*, Vol. 31, No. 11, November 1988, pp 1268-1287.
12. [DeKleer, 1986]. J. de Kleer, "An Assumption-based TMS", Artificial Intelligence, 28:127-163, 1986.
13. [Doyle, 1979], J. Doyle, "A Truth maintenance System", Artificial Intelligence, 12, 1979.
14. [Fisher et. al., 1991] G. Fisher, A. C. Lemke, R. McCall and A. I. Morch, "Making Argumentation serve Design", *Human Computer Interaction*, 1991, 6(3/4), pp. 393-420
15. [Gruber, 1992] T. R. Gruber amd D. M. Russell, "Generative Design Rationale: Beyond the record and replay paradigm", in *Design Rationale*, Lawrence Erlbaum Associates
16. [Guindon, 1988] R. Guindon and B. Curtis, "Control of Cognitive Processes during Software Design: What tools are needed?" *CHI* 1988.
17. [Kunz. and Rittel, 1970], W. Kunz and H.W.J. Rittel, "Issues as Elements of Information Systems", Institute of Urban and Region Development, Working Paper 131.
18. [Lee, 1990]. J. Lee, "SIBYL: A Qualitative Decision Management System," in *Artificial Intelligence at MIT: Expanding Frontiers*, P. Winston and S. Shellard, eds., MIT Press, 1990, pp. 106-133.
19. [MacLean, et. al, 1991] A. MacLean, R. Young, V. Bellotti and T. Moran, "Questions, Options and Criteria: Elements of a design rationale for user interfaces" *Human Computer Interaction*, 6(3/4), 201-250.

20. [McCall, 1986], R. McCall, “Issue-serve systems: A descriptive theory for design”, *Design Meth. Theories* 20(8), 443-458.

21. [Ramesh-Dhar, 1992]. B. Ramesh and V. Dhar, “Supporting Systems Development by Capturing Deliberations During Requirements Engineering,” *IEEE Trans. SW Engr.*, June 1992, pp. 498-510.

22. [Stefik et al., 1987]. M. Stefik, G. Foster, D.G. Bobrow, K. Kahn, S. Lanning, and L. Suchman, “Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings,” *Comm. ACM*, Vol. 30, No. 1, Jan. 1987, pp. 32-47.

23. [Toulmin, 1958]. S. Toulmin, editor, “*The Uses of Argument”*, Cambridge University Press, UK 1958.