Foundations of the WinWin Requirements Negotiation System

by

Ming-june Lee

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Science)

August 1996

Abstract

This dissertation summarizes the results of applying several formal modeling capabilities to the WinWin system, and identifies the system improvements resulting from the analysis. WinWin is a groupware support system driven by the WinWin spiral process model. It enables multiple stakeholders to collaborate and negotiate requirements in an incremental and evolutionary way. It uses "win conditions" to capture individual stakeholder objectives. "Issues" are provided by the system to capture sets of conflicting win conditions along with "options", which are possible resolutions to issues. An "agreement" adopts options chosen by stakeholders to resolve an issue and to reconcile the win conditions involved in that issue. These artifacts provide scalable structure for a groupware tool and accommodate changes to requirements.

Initial use of WinWin uncovered several anomalous usage situations. These made it feasible and important to formally model these artifacts and operations to provide solid foundations of the system. The research described in this thesis models the WinWin requirements negotiation infrastructure and dynamics. It involves formal descriptions of multiple views for the WinWin requirements negotiation system,

including win condition interaction in the requirements space view, artifacts and their relationships, artifact life cycles and the equilibrium model. The results showed that these models improved the system in three primary ways: 1) fully understand the lower-level interactions of the system's features; 2) prevent aberrant behavior of the system; and 3) provide process guidance for the users to avoid problem situations and efficiently reach win-win solutions. While these models are formulated, a big challenge for a multi-view framework is to maintain consistency. This thesis also presents how the relationships among the various views of WinWin are determined in order to reconcile them into an integrated model. The reconciliation methodology can be generalized and applied to other multi-view frameworks.

Contents

List Of Tables

List Of Figures

Part I

Overview

This part provides an overview on motives and background for this dissertation. It previews how this research tackles the challenges that other approaches were not able to address before. Chapter 1 outlines the problem and the objective of this dissertation. This dissertation focuses on formally modeling the WinWin system. WinWin is a requirements negotiation system driven by the WinWin spiral process model. It enables multiple stakeholders to collaborate and negotiate system objectives in an incremental and evolutionary way. Initial use of WinWin uncovered several anomalous usage situations. These made it feasible and important to formally model these artifacts and operations to provide solid foundations of the system as presented. Chapter 2 shows the evolution of requirements engineering and highlights related approaches aiming at collaborative requirements engineering. Chapter 3 describes the WinWin requirements negotiation system and how it addresses the collaborative requirements engineering issues with the WinWin spiral process model and the support system.

Chapter 1

Introduction

1.1 Problem Statement

Requirements Engineering (RE), which provides a systematic framework for representing and acquiring software requirements, constitutes an important part of Software Engineering. A prime motive for requirements engineering is early error detection, which saves development effort as illustrated in Figure 1.1[Boe81]. Research [NKF94, BR89, Dav90, T+96, EGR91] indicates that the following issues are critical to requirements engineering.

Multi-stakeholder considerations: In a collaborative software development environment, perspectives of all stakeholders should be integrated and reconciled. Past requirements engineering approaches focusing on user requirements failed in that users' perspectives may not match other stakeholders' perspectives and

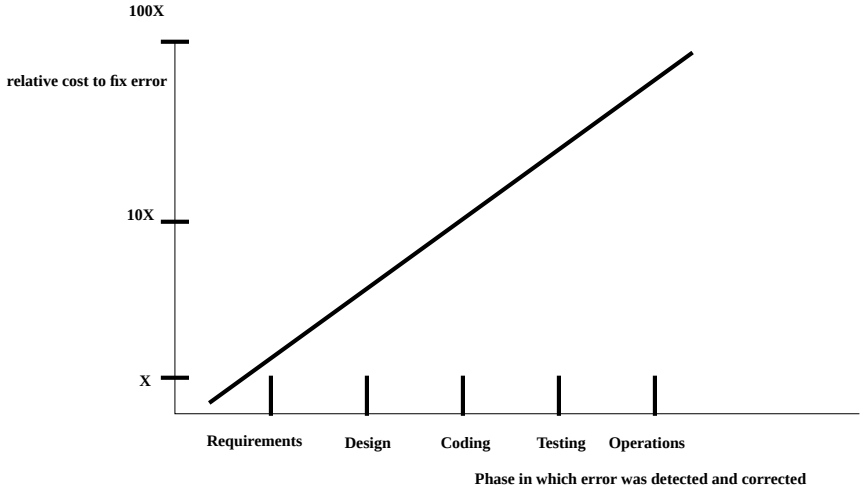


Figure 1.1: Relative Cost Decreases If Error Is Detected Early in the Software Life Cycle

thus result in a win-lose situation. In addition, other stakeholders' requirements are often misinterpreted because their representatives are not involved in the requirements engineering activity.

Change management: Very often, critical criteria are changed in the midst of requirements formulation. For example, a budget cut would invalidate some previous agreements. Therefore, change management is necessary to accommodate changes in objectives, constraints, or alternatives. In addition, the rationale for previous requirements needs to be incorporated to help determine how to change requirements.

Groupware support: To facilitate requirements negotiation across organizations, groupware that institutionalizes *computer supported cooperative work (CSCW)*

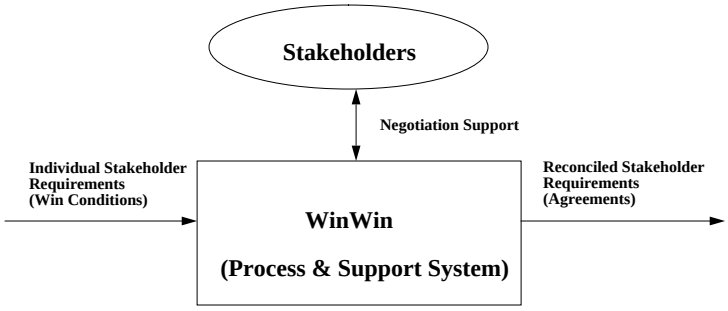


Figure 1.2: The WinWin requirements negotiation system overview

is needed to bridge the geographical and time gap as well as to achieve group decisions.

The USC-CSE (Center for Software Engineering at the University of Southern California) WinWin requirements negotiation system[BBHL94b, BBHL95] takes in individual stakeholder requirements (win conditions) and helps produce reconciled stakeholder requirements (agreements), as illustrated in Figure 1.2. In order to achieve this goal, "issues" are provided to capture sets of conflicting requirements and "options" are possible resolutions to "issues." The system, on the top level, is driven by the WinWin Spiral Process[BBHL95], to accommodate *change* of requirements. In addition, the supported WinWin artifacts keep track of status, and a WinWin equilibrium model facilitates convergence toward a win-win solution. As *groupware*, the system provides negotiation support, message passing and multi-media document navigation.

The development approach for the USC WinWin System to date has primarily involved exploratory prototyping. Several experiments[BBHL94a, Buc94] demonstrated that the WinWin System facilitates requirements negotiation. The system is now converging on a relatively stable set of artifacts and relationships. This makes it feasible and important to elaborate the underlying process model and to formalize these artifacts and relationships to provide solid scientific foundations for the WinWin system. This is the focused problem addressed by the research presented in this thesis.

1.2 Research Objective

The research objective of this thesis is to elaborate and formalize the WinWin requirements negotiation process and artifacts, to provide solid foundations for the WinWin system. This thesis presents several formal and semi-formal models of the WinWin system and its operations. The first part, including this chapter, provides an overview on the motives and background for this thesis. It previews how the research tackles the challenges that other approaches were not able to address before. Chapter 2 summarizes the evolution of requirements engineering, and highlights related approaches aiming at collaborative requirements engineering. Chapter 3 describes the WinWin Spiral Model, its operational concepts and the support system

to show how the WinWin system addresses the collaborative requirements engineering issues. The second part summarizes the results of my research efforts on applying several formal modeling capabilities to the WinWin system, and identifies the system improvements resulting from the analysis. Chapter 4 offers a problem space view that models the inter-win-condition relationships. Chapter 5 describes major WinWin artifacts and their inter-relationships for supporting the requirements negotiation infrastructure. Chapter 6 illustrates how WinWin artifact evolves in the negotiation process. Chapter 7 proposes the WinWin equilibrium model to guide the WinWin users to resolve issues and reach agreements in the negotiation. Chapter 8 provides a framework for determining the relationships among the various models or views presented in the previous chapters. Chapter 10 summarizes the contributions made with this research and discusses future works.

Chapter 2

Background

2.1 Evolution of Requirements Engineering

2.1.1 Classical Requirements Engineering

Requirements engineering is a relatively recently-established practice. Occasional early software projects such as SAGE had good requirements engineering approaches [Ben56], but their significance was not generally appreciated. In the 1960's, researchers started to recognize the significance of requirements definition and review in the software development process. Experience showed that the traditional "build and fix" approaches turned out to be extremely costly and inefficient. Without specifying and analyzing system and software requirements,one is very likely to obtain an undesired system. In 1970, Royce published a classic paper[Roy70] characterizing the waterfall model. It incorporated requirements acquisition and analysis as key steps in the software development life cycle. Several requirements definition and

traceability tools emerged in the early 1970's. The 1976 International Conference on Software Engineering (ICSE-2) marked requirements engineering a full-fledged subfield in the software engineering discipline. The subsequent dedicated issue of IEEE Transactions on Software Engineering (IEEE-TSE January 1977) exhibited representative requirements approaches of that time including $SADT^{TM}$ (Structure Analysis and Design Technique)[RKS77] by SofTech, PSL/PSA (Problem Statement Language/Analyzer)[TV77] by University of Michigan and SREM (Software Requirements Engineering Methodology)[Alf77] by TRW.

SADT was motivated by the belief that "a problem unstated is a problem unsolved." Ross and colleagues contended that major difficulties in software development were caused by lack of an adequate approach to requirements definition. SADT provides a graphical notation to model the hierarchic structure of a system. It documents the problem to be solved by the software system and the solutions proposed by the stakeholders. It guides the stakeholders to think about the problem and convey their understanding to others in the team via context analysis, functional specification and design constraints. For certain classes of problems, SADT helped encompass all aspects of a task in advance and resulted in a more productive team and more effective management.

Teichroew and Vicena wrote a notable paper[TV77] presenting the primary goals and functions of PSL/PSA. PSL/PSA is a computer-aided technique for structured requirements specification, documentation and analysis of information processing

systems. They designed the Problem Statement Language (PSL) to define the objects and relationships involved in the proposed system. These objects and relationship are entered into in a database using the Problem Statement Analyzer (PSA). Various reports on the results of the PSA can be generated upon request. In several cases, the showed that by making use of computers, the quality of the documentation was improved and the cost of design, implementation and maintenance was reduced.

SREM was developed by TRW to address the problems of correctly specifying the requirements for large software systems such as real-time weapon systems. TRW analyzed problems encountered in its software requirements, and proposed the Requirements Specification Language (RSL) and Requirements Engineering and Validation Systems (REVS) to address those problems, and to produce testable functional and performance requirements. RSL is a formalized language to reduce ambiguity and serve as input to the support software—REVS. REVS incorporate tools to check completeness and consistency of the requirements, maintain traceability to originating requirements and simulations, and generate simulations to validate the correctness of the requirements. The key contributions were that the project: 1) identified activities and their required inputs and expected outputs necessary for generating real-time software requirements 2) drew a clear distinction between requirements and design and 3) developed a methodology, a formal language and a support system and demonstrated their adequacies to address realistic problems.

2.1.2 Transitional Requirements Engineering

The previous section reviews requirements engineering as it emerged to be a recognized sub-discipline in software engineering. The early approaches were generally confined by the sequential concerns of the Waterfall Model, which contended that requirements specification should be completed before implementation. Swartout and Balzer, however, provided evidence of the inevitable intertwining between *requirements specification* and *implementation*[SB82]. First, specification is confined by the available implementation technology. For example, a particular data structure constrains the size of data it can handle. Second, implementation choices expand the specification. An instance is that use of an existing pattern-matching package will add to the specification the inclusion of a wildcard character. Swartout and Balzer argued that attempting to construct an aid keeping complete specifications and implementation separate would have trouble capturing specification changes that are forced by implementation decisions. They concluded that interleaving specification and implementation into a single development structure will result in a more coherent and realistic structure.

Boehm and colleagues conducted an experiment on *prototyping* versus *specifying* [BGS84]. In their experiment, seven teams developed versions of the same small-size applications. Four teams used the *specifying* approach and 3 others used the *prototyping* one. They demonstrated the following main results.

1. The prototyped products were with roughly equivalent performance but less than half the code size and effort,

2. The prototyped products were rated lower on functionality and robustness but higher on ease of use and ease of learning,

3. Specifying resulted in more coherent design and software for future integration.

The above agreed with Swartout and Balzer's argument that the model of complete specifications before implementation should be reconsidered. Software development depending solely on specifications encounters difficulties in application areas where it is hard to specify requirements in advance such as user-interface. The users really cannot tell what they want until they see it. However, the specification-oriented approach should not be completely thrown away, especially on large projects since it does render robustness, functionality, coherent design and ease of integration. It just needs to reorient to accommodating prototyping. [BGS84] concluded that the selection of the specific mix of prototyping and specifying should be driven by risk-management.

2.2 Collaborative Requirements Engineering

The following sections discuss research approaches according to the three issues recognized before as critical to today's requirements engineering. The category of

each work does not necessarily suggest that it will not support other issues but just accents its specialized area.

2.2.1 Multi-stakeholder Consideration

IBIS(Issue-Based Information System) addresses multi-stakeholder consideration by supporting relations among system objectives. It was firstly developed by Horst Rittel and colleagues [CY91, KR70]. The central structure is called "issue." Issues can be viewed as requirements that impact on design decisions. Each issue is linked with its supporting arguments and opposing arguments. It helps capture planning dialogue for design rationale that is important in group decision-making. gIBIS [CY91] and REMAP[RD92] are both graphical tools for supporting the IBIS structure with some extension. Although IBIS structures support analysis of requirements interactions, no tools are provided for analyzing trade-offs, so the design decision may overlook optimal solutions. There is also no negotiation strategy embedded to reconcile different perspectives.

The Nature (Novel Approaches to Theories Underlying Requirements Engineering) project[P+94] is a joint effort among institutes in Europe. They aim at applying AI and related techniques to requirements engineering. The underlying framework proposed by Klaus Pohl[Poh93] presents three important dimensions of requirements engineering addressed in NATURE: 1) the specification dimension, 2) the representation dimension, and 3) the agreement dimension.

The third dimension "agreement" of NATURE best maps to the multi-stakeholder consideration domain. Two related NATURE works will be discussed here. Viewpoint[NKF94] presents a framework similar to WinWin on requirements engineering for composite systems based on use of multiple viewpoints. Its focus is more on reconciling the heterogeneous categories of requirements than the contents (semantics of requirements). It does not provide a formal process model like the WinWin Spiral model or the elaborated WinWin Equilibrium to drive the reconciliation. [JK94] is another work in NATURE adopting the Quality Function Deployment (QFD) approach and IBIS structure to address multi-stakeholder consideration. In addition, it adopts "House of Quality (HoQ) Items" that serve as Domain Taxonomy to localize possible conflicting requirements. It embeds HoQ and QFD as negotiation strategies to converge different perspectives. One of the major problems with this approach is that the limited scope focusing on customers brings potential conflicts when other stakeholders join the work later in the software life cycle. Another limitation lies on the difficulty of automating the original pencil-paper approach. A great percentage of negotiation is still done off-line. Using customers' common sense instead of a well-founded trade-off analysis tool as the quality driver is very likely to produce unsatisfactory outputs.

Two additional software engineering arenas also address multi-stakeholder consideration. One is Participatory Design (PD) that encourages users (and maybe other stakeholders) to participate in the design of social computer systems[MK93].

CISP[MA93] falls into this category to facilitate cooperative and interactive storyboard prototyping. Fischer[F+92, Ret93] and colleagues have experimented with a domain-oriented environment consisting of (1) a human-centered CATALOGEXPLORER, (2) an EXPLAINER utilizing users' and programmers' strength and (3) a MODIFIER that integrates construction kit, argumentive hypermedia component, catalog component, and simulation component. Class[AC93] stresses codevelopment with joint meetings and remote prototype systems. Another arena drawing great attention is Joint Application Design (JAD) that enables people across organizations to incorporate their design ideas. It is exemplified by FASE 2000[Boz92] that offers common data model and rapid prototyping. The major deficiency of PD and JAD is that current works focus primarily on the interactions between users and designers. Other stakeholders' concern such as budget and schedule constraints set by customers are not well incorporated. There are also no trade-off analysis tools provided.

2.2.2 Incremental and Evolutionary Acquisition

ARIES (Acquisition of Requirements Incremental Evolution of Specifications) is codeveloped by USC/ISI and Lockheed Sanders to provide a knowledge-based assistant in the front end of system life cycle [H+92]. In ARIES, "folder" and "workspace" structures are provided to capture representation of multiple models. Folders and workspaces are related to the domain taxonomy in the WinWin approach that will

be discussed in details in the following chapter. ARIES facilitates incremental acquisition but does not address risk items that may potentially cause problems.

Tuiqiao[PTA94], an inquiry-based system, is developed by Potts and colleagues in Georgia Institute of Technology. It performs requirements analysis using the Inquiry Cycle Model. This model produces requirements documentation consisting of requirements, scenarios and other information via a Q-A session attended by stakeholders. This Q-A session may result in changes that demand requirements evolution and re-documentation. Requirements are expected to be negotiated and refined by cycling through this model. This model, however, may have breakdowns if system engineers overlook some important risk items in the Q-A session.

2.2.3 Groupware

This section discusses groupware tools that focus on providing a cooperative working environment. The groupware tools can be categorized into the following types:

Communication tools: facilitate the communication of project participants (such as electronic message system), bridge the time and space gap (such as video conferences), and provide a common working environment (such as computer white-board).

Common terminology and mediator: define common terms or provide mapping between systems so different components can communicate with the same

language; this category also includes integration work of heterogeneous environment.

Shared data: provides concurrency control, data linking, or cooperative editing system[KBH+92] for multiple users working on the same data.

COLAB(WYSIWIS)[S+87] exemplifies computer-aided meetings and on-line whiteboard facilities. Project Nick[C+88] claims that one of the reasons for meeting failure is that effects of a meeting are diverse due to inadequate documentation and lack of focus. It attempts to facilitate meeting progression by characterizing meetings and corresponding information. On the dimension of common terminology and mediator to enhance interoperability, Diplans[Hol88] proposes a coordination language for interoperable terminology. The Coordinator[FGHW88] drives actions with a semi-structure message system. On the facet of shared data, CES was designed to tackle this problem with a distributed Collaborative Editing System[KBH+92]. Lotus Notes facilitates shared document manager, replication algorithm, and customer contact tracking. Object Lens[LYM88] uses semi-structured objects combining object-oriented databases, hypertext, electronic messaging, and rule-based intelligent agents to provide a "spreadsheet" for cooperative work.

All these tools have had major difficulties in providing scalable information structures, where within which relations among system objectives can be established, examined and reasoned to reach a qualitative consensus. Even the most popular

tools like the Coordinator and Lotus Notes offer only semi-structured messages with limited navigation support. As these tools Thus, they lack any systematic approach to address cooperation between cross functional teams, as noted in [JK94].

2.3 Collaborative Requirements Engineering System Objectives

For addressing multi-stakeholder consideration as well as incremental and evolutionary acquisition, it is important to

- provide collaborative goals,
- address risks,
- support relations among system objectives,
- support trade-off analysis tools,
- embed negotiation theory,
- facilitate incremental and evolutionary development, and
- provide design rationale.

Table 2.1 summarizes representative requirements engineering approaches in Section 2.2.1 and Section 2.2.2 according to these terms.

Approach / Issues	ARIES	gIBIS	NATURE	Tuiqiao
provides collaborative goals	some	some	good	some
addresses risks	none	none	none	none
supports relations among system objectives	some	strong	good	good
supports trade-off anlysis tools	some	none	none	none
embeds negotiation theory	none	none	good	none
facilitates incremental and evolutionary development	strong	some	some	strong
provides design rationale	good	strong	good	good

Table 2.1: Issues Addressed in Requirements Engineering Approaches

2.4 Current Capabilities v.s. Objectives

As noted in Section 2.2.3, it is very important to provide a scalable information structure where relations among system objectives can be established, examined and reasoned to reach a qualitative consensus. Table 2.2 compares selected groupware tools based on how well each work facilitates requirements information.

System Features	Coordinator	Object Lens	Project Nick	Notes
information structure	good	good	some	some
organizational interaction	good	good	good	some
hypermedia representation	none	none	good	good
formal model underlying information structure	some	some	good	none
information dependency links	some	good	good	some
information database	none	strong	some	none
navigation support	none	good	none	none

Table 2.2: Groupware in Supporting Shared Requirements Information

Chapter 3

The WinWin Requirements Negotiation System

3.1 The WinWin Spiral Model

The WinWin Spiral Process Model is a Theory-W[BR89] extension to the Spiral Model[Boe88]. Theory-W is developed on the principle of *making everyone a winner*. Theory-W consists of nine steps which can be divided into three phases:

1. Establish a set of win-win preconditions.
   (a) Understand how people want to win;
   (b) Establish reasonable expectations;
   (c) Match people's tasks to their win conditions;
   (d) Provide a supportive environment.
2. Structure a win-win software process.
   (a) Establish a realistic process plan;

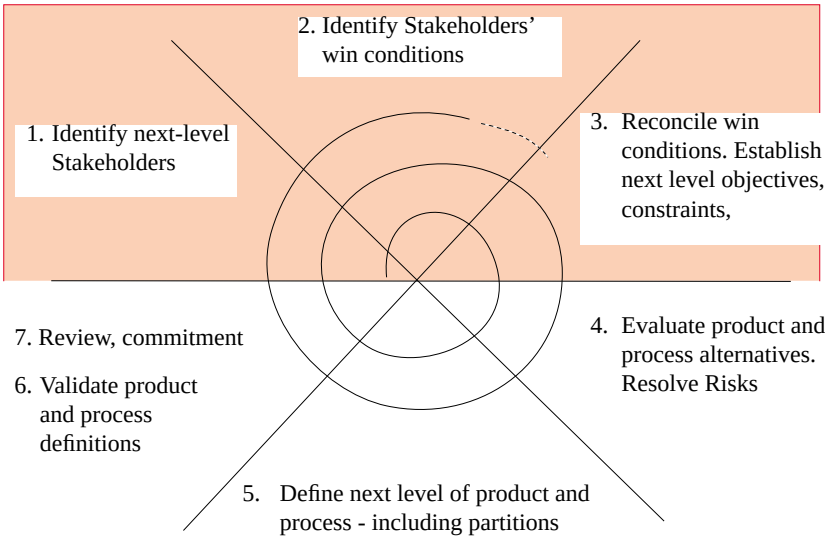


Figure 3.1: The WinWin Spiral Process Model

(b) Use the plan to control the project;

(c) Identify and manage your win-lose or lose-lose risks;

(d) Keep people involved.

3. Structure a win-win software product.

(a) Match product to users', maintainers' win conditions.

The nine steps of Theory-W process is then translated into the following Spiral Model extension, which is illustrated in Figure 3.1[BBHL94b, BBHL94a]:

Determine Objectives. Identify the system life-cycle constituents and their win conditions. Establish initial system boundaries, external interfaces.

Determine Constraints. Determine the conditions under which the system would produce win-lose or lose-lose outcomes for some constituencies.

Identify and Evaluate Alternatives. Solicit suggestions from constituents. Evaluate them with respect to constituents' win conditions. Synthesize and negotiate candidate win-win alternatives. Analyze, assess, and resolve win-lose or lose-lose risks.

Record Commitments. Record commitments and areas to be left flexible in the project's design record and life cycle plans.

Cycle Through the Spiral. Elaborate win conditions, screen alternatives, resolve risks, accumulate appropriate commitments, and develop and execute downstream plans.

The ideal goal of the WinWin Spiral Process Model is to satisfy all stakeholders' initial win conditions. A more practical goal is to satisfy all stakeholders reconciled win conditions to *make every one a winner.*

3.2 The Operational Concept

The operational concept for the WinWin system maps to the first 3 sectors, the shaded part of Figure 3.1, in the WinWin Spiral Model. The first group of arrows pointing from stakeholders to the WinWin system illustrate that stakeholders are

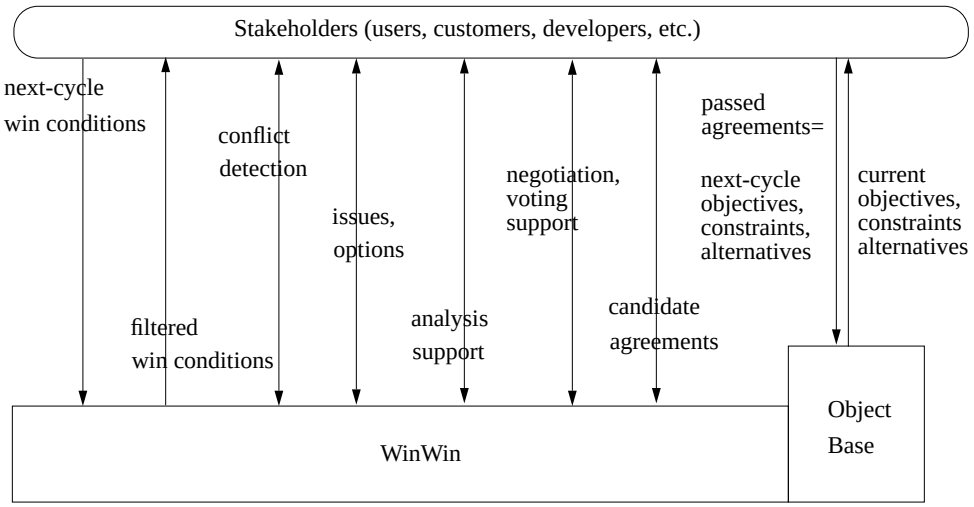


Figure 3.2: The WinWin Operational Concept

identified first. Their win conditions then are entered into the WinWin system. With the navigation support provided by the system, stakeholders are then able to filter win conditions according to their domain taxonomy groups in order to detect conflicts and identify possible Issues, and Options for resolving them. These candidate Issues and Options are then prompted to the stakeholders to start the reconciliation, the third sector in Figure 3.1. Negotiation is supported by summarizing options for resolving Issues as well as querying stakeholder's high-prioritized win conditions to drop options that are not acceptable. Trade-off analysis tools such as COCOMO support evaluating the many candidate options, to find out what combination of them will meet the established constraints and maximize the WinWin aspect. If such a combination is found and chosen, an agreement will be proposed to adopt the

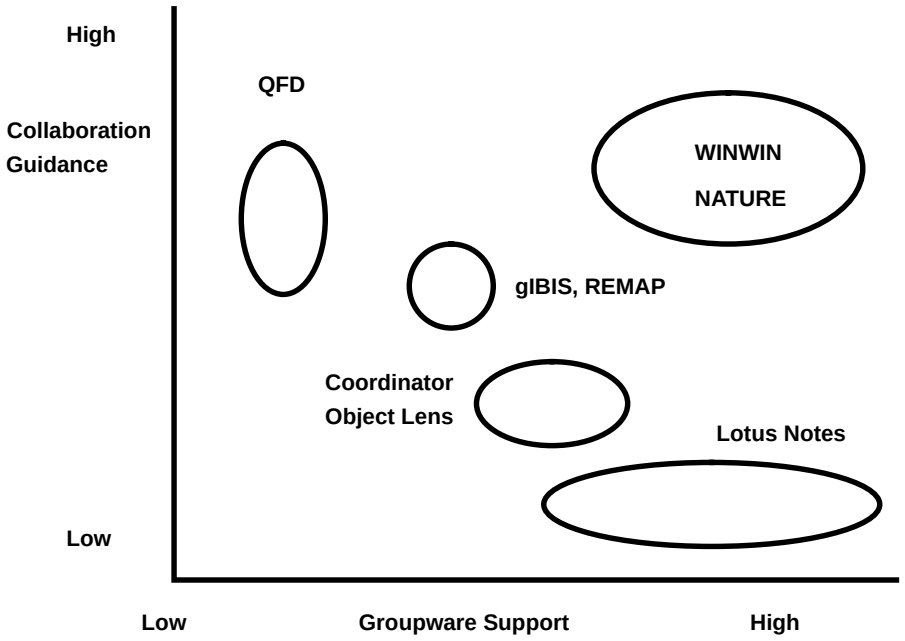


Figure 3.3: Needs versus capability comparison

chosen combination of options and stakeholders will vote on the agreement. If the vote is passed, the passed agreement will become a next cycle objective, constraint or alternative. If the vote is failed, the negotiation will continue until a win-win solution is found.

Figure 3.3 compares WinWin to selected related works discussed in the previous chapter in terms of groupware support and collaboration guidance. In summary, the WinWin requirements negotiation system is distinguished from most traditional requirements engineering approaches in the following aspects:

1. Traditional requirements engineering approaches emphasize on user requirements. There is insufficient support to acquire requirements from stakeholders like customers, maintainers, and interoperator. Nor is there negotiation theory embedded to reconcile multiple views. Collaborative goals are therefore incompletely supported in the traditional paradigm. The WinWin Spiral Process Model uses Theory-W to converge on system objectives and achieve view integration. In addition, trade-off analysis tool is provided to facilitate qualitative group consensus.

2. Traditional requirements are mostly rigid whereas win conditions in WinWin are negotiable. Design rationale is documented in the WinWin artifact to provide a corporate memory. Risks are explicitly addressed in WinWin to pinpoint possible breakdowns and propose early fixes. This makes it easier to increment and evolve requirements in the spiral model.

3. Past groupware tools do not connect with requirements engineering approaches very well because of the lack of scalable information structures for dealing with the requirements. The WinWin artifact structure can be easily exported to hypertext format and be viewed using web browsers with important information highlighted.

3.3 The WinWin Support System

A lot of insight for the WinWin support system resulted from our boot-strap experiments using Theory-W. In Summer 1993, a set of hypotheses on what kind of system functions best support the WinWin spiral process model was formulated. WinWin-0 was prototyped and used in a bootstrap exercise by Dr. Boehm, Dr. Horowitz, Dr. Bose, and Mingjune Lee to determine the requirements for WinWin-1[BBHL94a]. The outputs identified in the experiment converged on important features for WinWin-1, the first USC-CSE in-house version. WinWin-1 was designed and developed by Dr. Boehm, Dr. Horowitz, Dr Bose, Yimin Bao, Hoh In, June Sup Lee and Mingjune Lee. In WinWin-1, the following artifacts are supported:

- "Term": terminology definition that will be used in the negotiation.
- "Domain Taxonomy": a structure for modeling the domain and categorizing the following artifacts.
- "Win Condition": a system objective, constraint, or alternative that a stakeholder considers important or beneficial.
- "Conflict/Risk/Uncertainty(CRU)": an aggregate to capture the conflicts between win conditions/POA or the risk/uncertainty implied by any win condition; it is renamed "issue" in WinWin-95.
- "Option": an alternative for resolving a CRU.

- "Point of Agreement(POA)": an aggregate to cover reconciled win conditions/POA.

In addition to the basic artifacts, WinWin-1 provided functions like query-by-example for finding a particular set of artifacts that satisfies the input criteria. It also incorporated tools like COCOMO (COnstructive COst MOdel)[Boe81] to do trade-off analysis on the impact of an artifact, an audio tool to record a verbal explanation, and Mosaic to export the WinWin data to the web site.

WinWin-1 had several problems such as references were value-based rather than object-based. Value-based reference did not support referential integrity. Stakeholders could get dangling pointers if an artifact was deleted whereas its references were not. Any typing error could also result in a dangling pointer. If a stakeholder wanted to look at the contents of a particular reference, he/she had to either compose a query or exhaustively search through the artifact menu which was extremely inconvenient and error-prone. Another problem was that relationships in an artifact were combined and confusing.

WinWin-95[Hor96] was built by Dr. Horowitz, Dr. Curran, Yimin Bao, Eul Gyu Im, Hoh In, and June Sup Lee to upgrade the functions, stability, robustness, performance, user-interface and many other important features of WinWin-1. Several supplementary artifacts like "comment" and "vote" were added to enhance the communication between stakeholders. Relationships between artifacts were specifically identified and separated. The references were object-based to allow

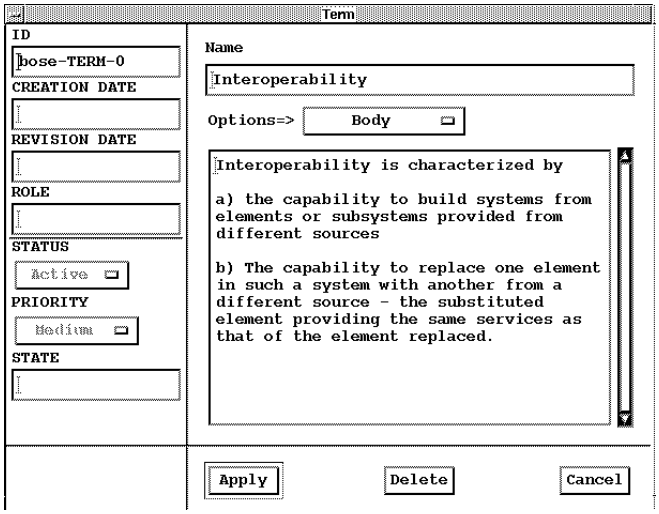


Figure 3.4: Term

navigation. In addition to "COCOMO," more tools could be easily incorporated using the "attachment" artifact. WinWin-95 also had the capability to export WinWin data to a major Unix word-processor "Framemaker" to generate hyper-linked requirements documentation.

The following sections work through screen dumps from a given scenario [UC94] to characterize this WinWin-95 support system. The example project is to develop a specialized SEE (Software Engineering Environment) in support of a SGS (Satellite Ground Station) product line.

3.3.1 Term

The WinWin-0 bootstrap experiment results showed that it was crucial to have consensus on the terminology used in the negotiation. It was difficult to figure out

Figure 3.5: Taxonomy

that A's *tool interoperability* was in fact B's *tool integration*. The term list provides a common ground for the many stakeholders to communicate. It also serves as a data dictionary for a project.

3.3.2 Taxonomy

In the bootstrap experiment, it was also realized that a domain taxonomy would help organize the domain and categorize artifacts. When the number of win conditions scaled up, it was particularly critical to break them into groups to localize candidate conflicts or consensus. In WinWin-1, the domain taxonomy was designed as a hierarchical structure and entered through a graphical user-interface. The links in the

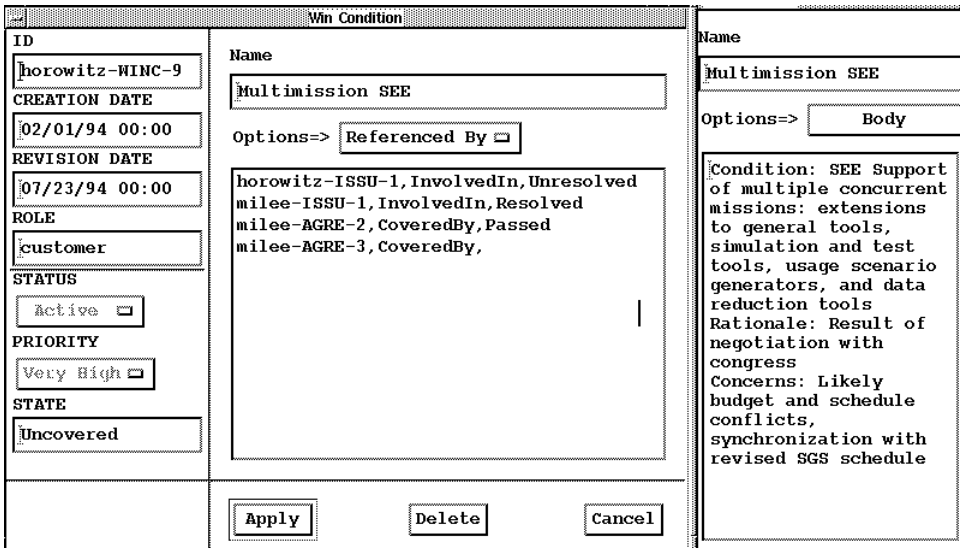


Figure 3.6: Win Condition

hierarchical structure were confusing since they represented many different meanings including IS-A, HAS-A, and INSTANCE-OF. In WinWin-95, it was re-designed to be a table of contents that was constructed via a list menu. Stakeholders could easily enter a taxonomy element as a chapter or a section in the table of contents and flexibly change the order and the hierarchy with the buttons provided.

3.3.3 Win Condition

A win condition is a stakeholder's objective, alternative, or constraint that he/she considers important and beneficial to the system. A win condition contains the following fields:

1. Name: name of this artifact

2. Role: owner of this artifact

3. ID: unique identifier generated by the WinWin system.

4. Creation Date: the date this data was created

5. Revision Date: last revision date

6. Status :

   - **active:** still under consideration
   - **inactive:** dropped

7. State :

   - **covered:** involved in only resolved issues and/or covered by only passed agreements
   - **uncovered:** otherwise

8. Priority: ranging from Very High, High, Medium, Low, Very Low to address the relative importance of a stakeholder's win conditions

9. Options: a screen area for entering and viewing various aspects of the artifact.

   (a) Body: a statement describing the nature of this artifact Stakeholders can switch from "body" to see other fields of this artifact

(b) Comment: used as a communication channel for the other stakeholders to express their opinion about this win condition

(c) Taxonomy Elements: identifiers relating the artifact to appropriate items in the taxonomy list

(d) Attachment: a file from any tool that helps explain, analyze, or specify this artifact such as COCOMO

(e) Refers_to: artifacts that it refers to through the *relates_to* and *replaces* relationships

(f) Referenced_by: the inverse relationship of "Refers_to" that allows traceability of any artifact that refers to this win condition; the following are references specific to win condition:

- **involved_in:** any issue that has an "involves" reference to it
- **covered_by:** any agreement that has a "covers" reference to it

Figure 3.6 demonstrates that the customer proposes a "Multimission SEE" win condition upon the request of the Congress.

3.3.4 Issue

The other artifacts (Issue, Option, Agreement) have the same 9-field structure as the Win Condition artifact. An issue is an aggregate that involves conflicting win conditions. In Figure 3.7, this issue is created because the previous win condition

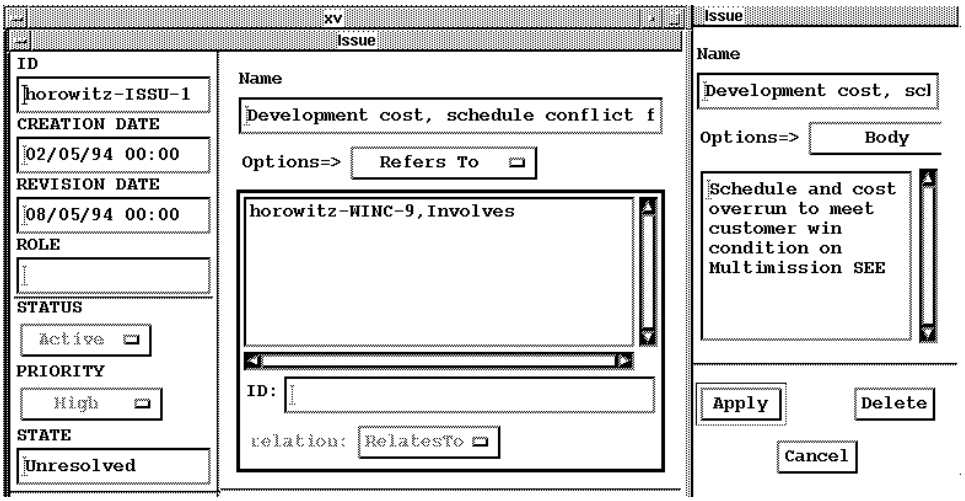


Figure 3.7: Issue

"Multimission SEE" was added and caused a schedule and cost overrun. Its fields differ from Win condition as follows:

1. State:
   - **resolved:** addressed by a used option
   - **unresolved:** otherwise
2. Refers_to: the following is specific to an issue
   - **involves:** reference to any win condition it involves
3. Referenced_by: the inverse relationship that allows traceability of any artifact that refers to this issue; the following are references specific to issue:
   - **addressed_by:** any option that has an "addresses" reference to it

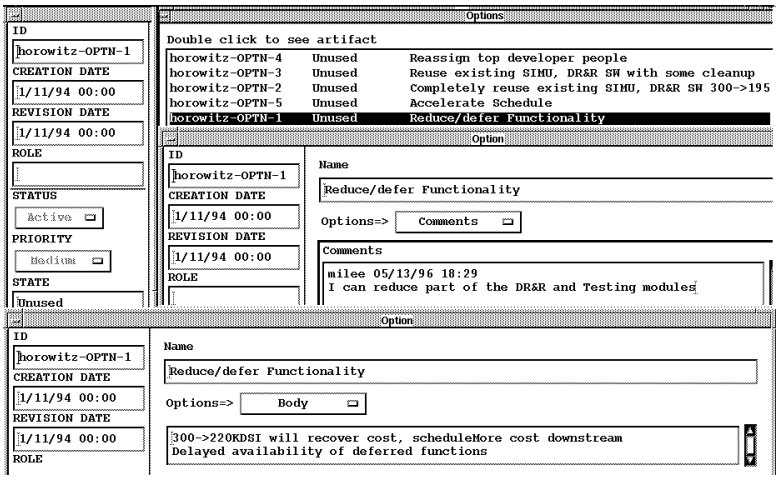


Figure 3.8: Option

3.3.5 Option

An option is an alternative that is proposed to resolve an issue. In this particular example, several possible options are provided as potential resolutions to the development cost and schedule overrun. The fields differ from Win condition as follows:

1. State:
   - **used:** addressed by a passed option
   - **unused:** otherwise
2. Refers_to: the following is specific to an option
   - **addresses:** pointer to the issue that it addresses

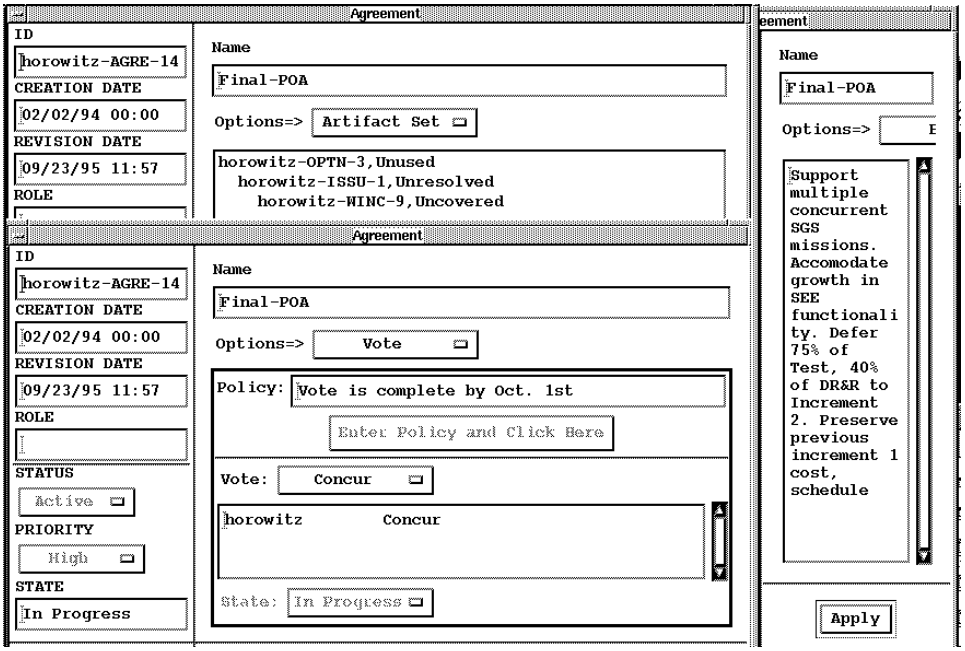


Figure 3.9: Agreement

3. Referenced_by:

   - **adopted_by:** any agreement that has an "adopts" reference to it

3.3.6 Agreement

Agreement is an aggregate that captures reconciled win conditions. An agreement is achieved either by directly covering a set of non-controversial win conditions or by adopting an option that resolves an issue in order to cover the win conditions involved in that issue. In this scenario, it adopts the "reduce/defer functionality" option. The user offered to defer 75% of the testing module and 40% of the DR&R

module till later to meet the cost and schedule constraints. Its fields differ from Win condition as follows:

1. State:
   - **-:** default value when it is created
   - **vote-in-progress:** when a vote is conducted on this agreement
   - **passed:** when the vote is passed
   - **failed:** when the vote is failed
2. Refers_to: the following is specific to an agreement
   - **adopts:** pointer to any option that it adopts
3. Vote: a supplementary artifact to conduct a vote on this agreement; after the owner of the agreement has entered a voting policy to officially start the vote, every stakeholder can express whether they "concur (pass)" to, "don't concur (fail)" to, or "abstain (neither pass nor fail)" from the vote. The owner changes the state of this agreement to "passed" or "failed" depending on the result of the vote.
4. Artifact_set: artifacts that this agreement directly or indirectly refers to; that is, all win conditions this agreement covers and all options this agreement adopts, all issues those options address, together with all win conditions those issues involve.

Part II

The Proposed Models

Part II presents the research results of this dissertation. It defines the requirements negotiation infrastructure and the requirements negotiation dynamics by constructing several formal and semi-formal models of the system and its operations. A big challenge here is to maintain consistency across the many views. Part II also defines the relationships among the various views in order to reconcile them to prevent inconsistency. Chapter 4 offers a problem space view that models the inter-win-condition relationships. Chapter 5 defines major WinWin artifacts and their inter-relationships. Chapter 6 defines how WinWin artifacts evolve in the negotiation process. Chapter 7 develops the WinWin equilibrium model to guide the WinWin users to recover the WinWin equilibrium state in the negotiation. Chapter 8 develops the framework of relationships among the various models or views in order to reconcile them. Chapter 9 discusses the implications of the research results in terms of improvements to the WinWin system. Chapter 10 summarizes the contributions made with this research.

Chapter 4

Inter-Win-Condition Relationship

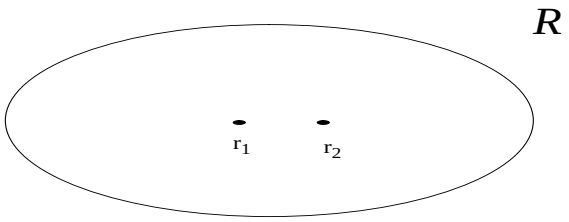


Figure 4.1: Requirements Space

Consider the space R of all requirements specifications r (see Figure 4.1). Each point r $\in$ R consists of a set of functional, performance, interface, and attribute specifications. For example, the only difference between r_1 and r_2 in Figure 4.1 may be that r_1's required response time is 1 second and r_2's is 2 seconds.

A Win Condition can then be defined as a constraint on R, dividing R into mutually exclusive subsets of requirements specifications which do or do not satisfy the Win Condition. Specifically, for stakeholder H_i, his/her j^{th} win condition $w_{i,j}$ defines the following subset of R (see Figure 4.2):

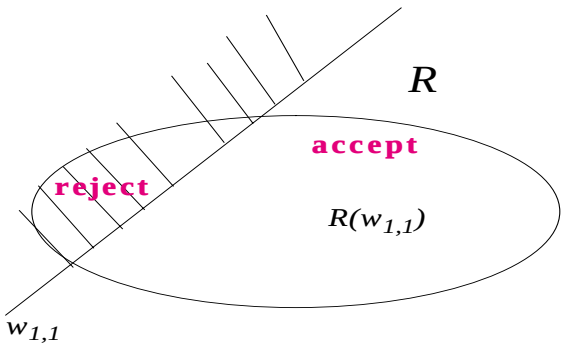


Figure 4.2: Requirements Space Divided by a Win Condition

$$R(w_{i,j}) = \{r : r\ satisfies\ w_{i,j}\}.$$

For example, if $w_{1,1}$ expressed stakeholder H_1's desire that the software cost be less than \$7 million, $R(w_{1,1})$ would be the set of all requirements specifications which could be implemented for less than \$7M.

If stakeholder H_1 has n_1 win conditions, his/her win region W_1 can be expressed as the intersection of all of the individual win condition regions (see Figure 4.3):

$$W_1 = \bigcap_{j=1}^{n_1} R(w_{1,j}).$$

By definition, any win conditions of stakeholder H_1 belongs to his/her win region W_1.

This framework can also be used to define the WinWin Issue, Option, and Agreement artifacts.

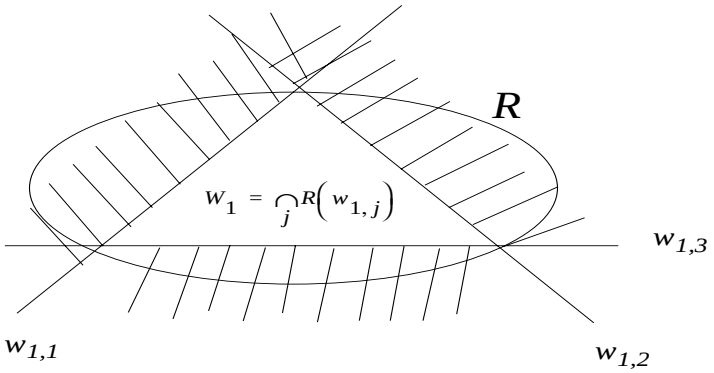


Figure 4.3: The Win Area for Stakeholder H_1

An agreement A_k covers a set of non-controversial win conditions whose win regions have a non-empty intersection.

Definition 4.1

$$\bigcap_{w_{i,j} \in A_k} R(w_{i,j}) \neq \phi.$$

An Issue I_k defines a set of win conditions whose win regions have an empty intersection.

Definition 4.2

$$\bigcap_{w_{i,j} \in I_k} R(w_{i,j}) = \phi.$$

It can be shown that the win conditions in I_k must fall in the area outside of some other stakeholder's win region W_i:

$$\forall w \in I_k, \exists W_i \; s.t. \; w \notin W_i.$$

An example would be that a customer H_1 has a win condition w_{11} that the software cost be less than \$7M. And the user H_2 has win conditions $\{w_{21}, w_{22}, w_{23}\}$ specifying functions {Simulation,Reporting,Testing} whose costs are {\$4M,\$3M,\$5M}, respectively. If the user wants all the functions to be implemented, it will cost \$12M. It is then identified by the customer as an issue containing $\{w_{11}, w_{21}, w_{22}, w_{23}\}$ since the total cost for implementing the user functions is beyond the customer's budget.

For an Issue I_k , an Option O_l proposes a relaxation $\{\triangle w_{i,j}\}_l$ to every win condition $w_{i,j}$ involved in that Issue. $\{\triangle w_{i,j}\}_l$ relaxes $w_{i,j}$ to $\{w'_{i,j}\}_l$ whose win region is enlarged and has a better chance to intersect with win regions of other win conditions involved in Issue I_k . $\triangle w_{i,j}$ is defined as the combination of all possible options applicable to $w_{i,j}$.

Define $\{w'_{i,j}\}$ as the relaxed form of $w_{i,j}$ after applying all possible options to enlarge the win region of $w_{i,j}$, whose win region is the union of the win regions of $w_{i,j}$ and $\triangle w_{i,j}$:

$$R(w'_{i,j}) = R(w_{i,j}) \cup \; R(\triangle w_{i,j})$$

The requirements space expanded for stakeholder H_1 by acceptable options is called the satisfactory region S_1:

$$\begin{aligned} S_1 &= \bigcap_{j=1}^{n_1} R(w'_{1,j}) - \bigcap_{j=1}^{n_1} R(w_{1,j}) \\ &= \bigcap_{j=1}^{n_1} R(w'_{1,j}) - W_1 \end{aligned}$$

The lose region L_1 of stakeholder H_1 is

$$L_1 = \mathcal{R} - (W_1 \cup S_1).$$

or

$$\begin{aligned} L_1 &= \mathcal{R} - (W_1 \cup (\bigcap_{j=1}^{n_1} R(w'_{1,j}) - W1) \\ &= \\ & \mathcal{R} - \bigcap_{j=1}^{n_1} R(w'_{1,j}) \end{aligned}$$

And it can be proved that an unresolvable Issue I_k must fall in at least one stakeholder's lose region:

$$\forall issue\ I_k, I_k\ is\ unresolvable \Leftrightarrow \exists L_m\ s.t.\ I_k \subset L_m.$$

With respect to the example, some candidate options would be to increase the budget limit by $\{\triangle w(1,1)\}_1 = \$1\text{M}$; to defer the simulation function $\{\triangle w(2,1)\}_2$; or various combinations of these.

An agreement A_k is then a resolution to resolve Issue I_k by adopting a feasible combination of options, which is passed by all the stakeholders.

Definition 4.3

$$\bigcap_{\{w'_{i,j}\}_l \in A_k} R(\{w'_{i,j}\}_l) \neq \phi.$$

Definition 4.1 is a special case of Definition 4.3, where no relaxation is needed and thus $\{w'_{i,j}\}_l = w_{i,j}$. It can be shown that an agreement is contained by the acceptable regions, namely, {SS, WS, WW, SW} in the 2-stakeholder case.

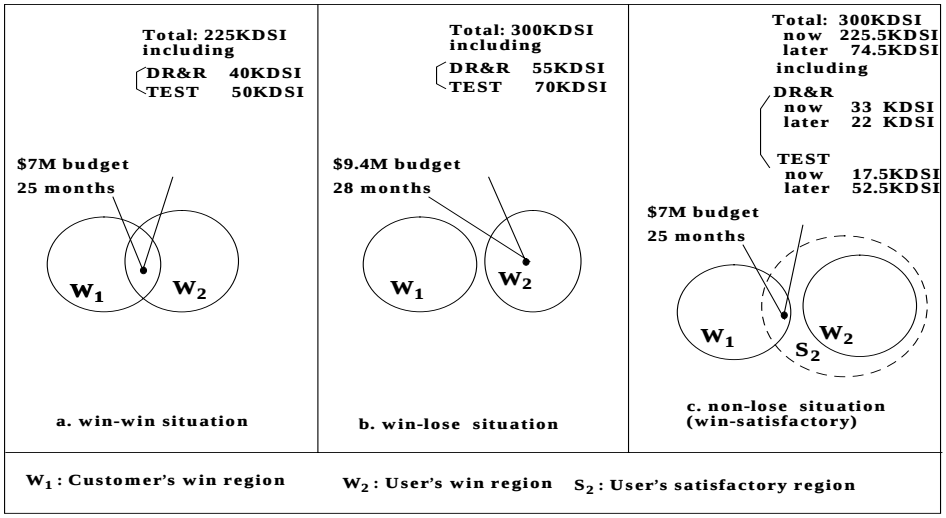


Figure 4.4: The inter-win-condition relationships (example)

The set definitions can be illustrated using set diagrams exemplified by Figure 4.4. W_1 and W_2 are Customer's and User's win regions, respectively. User at first presents win conditions for functionalities that require 225KDSI (thousand delivered source

instructions), including 40KDSI for Data Reduction and Reporting (DR&R) and 50KDSI for TEST. This can be done within $7M (million) budget and 25 months. As it is in Customer's win region, it is a win-win situation. Due to change of requirements, User is asking for more functionalities that increase the code size to 300KDSI. The budget and the schedule are thus raised beyond what Customer can supply. And now, it results in a win-lose situation.

One way to resolve this win-lose situation is to explore both stakeholders' satisfactory regions S_i's as options. In this example, User agrees to defer part of the DR&R and the TEST modules until more money becomes available. This implies that 225.5KDSI be implemented at this moment which again can be done within 25 months and $7M budget. It now settles in a non-lose (or win-satisfactory) situation.

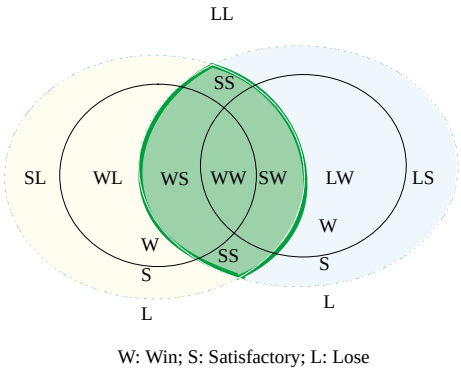


Figure 4.5: The inter-win-condition relationships

A complete enumeration of relationships between {Win,Lose,Satisfactory} regions of two stakeholders is illustrated in Figure 4.5. Figure 4.6 is a projection of

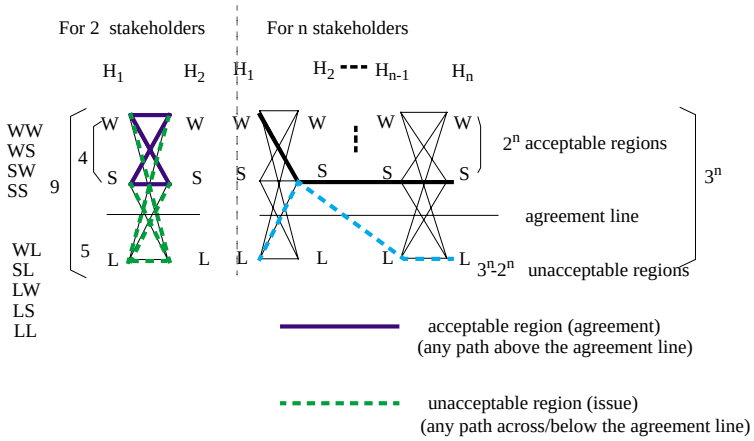


Figure 4.6: The generalized inter-win-condition relationships

Figure 4.5, generalizing the 2-stakeholder case into the n-stakeholder one. The total number of regions is 3^n as it is equivalent to the number of a 3-letter {W,S,L} permutations in n-digits where repeated letters are allowed. These regions are dichotomized into the following types:

- Acceptable region:

  This type of region contains requirements that do not fall in any stakeholder's lose region. By definition, it has a non-empty intersection among its requirements or win regions, and contains passed agreements. Any path above the agreement line satisfies this criterion as it contains only stakeholders' win or satisfactory regions. The number of qualified regions is 2^n as it is equivalent to the number of 2-letter {S,W} permutations in n-digit where repeated letters are allowed.

- Unacceptable region:

  This type of region contains requirements that do fall in some stakeholder's lose region and results in unresolvable issues. The number of such regions is $3^n(total) - 2^n(acceptable)$.

Chapter 5

The WinWin Artifact Types and Their Relationships

As described previously, WinWin provides four major types of artifacts to support the negotiation infrastructure:

Win Condition: a stakeholder's objective constraint, or alternative that (s)he considers important and beneficial.

Issue: an aggregate that involves a controversial combination of win conditions.

Option: an alternative that is proposed to resolve an issue.

Agreement: an aggregate that adopts options to resolve issues and cover reconciled win conditions.

In this chapter, set definitions will be given to these artifact types and the relationships between different types will be presented.

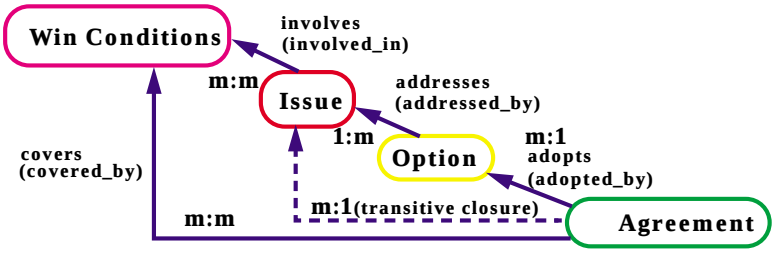


Figure 5.1: WinWin artifact relationships

5.1 Artifact Set and Relationship Definitions

The following are set definitions for the four major artifact types.

W: set of win conditions

I: set of issues

O: set of options

A: set of agreements

In chapter 4, the membership criteria for these sets are developed and analyzed. This chapter addresses the cardinality of relationships and chains of relationships among the artifact types.

The valid relationships between different WinWin artifact types are portrayed in Figure 5.1. An issue *involves* many conflicting win conditions. An option *addresses* an identified issue. An agreement *adopts* the best option to resolve the corresponding issue and *covers* all the win conditions involved in that issue.

These relationships between different artifact types are denoted by the following functions:

$$\begin{aligned} involves(x): \ & I \ \rightarrow \ W \\ addresses(x): \ & O \ \rightarrow \ I \\ adopts(x): \ & A \ \rightarrow \ O \\ covers(x): \ & A \ \rightarrow \ W \end{aligned}$$

Each function listed above has a corresponding inverse relationship[1]:

$$\begin{aligned} involved\_in(x): \ & W \ \rightarrow \ I \\ addressed\_by(x): \ & I \ \rightarrow \ O \\ adopted\_by(x): \ & O \ \rightarrow \ A \\ covered\_by(x): \ & W \ \rightarrow \ A \end{aligned}$$

5.2 Rules and Assumptions of Relationship

A problem concerning developing realistic model is that the real-world cases are hard to be exhaustively enumerated, and even if it can be done, the resulting model is usually very complicated and hard to understand. The strategy is thus to start from an easy-to-understand model that is a reasonable approximation of the real world with rules and assumptions. This section outlines the rules and assumptions

[1]These inverse relationships are automatically derived by the system from the relationships established by the users.

on inter-artifact relationships. Their limitations and possible extensions will also be discussed.

5.2.1 Existence Rule

WinWin helps stakeholders to negotiate requirements by acquiring their *win conditions* as the input, capturing conflicts between win conditions using *issues*, proposing alternatives to resolve issues using *options*, and composing *agreements* to adopt the best option and to cover reconciled win conditions. Given this process, the following rules assure that:

- an issue will not be created unless it involves some controversial win condition.
- an option will not be created unless it addresses some issue.
- an agreement will not be created unless either it covers some non-controversial win condition or it adopts some option which resolves an issue.

Rule 5.2.1 *For every issue, there must be at least one win condition that this issue involves.*

$$(\forall i \in I | \exists w \in W,\ s.t.\ w \in involves(i))$$

Rule 5.2.2 *For every option, there must be at least one issue that this option addresses.*

$$(\forall o \in O | \exists i \in I,\ s.t.\ i \in addresses(o))$$

Rule 5.2.3 *For every agreement, there must be at least one option that this agreement adopts or at least one win condition that the agreement covers.*

$$(\forall a \in A | (\exists o \in O,\ s.t.\ o \in adopts(a)) \vee (\exists w \in W,\ s.t.\ w \in covers(a)))$$

This first rule assumes that stakeholders will not find conflicts out of nothing (no win conditions). It prevents a stakeholder from recording vague, general concerns on risk, uncertainty or any breakdown regarding the system-to-build, which are not related to at least win condition. The second rule assumes that an option must be connected with some issue. It eliminates the possibility that stakeholders may want to propose options that are not related to any current issue and store them in the repository for later use. (If they feel the option is sufficiently necessary, they can enter it as a win condition). The third rule insists that an agreement has to cover some win condition. Even if there is already consensus among stakeholders on creating an agreement, they have to compose a win condition before they create that agreement. These rules were not implemented in the initial version of WinWin-95.

They were considered worth adding after several users caused the system com confusion by entering untraceable issues, options, and agreements. The formal modeling effort here was extended to define and analyze appropriate rules for the system.

5.2.2 Cardinality Rule

The following rules define the cardinalities on inter-artifact relationships as illustrated in Figure 5.1. The principle here is for each issue to allow as many options to address it but only one agreement to resolve it. Only rules which set constraints on the cardinalities of the relationships will be given predicate calculus definitions.

Rule 5.2.4 *An issue can involve many win conditions.*

Rule 5.2.5 *An option can address only one issue.*

$$(\forall o \in O| \quad (\forall i_1, i_2 \in I, \\ (((i_1 \in addresses(o)) \wedge (i_2 \in addresses(o))) \rightarrow (i_1 = i_2))))$$

Rule 5.2.6 *An agreement can only adopt options that are not adopted by any other agreements.*

$$(\forall a \in A| \quad \forall o \in O, \\ ((o \in adopts(a)) \wedge (\exists a_1 \in A \; s.t. \; (o \in adopts(a_1)))) \\ \rightarrow (a_1 = a))$$

For the following rule, a supplementary relationship "resolves" between an agreement and an issue is defined as the transitive closure of "adopts" and "addresses." An agreement a resolves an issue i if a adopts an option that addresses i.

$$\forall a \in A, resolves(a) = \{i | i \in I \wedge (\exists o \in O \; s.t. \; (o \in adopts(a) \wedge i \in addresses(o)))\}$$

Rule 5.2.7 *An issue can be resolved by only one agreement*[2].

$$(\forall a \in A | \quad \forall i \in resolves(a), \\ ((\exists a_1 \in A \; s.t. \; (i \in resolves(a_1))) \rightarrow (a_1 = a))$$

Rule 5.2.8 *an agreement can cover many non-controversial win conditions*

The only rules setting constraints on the cardinalities of the inter-artifact relationships are Rules 5.2.5, 5.2.6 and 5.2.7. These constraints are added to simplify the artifact life cycle model that will be discussed later. In short, the life cycle of an issue is determined by the states of its addressing options. The states of these options again are determined by the states of their adopting agreements. If more than one agreement is allowed to adopt the same option or to potentially resolve the same issue, how to determine the state of that option or that issue will involve expensive computation in enumerating the many possible state combinations of the

[2] Without defining "resolves," this rule would result in a longer version: an agreement can adopt options that address different issues as long as these issues are not addressed by options that are adopted by other agreements.

many agreements. The limitation for Rule 5.2.5 is that if a stakeholder wants to reuse an option to address another issue, he/she will have to duplicate that option and versions of the same option may create inconsistency. It also generates many overlapping agreements if these agreements adopt duplicated options. Rules 5.2.6 and 5.2.7 deter stakeholders from combining agreements to resolve the same issue. The idea is to push forward the combination of solutions to the option level. If he/she wants to refine the resolution to an issue, he/she must replace the old agreement with a new one that consolidates the added options with options in the old agreement.

5.2.3 Artifact Dropping Rule

One of the goals for WinWin is to deal with changes to requirements. A very common change to requirements is deletion/dropping. This section contains rules for dropping (i.e. inactivating/deleting)[3] an artifact.

An agreement can be dropped at any time. If it is dropped, all artifacts referenced by it directly (like the options adopted by it) or indirectly (like the issues that are addressed by its adopted options) will be reset to the states as they were before that agreement is proposed.

[3] "inactivate" means this dropped artifact still can be viewed, whereas "delete" means this artifact is dropped from the data base permanently

An option can only be dropped if there is no agreement adopting it. If there is an agreement adopting it, stakeholders must first remove the "adopts" reference from that agreement. If the dropped option is the only option that the agreement adopts, the agreement ought to be dropped first since every agreement must point to at least one option. If the adopting agreement has a vote in progress or is already passed, no references can be removed until the vote fails or the agreement gets dropped. Once an option is dropped, all artifacts referenced by it directly or indirectly restore their states back to what they were before that option is proposed.

An issue can only be dropped if there is no option addressing it. Otherwise, the references must be removed from its addressing options. As each option can only address one issue and an option cannot exist if it does not point to any issue, this implies that all the options addressing this issue must be inactivated or redirected to other issues before this issue can be dropped.

A win condition can be dropped if

- it is not the only one covered by an agreement.

  Otherwise, the agreement needs to be dropped first.

- it is not the only one involved in an issue.

  Otherwise, the issue needs to be dropped first.

- it is not (directly or indirectly) covered by an agreement that has a vote in progress.

  Otherwise, it requires the vote to fail or the agreement to be dropped first.

- it is not (directly or indirectly) covered by an agreement whose vote has passed.

  Otherwise, the agreement needs to be dropped first.

5.3 Artifact Chain and Artifact Set

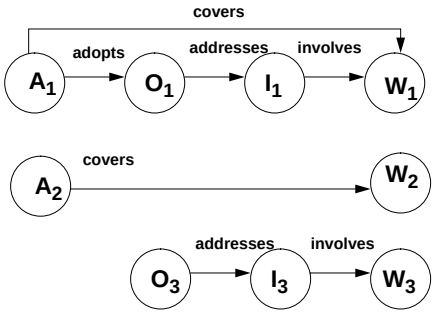


Figure 5.2: Examples of artifact chain

Given the artifact types and relationships, the artifact chains and artifact sets can be composed to portray graphs that capture possible connections between artifact nodes. An artifact chain is a graph of artifact nodes connected by valid relationship links. An artifact set of a win condition w is the union of all artifact chains starting with w. The following sections define and illustrate artifact chains and artifact sets.

5.3.1 Artifact Chain

An artifact chain is a graph of artifact nodes connected by valid relationship links. The valid relationship links are defined according to the previous section:

Definition 5.3.1

$$\stackrel{involves}{\longrightarrow}: \quad involves$$

$$\stackrel{addresses}{\longrightarrow}: \quad addresses$$

$$\stackrel{adopts}{\longrightarrow}: \quad adopts$$

$$\stackrel{covers}{\longrightarrow}: \quad covers$$

Define an artifact chain to be:

Definition 5.3.2

$$\begin{aligned}
chain \quad & ::= [I-chain | A_c - chain] < Win\ Condition > \\
I-chain \quad & ::= [O-chain] < Issue > \stackrel{involves}{\longrightarrow} \\
O-chain \quad & ::= [A_a - chain] < Option > \stackrel{addresses}{\longrightarrow} \\
A_a - chain \quad & ::= < Agreement > \stackrel{adopts}{\longrightarrow} \\
A_c - chain \quad & ::= < Agreement > \stackrel{covers}{\longrightarrow}
\end{aligned}$$

In an artifact chain, it is important to know what nodes are participating in that chain. The following definitions are formulated to help project different artifact nodes of an artifact chain.

Definition 5.3.3 *Define H to be the set of all chains.*

Definition 5.3.4 *Define SH^I to be the set of all I-chains.*

Definition 5.3.5 *Define SH^O to be the set of all O-chains*

Definition 5.3.6 *Define SH^A_a to be the set of all A_a-chains*

Definition 5.3.7 *Define SH^A_c to be the set of all A_c-chains*

The function "head" returns the starting win condition w of an artifact chain h.

Definition 5.3.8

$$\begin{aligned} &\exists w \in W, \qquad \exists h \in H, \\ &head(h) = \quad w \Leftrightarrow \qquad (h = w) \vee \\ &\qquad\qquad\qquad\qquad (\exists h_i \in SH^I \; s.t. \; h = h_i \; w) \vee \\ &\qquad\qquad\qquad\qquad (\exists h_a \in SH^A_c \; s.t. \; h = h_a \; w) \end{aligned}$$

The function "involver" returns the involving issue i of an artifact chain h.

Definition 5.3.9

$$\begin{aligned} &\exists i \in I, \;\; \exists h \in H, \\ &involver(h) = \qquad i \Leftrightarrow \\ &\qquad\qquad\qquad (\exists w \in W \; s.t. \; h = i \stackrel{involves}{\longrightarrow} w) \vee \\ &\qquad\qquad\qquad (\exists w \in W \wedge \exists h_o \in SH^O \; s.t. \; h = h_o i \stackrel{involves}{\longrightarrow} w) \end{aligned}$$

The function "addresser" returns the addressing option o of an artifact chain h.

Definition 5.3.10

$$\exists o \in O, \ \exists h \in H,$$

$$addresser(h) = \quad o \Leftrightarrow$$

$$(\exists w \in W, \exists i \in I \ s.t. \ h = o \overset{addresses}{\longrightarrow} i \overset{involves}{\longrightarrow} w) \vee$$

$$(\exists w \in W, \exists i \in I, \exists h_a \in SH_c^A \ s.t. \ h = h_a o \overset{addresses}{\longrightarrow} i \overset{involves}{\longrightarrow} w)$$

The function "coverer" returns the covering agreement a of an artifact chain h.

Definition 5.3.11

$$\exists a \in A, \ \exists h \in H,$$

$$coverer(h) = \quad a \Leftrightarrow$$

$$(\exists w \in W, \exists i \in I, \exists o \in O \ s.t. \ h = a \overset{adopts}{\longrightarrow} o \overset{addresses}{\longrightarrow} i \overset{involves}{\longrightarrow} w) \vee$$

$$(\exists w \in W \ s.t. \ h = a \overset{covers}{\longrightarrow} w)$$

Any chain containing an issue indicates there is conflict that requires resolution. H^I is defined to mark this kind of chain.

Definition 5.3.12

$$H^I = \{h | h \in H, \exists w \in W, \exists h_i \in SH^I \ s.t. \ h = h_i \ w\}$$

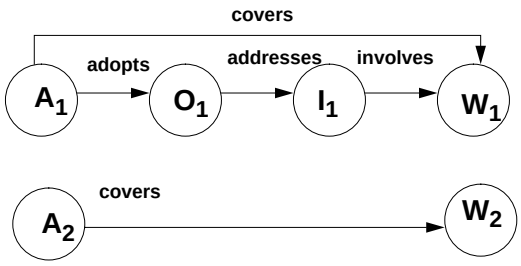


Figure 5.3: Examples of complete artifact chains

The following definitions define what is a complete artifact chain and what is a degraded artifact chain.

Definition 5.3.13 *A complete artifact chain is an artifact chain starting with a win condition and ending with an agreement.*

$$\exists h \in H,$$

$$h \; is \qquad complete \;\Leftrightarrow$$

$$\exists w \in W, \exists i \in I, \exists o \in O, \exists a \in A, \; s.t.$$

$$(h = a \overset{covers}{\longrightarrow} w) \vee (h = a \overset{adopts}{\longrightarrow} o \overset{addresses}{\longrightarrow} i \overset{involves}{\longrightarrow} w)$$

Definition 5.3.14 *A degraded chain is a chain containing only a win condition.*

$$\exists h \in H,$$

$$h \; is \qquad degraded \;\Leftrightarrow \exists w \in W, \; s.t. \; h = w$$

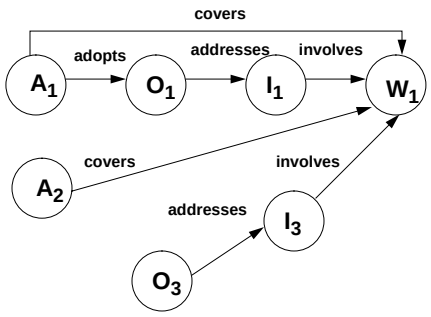


Figure 5.4: An artifact set example

5.3.2 Artifact Set

Definition 5.3.15 *An artifact set of a win condition is the union of all artifact chains starting with this win condition.*

$$\exists w \in W,$$

$$Artifact\_set(w) = \{h | h \in H, \exists h_s \in (SH_c^A \cup SH^I)\ s.t.\ (h = h_s\ w)\}$$

Chapter 6

The WinWin Artifact Life Cycle

The WinWin Artifact Life Cycle is defined by two attributes of an artifact: *status* and *state*. The attribute "status"is to show whether an artifact is "active" or "inactive." A functional definition for the attribute "status" is as follows:

Defining the domain and range of the function status(x):

Definition 6.1 *status(x):* $(W \cup I \cup O \cup A) \rightarrow \{active, inactive\}$

The attribute "state," on the one hand, reflects the inter-relationship between artifacts. On the other hand, it helps define the state in the equilibrium model that tells the stakeholders how far away from equilibrium it is and what potential operations there are to approach a desired state in the equilibrium model. In this chapter, a detailed State model will be elaborated to portray how it supports the two purposes mentioned above.

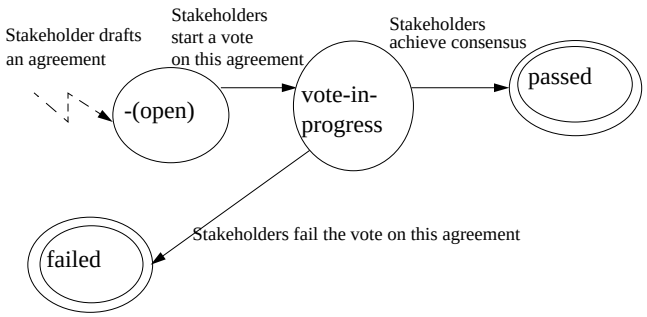


Figure 6.1: Agreement life cycle

6.1 Agreement

For any agreement a, it is associated with a state determined by the progress of the voting process as follows:

- **open:** when a is proposed.
- **vote-in-progress:** when a is being voted.
- **passed:** when stakeholders pass the vote.
- **failed:** when stakeholders fail the vote.

These states are exhaustive and mutually exclusive.

The functional definition for the attribute "state" of an agreement is as follows:

Definition 6.2 $state(a): A \rightarrow \{open, vote-in-progress, passed, failed\}$

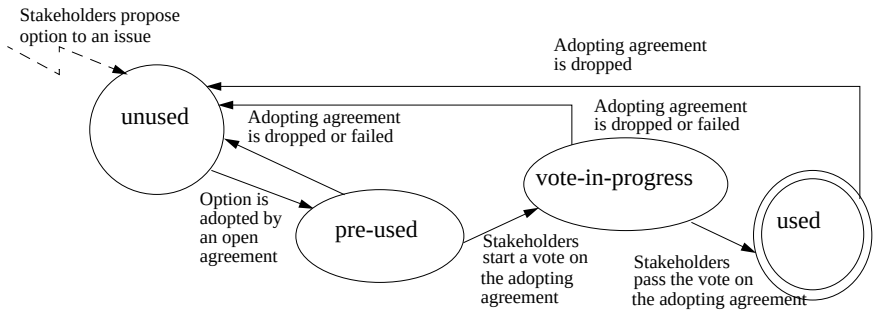


Figure 6.2: Option life cycle

6.2 Option

For any option o, it is associated with a state determined by its adopting agreement as follows:

- **unused**
  - **English description:** o is not adopted by any agreement.
  - **Description in predicate calculus:**

    $$(\forall(a \in A)|(o \notin adopts(a)) \vee (state(a) = failed))$$

- **pre-used**
  - **English description:** o is adopted by an open agreement.
  - **Description in predicate calculus:**

    $$(\exists(a \in A)|(o \in adopts(a)) \wedge (state(a) = open))$$

- **vote-in-progress**
  - **English description:** o is adopted by a vote-in-progress agreement.
  - **Description in predicate calculus:**

$$(\exists(a \in A)|(o \in adopts(a)) \wedge (state(a) = vote - in - progress))$$

- **used**
  - **English description:** o is adopted by a passed agreement.
  - **Description in predicate calculus:**

$$(\exists(a \in A)|(o \in adopts(a)) \wedge (state(a) = passed))$$

The functional definition for the attribute "state" of an option is as follows:

Definition 6.3 $state(o): O \rightarrow \{unused, pre - used, vote - in - progress, used\}$

6.3 Issue

For any issue i, it is associated with a state determined by its addressing options as follows:

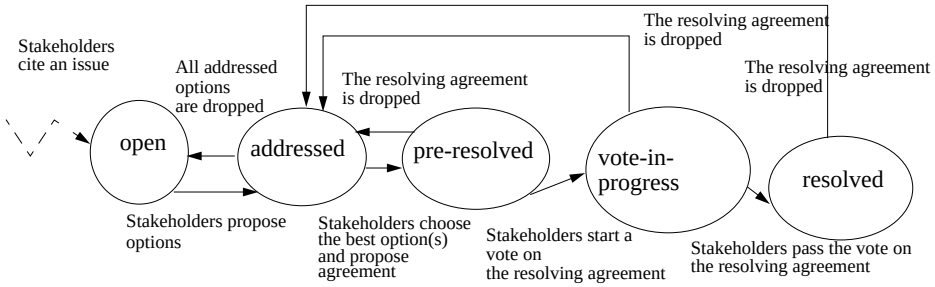


Figure 6.3: Issue life cycle

- **open**
  - **English description:** i is not addressed by any options.
  - **Description in predicate calculus:**

    $(\forall(o \in O)|(i \notin addresses(a)))$

- **addressed**
  - **English description:** i is addressed by only unused options.
  - **Description in predicate calculus:**

    $(\exists(o \in O)|(i \in addresses(a)) \wedge (state(o) = unused))$

- **pre-resolved**
  - **English description:** i is addressed by a pre-used option.
  - **Description in predicate calculus:**

    $(\exists(o \in O)|(i \in addresses(a)) \wedge (state(o) = pre - used))$

- **vote-in-progress**
  - **English description:** i is addressed by a vote-in-progress option.
  - **Description in predicate calculus:**

    $$(\exists(o \in O)|(i \in addresses(a)) \wedge (state(o) = vote - in - progress))$$

- **resolved**
  - **English description:** i is addressed by a used option.
  - **Description in predicate calculus:**

    $$(\exists(o \in O)|(i \in addresses(a)) \wedge (state(o) = used))$$

The functional definition for the attribute "state" of an issue is as follows:

Definition 6.3.1

$$state(i): \ I \rightarrow \{open, addressed, pre - resolved, vote - in - progress, resolved\}$$

6.4 Win Condition

A win condition can be referred to by both an issue (via the relationship "involves"), and an agreement (via the relationship "covers") to result in various negotiation states.

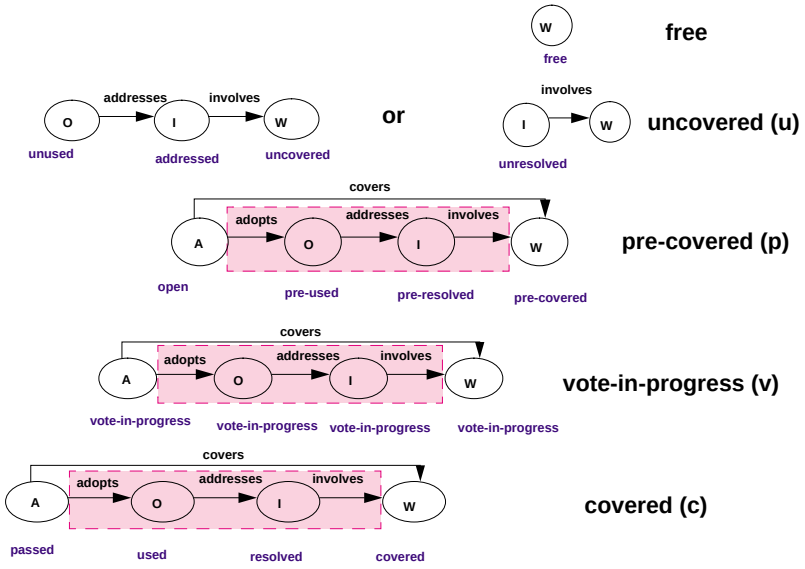


Figure 6.4: Basic states in a particular artifact chain

6.4.1 Basic States

The basic state of a win condition is determined by either the issue that involves this win condition and or the agreement that covers it within a particular artifact chain as shown in Figure 6.4. The grey shading indicates that an agreement can also indirectly cover a win condition by adopting an option that addresses an issue involving that win condition. Table 6.1 provides a view that dichotomizes Figure 6.4 into artifact chains including or excluding issues. Figures 6.5 and 6.6 show the life cycle of a win condition within a chain including an issue or excluding an issue respectively. Figure 6.7 provides a merged view of the previous two figures to enumerate

Artifact Type	Win Condtion	Issue	Option	Agreement
State (for an artifact chain with an issue)	**free**	-	-	-
	uncovered	**open**	-	-
	uncovered	**addressed**	**unused**	-
	pre-covered	**pre-resolved**	**pre-used**	**open**
	vote-in-progress	**vote-in-progress**	**vote-in-progress**	**vote-in-progress**
	covered	**resolved**	**used**	**passed**
	uncovered	**addressed**	**unused**	**failed**
State (for an artifact chain without an issue)	**pre-covered**	-	-	**open**
	voted	-	-	**in-progress**
	covered	-	-	**passed**
	free	-	-	**failed**

Table 6.1: State correspondence in an artifact chain

all the possible basic states and transitions for the life cycle of a win condition in the context of a specific artifact chain.

The following give definitions to the basic states for a win condition w as illustrated in Figure 6.7.

- **free**
  - **English description:** w is neither involved in any issues nor covered by any agreements.
  - **Description in predicate calculus:**

    $(\forall(i \in I),\ \forall(a \in A)|(w \notin involves(i)) \wedge (w \notin covers(a)\}))$

- **uncovered**
  - **English description:** w is involved in an open/addressed issue.

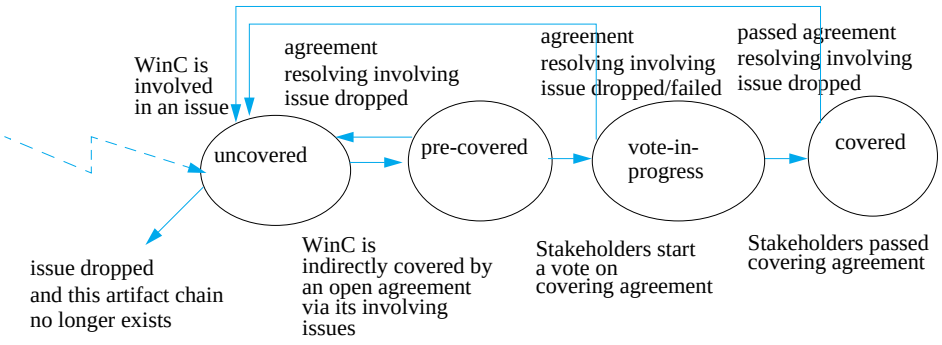


Figure 6.5: Win condition life cycle: within an artifact chain starting with a win condition involved in an issue

  - **Description in predicate calculus:**

$$(\exists(i \in I)|(w \in involves(i)) \wedge (state(i) \in \{open, addressed\}))$$

- **pre-covered**

  - **English description:** w is covered by an open agreement.

  - **Description in predicate calculus:**

$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = open))$$

- **vote-in-progress**

  - **English description:** w is covered by a vote-in-progress agreement.

  - **Description in predicate calculus:**

$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = vote - in - progress))$$

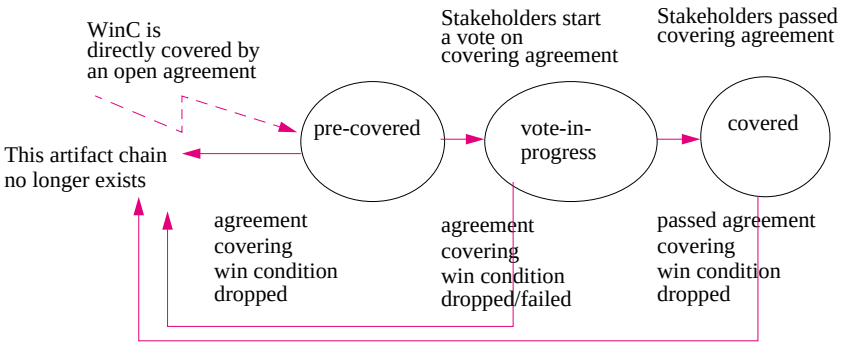


Figure 6.6: Win condition life cycle: within an artifact chain starting with win condition covered by an agreement

- **covered**

  - **English description:** w is covered by a passed agreement.

  - **Description in predicate calculus:**

    $$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = passed))$$

The functional definition for the attribute "basic-state" of a win condition is as follows:

Definition 6.4

$$basic\_state(w): W \rightarrow \{free, uncovered, vote-in-progress, covered\}$$

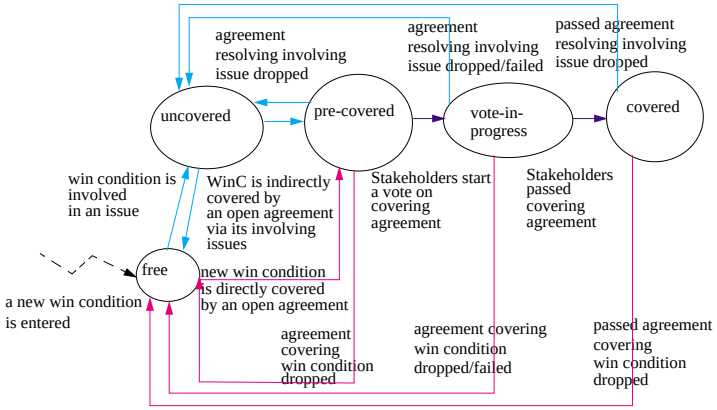


Figure 6.7: Win condition life cycle: merged view

The following notation is used to project the basic state for a win condition w in a particular artifact chain h.

basic_state(w):h

6.4.2 State Transition Operators

As described in 5.3.2, a win condition w has an artifact set that is composed of all artifact chains starting with w. While each artifact chain can have only one basic state—uncovered(U), pre-covered(P), vote-in-progress (V), and covered(C), the many artifact chains can result in many possible combinations of the basic states. Figure 6.8 shows an example of the UPV combination—a win condition that has three basic states: uncovered(U), pre-covered(P), and vote-in-progress(V) in the three artifact chains originating from it.

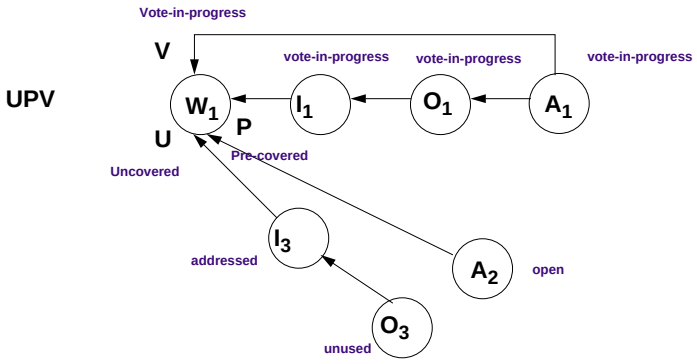


Figure 6.8: The UPV state

The following set describes this augmented state space.

Definition 6.4.1

$$S = \mathbf{L}((U+\lambda)(P+\lambda)(V+\lambda)(C+\lambda))$$
$$= \{\lambda, U, P, V, C, UP, UV, UC, PV, PC, VC, UPV, UPC, UVC, PVC, UPVC\}$$

These states are classified into 6 super states as shown in Table 6.2. This classification characterizes:

1. **Free:** w is neither involved in an issue nor covered by an agreement.

2. **Bound:** w is involved in some open/addressed issue(s) and/or covered only by some open agreement(s).

3. **Bound-Frozen:** w is covered by no passed but at least one vote-in-progress agreement that causes its content to be frozen for that vote.

Augmented State	uncovered(u)	pre-covered(p)	voted(v)	covered(c)
free	0	0	0	0
Uncovered (U)	1	0	0	0
Pre-covered(P)	0	1	0	0
uncovered &pre-covered(up)	1	1	0	0
Voted(V)	0	0	1	0
uncovered&voted (uv)	1	0	1	0
Pre-covered&voted(pv)	0	1	1	0
uncovered &pre-covered&voted(upv)	1	1	1	0
uncovered &covered(uc)	1	0	0	1
pre-resolved& covered(pc)	0	1	0	1
uncovered &pre-covered& covered(upc)	1	1	0	1
voted&covered(vc)	0	0	1	1
uncovered &voted&covered(uvc)	1	0	1	1
pre-resolved&voted&covered(pvc)	0	1	1	1
uncovered &pre-covered&voted&covered(upvc)	1	1	1	1
Covered(C)	0	0	0	1

Table 6.2: Win condition life cycle: hierarchical view

4. **Partially-Covered:** w is covered by some agreement(s); and among the agreements that cover it, at least one (but not all) is passed and there is no vote-in-progress agreement.

5. **Partially-Covered-Frozen:** w is covered by at least one passed agreement and one vote-in-progress agreement.

6. **Fully-Covered:** w is only covered by passed agreements.

Definition 6.4.2

$$S_a = \{ \; free,\ bound,\ bound-frozen,\ partially-covered,$$
$$partially-covered-frozen,\ fully-covered \; \}$$

Define the function *sub* for the states in S_a as follows:

Definition 6.4.3

$$
\begin{aligned}
sub(free) &= \{\lambda\} \\
sub(bound) &= \{P, U, PU\} \\
sub(bound - frozen) &= \{V, PV, UV, PUV\} \\
sub(partially - covered) &= \{UC, PC, PUC\} \\
sub(partially - covered - frozen) &= \{VC, UVC, PVC, PUVC\} \\
sub(covered) &= \{C\}
\end{aligned}
$$

Definition 6.4.4 $\forall s \in S,\ super(s) = \{x | x \in S_a \wedge s \in sub(x)\}$

States in S will be called the *sub* states. States in S_a will be called the *super* states. The sub states together with the super states formulate a hierarchical state model that will be explained in details in section 6.4.4.

The following operators are possible transitions among these many possible states in the lower level.

6.4.2.1 Basic Operators

+U: The win condition is involved in a new issue (of state "open/addressed").

+P: The win condition is covered by an open agreement.

!U: One of the involving "open/addressed" issues is dropped.

-U: The only involving "open/addressed" issue is dropped.

!P: One of the covering open agreements is dropped or failed.

-P: The only covering open agreement is dropped or failed.

!V: One of the covering vote-in-progress agreements is dropped or failed.

-V: The only covering vote-in-progress agreement is dropped or failed.

!C: One of the covering passed agreements is dropped.

-C: The only covering passed agreement is dropped.

!U+P: An open agreement is proposed to resolve some (but not all) issues that involve the win condition.

-U+P: An open agreement is proposed to resolve all issues that involve the win condition.

!P+V: A vote is conducted on one of the open agreements that cover the win condition.

-P+V: A vote is conducted on the only open agreement that covers the win condition.

!V+C: The vote on one of the vote-in-progress agreements that cover the win condition is passed.

-V+C: The vote on the only vote-in-progress agreement that covers the win condition is passed.

!P+U: Dropping one of the open agreements that indirectly cover this win condition by issues causes the corresponding issues to back off from "pre-resolved" to "addressed".

-P+U: Dropping the only open agreement that covers this win condition by issues causes the corresponding issues to back off from "pre-resolved" to "addressed".

!V+U: Dropping one of the vote-in-progress agreements that cover this win condition by issues causes the corresponding issues to back off from "vote-in-progress to "addressed".

-V+U: Dropping the only vote-in-progress agreement that covers this win condition by issues causes the corresponding issues to back off from "vote-in-progress" to "addressed".

!C+U: Dropping one of the passed agreements that indirectly cover this win condition by issues causes the corresponding issues to back off from "resolved" to "addressed".

-C+U: Dropping the only passed agreement that indirectly covers this win condition by issues causes the corresponding issues to back off from "resolved" to "addressed".

Define a set T (Transition Operator) to be the set of the above operators.

Definition 6.4.5

$$\begin{aligned} T = \{ \; & +U,\ +P,\ -U,\ !U,\ -P,\ !P,\ -V,\ !V,\ -C,\ !C, \\ & -U+P,\ !U+P,\ -P+V,\ !P+V,\ -V+C,\ !V+C, \\ & -P+U,\ !P+U,\ -V+U,\ !V+U,\ -C+U,\ !C+U \ \} \end{aligned}$$

6.4.2.2 Rules

1. **+U** can be applied at any state. That is, a win condition can be involved in an open/addressed issue at any state.

2. **+P** can be applied at any state. That is, a win condition can be covered by an open agreement at any state.

3. **!U**, **-U**, **!U+P**, and **-U+P** can only be applied when the current state contains a U.

4. **!P**, **-P**, **!P+V**, **-P+V**, **!P+U**, and **-P+U** can only be applied when the current state contains a P.

5. **!V**, **-V**, **!V+C**, **-V+C**, **!V+U**, and **-V+U** can only be applied when the current state contains a V.

6. **!C**, **-C**, **!C+U**, and **-C+U** can only be applied when the current state contains a C.

7. **!P+U**, **-P+U**, **!V+U**, **-V+U**, **!C+U**, and **-C+U** can only be applied when the corresponding artifact chain has an issue.

8. Operator **!X** can only be applied when there are multiple X's; where X can be U, P, V or C.

6.4.2.3 State Set Definitions

Definition 6.4.6 $L^U = (U + \lambda)$

Definition 6.4.7 $L^P = (P + \lambda)$

Definition 6.4.8 $L^V = (V + \lambda)$

Definition 6.4.9 $L^C = (C + \lambda)$

Definition 6.4.10 $S^U = L(U L^P L^V L^C) \subseteq S$

Definition 6.4.11 $\overline{S^U} = S - S^U$

Definition 6.4.12 $S^P = L(L^U P L^V L^C) \subseteq S$

Definition 6.4.13 $\overline{S^P} = S - S^P$

Definition 6.4.14 $S^V = L(L^U L^P V L^C) \subseteq S$

Definition 6.4.15 $\overline{S^V} = S - S^V$

Definition 6.4.16 $S^C = L(L^U L^P L^V C) \subseteq S$

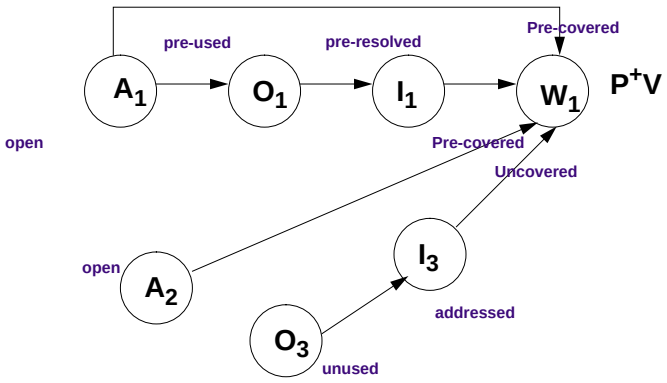


Figure 6.9: An example of an artifact chain in the set M^P

Definition 6.4.17 $\overline{S^C} = S - S^C$

The following sets define the situations when a win condition has the same basic state in two of the artifact chains that it originates as exemplified by figure 6.9.

Definition 6.4.18 $M^U = \{s | s \in S^U, \exists w \in W, \exists h_1, h_2 \in H,\ s.t.\ (head(h_1) = w) \wedge (head(h_2) = w) \wedge (state(w) : h_1 = uncovered) \wedge (state(w) : h_2 = uncovered)\}$

Definition 6.4.19 $M^P = \{s | s \in S^P, \exists w \in W, \exists h_1, h_2 \in H,\ s.t.\ (head(h_1) = w) \wedge (head(h_2) = w) \wedge (state(w) : h_1 = pre-covered) \wedge (state(w) : h_2 = pre-covered)\}$

Definition 6.4.20 $M^V = \{s | s \in S^V, \exists w \in W, \exists h_1, h_2 \in H,\ s.t.\ (head(h_1) = w) \wedge (head(h_2) = w) \wedge (state(w) : h_1 = vote-in-progress) \wedge (state(w) : h_2 = vote-in-progress)\}$

Definition 6.4.21 $M^C = \{s | s \in S^C, \exists w \in W, \exists h_1, h_2 \in H,\ s.t.\ (head(h_1) = w) \wedge (head(h_2) = w) \wedge (state(w) : h_1 = passed) \wedge (state(w) : h_2 = passed)\}$

Definition 6.4.22 $\overline{M^U} = S^U - M^U$

Definition 6.4.23 $\overline{M^P} = S^P - M^P$

Definition 6.4.24 $\overline{M^V} = S^V - M^V$

Definition 6.4.25 $\overline{M^C} = S^C - M^C$

6.4.2.4 Axioms

In this section, the pre-condition and post-condition of each operator defined in 6.4.2.1 will be given. That is, for each operator, the axiom will show in what state an operator is applicable, and after the transition operator has been applied, how the next state is determined.

Definition 6.4.26 $\forall x \in X$, *a transition operator* $t \in T$ *is applicable* $\Leftrightarrow t(x) \in S$

1. **+U**

   - **pre-condition:** **+U** can be applied at any state. That is, a win condition can be involved in an open/addressed issue at any state.

$$+\mathbf{U}(x) \in S \Leftrightarrow x \in S$$

Axiom 6.4.1

$$+\mathbf{U}(x) = \begin{cases} x & \forall x \in S^U \\ Ux & \forall x \in \overline{S^U} \end{cases}$$

2. **+P**

- **pre-condition:** **+P** can be applied at any state. That is, a win condition can be covered by an open agreement at any state.

$$+\mathbf{P}(x) \in S \Leftrightarrow x \in S$$

Axiom 6.4.2

$$+\mathbf{P}(x) = \begin{cases} x & \forall x \in S^P \\ x_1 P x_2 & \forall x_1 \in L(L^U)\ \forall x_2 \in L(L^V L^C)\ s.t.\ x = x_1 x_2 \in \overline{S^P} \end{cases}$$

3. **!U**

- **pre-condition:** **!U** can be only be applied when in the current state, there are multiple open/addressed issues (U) involving this win condition. And when one involving open/addressed issue is dropped, there is still another open/addressed issue to keep U.

$$x \in M^U \Leftrightarrow \mathbf{!U}(x) \in S$$

Axiom 6.4.3

$$!\mathbf{U}(x) = x, \quad \forall x \in M^U$$

4. **-U**

- **pre-condition:** **-U** can be only be applied when the current state contains only one U. That is, there is only one open/addressed issue (U) involving this win condition. When the only open/addressed issue is dropped, U is therefore eliminated.

$$x \in \overline{M^U} \Leftrightarrow -\mathbf{U}(x) \in S$$

Axiom 6.4.4

$$-\mathbf{U}(x) = x_1 \quad \forall x = Ux_1 \; s.t. \; x \in \overline{M^U}$$

5. **!P**

- **pre-condition:** **!P** can be only be applied when in the current state, there are multiple open agreements (P) covering this win condition. And when one open agreement is dropped, there is still another open agreement to keep P.

$$x \in M^P \Leftrightarrow !\mathbf{P}(x) \in S$$

Axiom 6.4.5

$$!\mathbf{P}(x) = x \quad \forall x \in M^P$$

6. **-P**

- **pre-condition:** **-P** can only be be applied when the current state contains only one P. That is, there is only one open agreement covering this win condition. When the only open agreement is dropped, P is therefore eliminated.

$$x \in \overline{M^P} \Leftrightarrow -\mathbf{P}(x) \in S$$

Axiom 6.4.6

$$-\mathbf{P}(x) = x_1 x_2 \quad \forall x = x_1 P x_2 \; s.t. \; x \in \overline{M^P}$$

7. **!V**

- **pre-condition:** **!V** can only be be applied when in the current state, there are multiple vote-in-progress agreements (V) covering this win condition. When one vote-in-progress agreement is dropped, there is still another vote-in-progress agreement to keep V.

$$x \in M^V \Leftrightarrow \mathbf{!V}(x) \in S$$

Axiom 6.4.7

$$\mathbf{!V}(x) = x \; \forall x \in M^V$$

8. **-V**

- **pre-condition:** **-V** can only be be applied when the current state contains only one V. That is, there is only one vote-in-progress agreement covering this win condition. When the only vote-in-progress agreement is dropped, V is therefore eliminated.

$$x \in \overline{M^V} \Leftrightarrow -\mathbf{V}(x) \in S$$

Axiom 6.4.8

$$-\mathbf{V}(x) = x_1 x_2 \quad \forall x = x_1 V x_2 \; s.t. \; x \in \overline{M^V}$$

9. **!C**

- **pre-condition:** **!C** can only be be applied when in the current state, there are multiple passed agreements (C) covering this win condition. When one passed agreement is dropped, there is still another passed agreement to keep C.

$$x \in M^C \Leftrightarrow \mathbf{!C}(x) \in S$$

Axiom 6.4.9

$$\mathbf{!C}(x) = x \quad \forall x \in M^C$$

10. **-C**

- **pre-condition:** **-C** can only be applied when the current state contains onlye one C. That is, there is only one passed agreement covering this win condition. When the only passed agreement is dropped, C is therefore eliminated.

$$x \in \overline{M^C} \Leftrightarrow -\mathbf{C}(x) \in S$$

Axiom 6.4.10

$$-\mathbf{C}(x) = x_1 \quad \forall x = x_1 C \;\; s.t.\, x \in \overline{M^C}$$

11. **!U+P**

- **pre-condition:** **!U+P** can only be applied when the current state has multiple open/addressed issues (U); where some become pre-resolved (P is added), but others do not (U remains).

$$x \in M^U \Leftrightarrow !\mathbf{U} + \mathbf{P}(x) \in S$$

Axiom 6.4.11

$$!\mathbf{U} + \mathbf{P}(x) = +\mathbf{P}(!\mathbf{U}(x)) \quad \forall x \in M^U$$

12. **-U+P**

- **pre-condition:** **-U+P** can only be applied when the current state has only one open/addressed issue (U), which is pre-resolved by a new agreement so U is eliminated and P is added if it was not there.

$$x \in \overline{M^U} \Leftrightarrow -\mathbf{U} + \mathbf{P}(x) \in S$$

Axiom 6.4.12

$$-\mathbf{U} + \mathbf{P}(x) = +\mathbf{P}(-\mathbf{U}(x)) \quad \forall x \in \overline{M^U}$$

13. **!P+V**

- **pre-condition:** **!P+V** can only be applied when the current state has multiple open agreements (P); where some move to vote-in-progress (V added), but others are not (P remains).

$$x \in M^P \Leftrightarrow !\mathbf{P} + \mathbf{V}(x) \in S$$

Axiom 6.4.13

$$!\mathbf{P} + \mathbf{V}(x) = +\mathbf{V}(!\mathbf{P}(x)) \quad \forall x \in M^P$$

14. **-P+V**

- **pre-condition:** **-P+V** can only be applied when the current state has only one open agreement (P) and it moves to vote-in-progress. P is therefore eliminated and V is added if it was not there.

$$x \in \overline{M^P} \Leftrightarrow -\mathbf{P} + \mathbf{V}(x) \in S$$

Axiom 6.4.14

$$-\mathbf{P} + \mathbf{V}(x) = +\mathbf{V}(-\mathbf{P}(x)) \quad \forall x \in \overline{M^P}$$

15. **!V+C**

- **pre-condition:** **!V+C** can only be applied when the current state has multiple vote-in-progress agreements (V); where some become passed (C added), but others are not (C remains).

$$x \in M^V \Leftrightarrow !\mathbf{V} + \mathbf{C}(x) \in S$$

Axiom 6.4.15

$$!\mathbf{V} + \mathbf{C}(x) = +\mathbf{C}(!\mathbf{V}(x)) \quad \forall x \in M^V$$

16. **-V+C**

- **pre-condition:** **-V+C** can only be applied when the current state has only one vote-in-progress agreement (V) whose vote is passed. V is therefore eliminated and C is added if it was not there.

$$x \in \overline{M^V} \Leftrightarrow -\mathbf{V} + \mathbf{C}(x) \in S$$

Axiom 6.4.16

$$-\mathbf{V} + \mathbf{C}(x) = +\mathbf{C}(-\mathbf{V}(x)) \quad \forall x \in \overline{M^V}$$

17. **!P+U**

- **pre-condition:** **!P+U** can only be applied when the current state has multiple open agreements (P) that used to resolve some issues; where some are dropped to cause the corresponding issues to become uncovered (U is added), but others still remain (P remains).

$$x \in \{s| \quad s \in M^P, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = pre - covered)\}$$

$$\Leftrightarrow$$

$$\mathbf{!P + U}(x) \in S$$

Axiom 6.4.17

$$\mathbf{!P + U}(x) = \quad \mathbf{+U}(\mathbf{!P}(x))$$

$$\forall x \in \{s| \qquad s \in M^P, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = pre - covered)\}$$

18. **-P+U**

- **pre-condition:** **-P+U** can only be applied when the current state has only one open agreement (P) that used to resolve some issues and is now dropped. P is therefore eliminated and U is added if it was not there.

$$x \in \{s| \quad s \in \overline{M^P}, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = pre - covered)\}$$

$$\Leftrightarrow$$

$$-\mathbf{P} + \mathbf{U}(x) \in S$$

Axiom 6.4.18

$$-\mathbf{P} + \mathbf{U}(x) = \quad +\mathbf{U}(-\mathbf{P}(x))$$

$$\forall x \in \{s| \qquad s \in \overline{M^P}, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = pre - covered)\}$$

19. **!V+U**

- **pre-condition: !V+U** can only be applied when the current state has multiple vote-in-progress agreements (V) that used to resolve some issues; where some are dropped (U) added), but others are not (V remains).

$$x \in \{s| \quad s \in M^V, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = vote - in - progress)\}$$

$$\Leftrightarrow$$

$$!\mathbf{V} + \mathbf{U}(x) \in S$$

Axiom 6.4.19

$$!\mathbf{V} + \mathbf{U}(x) = \quad +\mathbf{U}(!\mathbf{V}(x))$$

$$\forall x \in \{s| \qquad s \in M^V, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = vote - in - progress)\}$$

20. **-V+U**

- **pre-condition:** **-V+U** can only be applied when the current state has only one vote-in-progress agreement (V) that used to resolve some issues and is now dropped. V is therefore eliminated and U is added if it was not there.

$$x \in \{s| \quad s \in \overline{M^V}, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = vote - in - progress)\}$$

$$\Leftrightarrow$$

$$-\mathbf{V} + \mathbf{U}(x) \in S$$

Axiom 6.4.20

$$-\mathbf{V} + \mathbf{U}(x) = \quad +\mathbf{U}(-\mathbf{V}(x))$$

$$\forall x \in \{s| \qquad s \in \overline{M^V}, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = vote - in - progress)\}$$

21. **!C+U**

- **pre-condition:** **!C+U** can only be applied when the current state has multiple passed agreements (C) that used to resolve some issues; where some are dropped (U is added), but others are not (C remains).

$$x \in \{s| \quad s \in M^C, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = passed)\}$$

$$\Leftrightarrow$$

$$!\mathbf{C} + \mathbf{U}(x) \in S$$

Axiom 6.4.21

$$!\mathbf{C} + \mathbf{U}(x) = \quad +\mathbf{U}(!\mathbf{C}(x))$$

$$\forall x \in \{s| \qquad s \in M^C, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = passed)\}$$

22. **-C+U**

- **pre-condition:** **-C+U** can only be applied when the current state has only one passed agreement (C) that used to resolve some issues and is now dropped. C is therefore eliminated and U is added if it was not there.

$$x \in \{s| \quad s \in \overline{M^C}, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = passed)\}$$

$$\Leftrightarrow$$

$$-\mathbf{C} + \mathbf{U}(x) \in S$$

Axiom 6.4.22

$$-\mathbf{C} + \mathbf{U}(x) = \quad +\mathbf{U}(-\mathbf{C}(x))$$

$$\forall x \in \{s| \qquad s \in \overline{M^C}, \exists w \in W, \exists h \in H^I, \ s.t.$$

$$(state(w) = s) \wedge (head(h) = w) \wedge$$

$$(basic\_state(w) : h \ = passed)\}$$

6.4.3 State Transition Computation

In this section, theories and algorithms are developed to compute the next states by applying feasible operators to the current state.

6.4.3.1 Next State: Sub States

A sub state s_j is the next state of another sub state s_i if and only if there exists an state transition operator t and by applying t to s_i, the result is s_j.

Definition 6.4.27 $\forall s_i \in S,\ next(s_i) = \{s_j | s_j \in S \wedge (\exists t \in T \wedge t(s_i) = s_j)\}$

Theorem 6.4.1

$$\exists s_i, s_j \in S,\ s_j \in next(s_i) \Leftrightarrow \exists t \in T \wedge t(s_i) = s_j$$

Algorithm 6.1 Next-State-Sub(S,T)

For each state CurrentState *in S*

- *Print* CurrentState
- *next(*CurrentState *)* $\leftarrow \phi$.
- *For each operator OP in T*
  - NextState $\leftarrow$ *OP(s)*
  - *If* NextState *in S,*
    - *Print (*CurrentState *, OP,* NextState *)*
    - *next(*CurrentState *)* $\leftarrow$ {NextState } $\cup$ *next(*CurrentState *)*
- *Print next(*CurrentState *)*

6.4.3.2 Next State: Super States

A super state s_n is the next state of another super state s_m if and only if s_n has a sub state s_{nj} that is the next state of s_{mi}, whose super state is s_m.

Definition 6.4.28

$$\forall \quad s_m \in S_a,$$

$$next(s_m) \quad = \{s_n | (s_n \in S_a), \wedge$$

$$(\exists s_{mi} \in sub(s_m), s_{nj} \in sub(s_n) \wedge s_{nj} \in next(s_{mi}))\}$$

Theorem 6.4.2

$$\exists s_m, s_n \in S_a,\ s_n \in next(s_m) \Leftrightarrow$$

$$\exists s_{mi} \in sub(s_m), s_{nj} \in sub(s_n) \wedge s_{nj} \in next(s_{mi})$$

Algorithm 6.2 Next-State-Super(S_a)

For each state CurrentState *in* S_a

- *Print* CurrentState
- *next(*CurrentState *)* $\leftarrow \phi$.
- *For each sub state* SubState *in sub(*CurrentState *),*
  - *For each sub state* NextSub *in next(*SubState *)*
    - *next(*CurrentState *)* $\leftarrow$ *next(*CurrentState *)* $\cup$ *super(*NextSub *)*
- *Print next(*CurrentState *)*

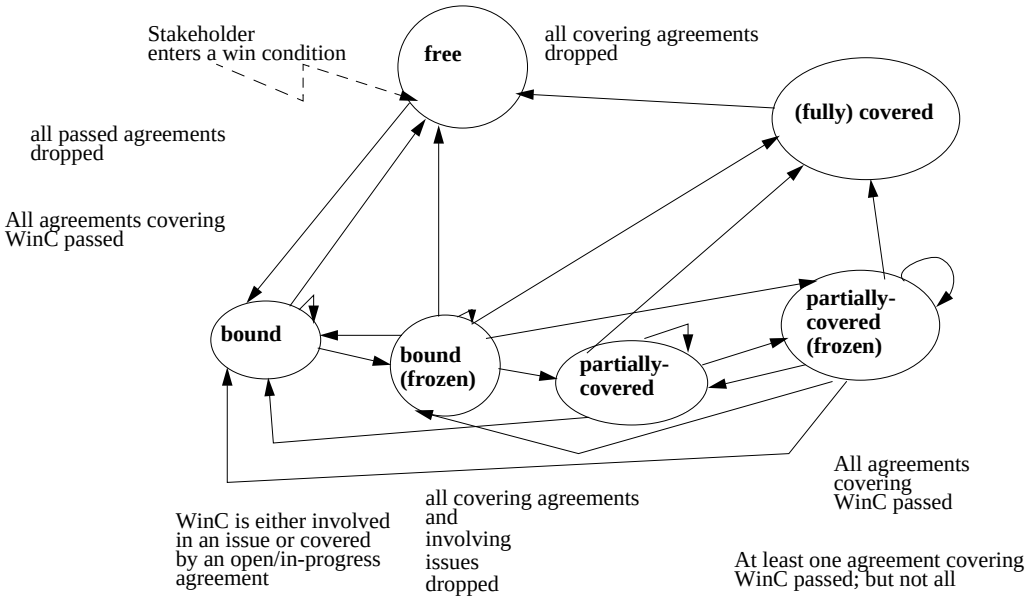


Figure 6.10: Top level of the hierarchical win condition life cycle model

6.4.4 Augmented Hierarchical State Model

For any win condition w, it is associated with only one of the following states at a given layer of the augmented win condition hierarchical state model.

6.4.4.1 free(λ)

- **English description:** w is neither involved in any issue nor covered by any agreement.
- **Definition using basic states:** $\neg((U) \vee (P) \vee (V) \vee (C)$
- **Definition in predicate logic:**

  $(\forall(a \in A)|(w \notin covers(a)) \wedge (w \notin involves(i)))$

- **Next states:**

  On the top level: bound

  On the second level: pre-covered(P), and uncovered(U).

6.4.4.2 bound

- **English description:** w is involved in some open/addressed issue(s) and/or covered only by some open agreement(s).

- **Definition using basic states:** $((U) \vee (P)) \wedge (\neg(V)) \wedge (\neg(C))$

- **Definition in predicate logic:**

$$(\exists(a \in A) \quad | \quad (w \in covers(a))) \vee (\exists(i \in I) | (w \in involves(i))) \wedge$$

$$(\forall(a \in A) \quad | \quad (w \in covers(a)) \rightarrow (state(a) \in \{open, failed\}))$$

- **Next states:** free, bound and bound-frozen.

- **Substates:**

  1. **uncovered(U)**[1]

     - **English description:** w is only involved in some open/addressed issue(s) but not covered by any agreement.

[1]The augmented "uncovered" differs from the basic "uncovered" in that the basic one does not exclude the existence of the other two basic states "pre-covered" and "covered"

- **Definition using basic states:** $(U) \wedge (\neg(P)) \wedge (\neg(V)) \wedge (\neg(C))$
- **Definition in predicate logic:**

$$(\exists(i \in I)|(w \in involves(i)) \wedge (state(i) \in \{open, addressed\}))\wedge$$
$$(\forall(a \in A)|(w \notin covers(a)) \vee (state(a) = failed))$$

- **Next states:** free(λ), pre-covered(P), uncovered+pre-covered(UP), and uncovered(U).

2. **pre-covered(P)**[2]
   - **English description:** w is only covered by some open agreement(s) (which implies if w is involved in any issue, the state of its involving issue must be pre-resolved).
   - **Definition using basic states:** $(\neg(U)) \wedge (P) \wedge (\neg(V)) \wedge (\neg(C))$
   - **Definition in predicate logic:**

$$(\quad \forall(i \in I)|(w \in involves(i)) \in (state(i) = pre - resolved))\wedge$$
$$(\quad \exists(a \in A)|(w \in covers(a)) \wedge (state(a) = open))\wedge$$
$$(\quad \forall(a \in A)|$$
$$(w \in covers(a)) \rightarrow (state(a) \notin \{vote - in - progress, passed\}))$$

[2]Like the augmented "uncovered," the augmented "pre-covered" differs from the basic "pre-covered" in that the basic one does not exclude the existence of the other two basic states "uncovered" and "covered".

- **Next states:** uncovered(U), free(λ),

  pre-covered+vote-in-progress(PV), vote-in-progress(V),

  pre-covered(P), and uncovered+pre-covered(UP).

3. **uncovered+pre-covered(UP)**

   - **English description:** w is involved in some open/addressed issue(s) and covered by some open agreement(s) but not covered by any passed agreements.
   - **Definition using basic states:** $((U) \wedge (P)) \wedge (\neg(V)) \wedge (\neg(C))$
   - **Definition in predicate logic:**

$$
\begin{aligned}
&(\quad \exists(i \in I) | (w \in involves(i)) \wedge (state(i) \in \{open, addressed\})) \wedge \\
&(\quad \exists(a \in A) | (w \in covers(a)) \wedge (state(a) = open)) \wedge \\
&(\quad \forall(a \in A) | \\
&\qquad (w \in covers(a)) \rightarrow (state(a) \notin \{vote - in - progress, passed\}))
\end{aligned}
$$

   - **Next states:** uncovered(U),

     uncovered+pre-covered+vote-in-progress(UPV),

     uncovered+vote-in-progress(UV), pre-covered(P),

     uncovered+pre-covered(UP).

6.4.4.3 bound(frozen)

- **English description:** w is involved in some open/addressed issue(s) and covered by some vote-in-progress but no passed agreement(s).
- **Definition using basic states:** $(V) \wedge (\neg(C))$
- **Definition in predicate logic:**

$$(\exists(a \in A)|(w \in covers(a))) \vee (\exists(i \in I)|(w \in involves(i))) \wedge$$
$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = vote - in - progress)) \wedge$$
$$(\forall(a \in A)|(w \in covers(a)) \rightarrow (state(a) \neq passed))$$

- **Next states:** partially-covered, covered, partially-covered-frozen, free, bound-frozen and bound.
- **Substates:**

1. **vote-in-progress(V)**
   - **English description:** w is only covered by some vote-in-progress agreement(s).
   - **Definition using basic states:** $(\neg(U)) \wedge (\neg(P)) \wedge (V) \wedge (\neg(C))$

- **Definition in predicate logic:**

$$
\begin{aligned}
&(\quad \forall(i \in I)|(w \in involves(i)) \rightarrow (state(i) \notin \{open, addressed\}))\wedge \\
&(\quad \exists(a \in A)|(w \in covers(w)) \wedge (state(a) = vote - in - progress))\wedge \\
&(\quad \forall(a \in A)| \\
&\qquad (w \in covers(w)) \rightarrow (state(a) \notin \{open, passed\}))
\end{aligned}
$$

- **Next states:** uncovered (U), vote-in-progress (V), free(λ), vote-in-progress+covered (VC), covered(C), pre-covered+vote-in-progress (PV), and uncovered+vote-in-progress (UV).

2. **uncovered+vote-in-progress(UV)**

- **English description:** w is only involved in some open/addressed issue(s) and covered by only some vote-in-progress agreement(s).
- **Definition using basic states:** $(U) \wedge (\neg(P)) \wedge (V) \wedge (\neg(C))$
- **Definition in predicate logic:**

$$
\begin{aligned}
&(\exists(i \in I)|(w \in involves(i)) \wedge (state(i) \in \{open, addressed\}))\wedge \\
&(\exists(a \in A)|(w \in covers(w)) \wedge (state(a) = vote - in - progress))\wedge \\
&(\forall(a \in A)|(w \in covers(w)) \rightarrow (state(a) \notin \{open, passed\}))
\end{aligned}
$$

- **Next states:** uncovered (U), vote-in-progress (V), uncovered+vote-in-progress+covered (UVC), uncovered+covered (UC), pre-covered+vote-in-progress (PV), uncovered+pre-covered+vote-in-progress (UPV), and uncovered+vote-in-progress (UV).

3. **pre-covered+vote-in-progress(PV)**

   - **English description:** w is not involved in any open/addressed issues and covered only by some open and some vote-in-progress agreements.
   - **Definition using basic states:** $(\neg(U)) \wedge (P) \wedge (V) \wedge (\neg(C))$
   - **Definition in predicate logic:**

$$(\forall(i \in I)|(w \in involves(i)) \rightarrow (state(i) \notin \{open, addressed\}))\wedge$$
$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = open))\wedge$$
$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = vote - in - progress))\wedge$$
$$(\forall(a \in A)|(w \in covers(a)) \rightarrow (state(a) \neq passed))$$

   - **Next states:** uncovered+pre-covered (UP), uncovered+vote-in-progress (UV), pre-covered (P), pre-covered+vote-in-progress (PVC), pre-covered+covered (PC), vote-in-progress (V),

pre-covered+vote-in-progress (PV), and

uncovered+pre-covered+vote-in-progress (UPV).

4. **uncovered+pre-covered+vote-in-progress(UPV)**

   - **English description:** w is involved in some open/addressed issue(s) and covered by some open and some vote-in-progress agreements but not any passed agreements.
   - **Definition using basic states:** $(U) \wedge (P) \wedge (V) \wedge (\neg(C))$
   - **Definition in predicate logic:**

$$(\exists(i \in I)|(w \in involves(i)) \wedge (state(i) \in \{open, addressed\}))\wedge$$
$$(\exists(a \in A)|(w \in covers(a)) \rightarrow (state(a) = open))\wedge$$
$$(\exists(a \in A)|(w \in covers(a)) \rightarrow (state(a) = vote - in - progress))\wedge$$
$$(\forall(a \in A)|(w \in covers(a)) \rightarrow (state(a) \neq passed))$$

   - **Next states:** uncovered+pre-covered (UP),

     uncovered+pre-covered+vote-in-progress+covered (UPVC),

     uncovered+vote-in-progress+covered (UVC),

     uncovered+vote-in-progress (UV),

     pre-covered+vote-in-progress (PV), and

     uncovered+pre-covered+vote-in-progress (UPV).

6.4.4.4 partially covered

- **English description:** among the agreements that cover w, at least one (but not all) is passed and there is no vote-in-progress agreement.
- **Definition using basic states:** $((U) \vee (P)) \wedge (\neg(V)) \wedge (C)$
- **Definition in predicate logic:**

$$((\exists(i \in I)|(w \in involves(i)) \wedge (state(i) \in \{open, addressed\}))\vee$$
$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = open)))\wedge$$
$$(\forall(a \in A)|(w \in covers(a)) \rightarrow (state(a) \neq vote - in - progress))\wedge$$
$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = passed))$$

- **Next states:** partially-covered-frozen, partially-covered, covered, and bound.
- **Substates:**

1. **uncovered+covered(UC)**
   - **English description:** w is involved in some open/addressed issue(s) and covered by some passed agreement(s) but not covered by any open or vote-in-progress agreements.
   - **Definition using basic states:** $(U) \wedge (\neg(P)) \wedge (\neg(V)) \wedge (C)$

- **Definition in predicate logic:**

$$
\begin{aligned}
&(\quad \exists(a \in A)|(w \in covers(a)) \wedge (state(a) = passed)) \wedge \\
&(\quad \exists(i \in I)|(w \in involves(i)) \wedge (state(i) \in \{open, addressed\})) \wedge \\
&(\quad \forall(a \in A)| \\
&\quad (w \in covers(a)) \wedge (state(a) \notin \{open, vote-in-progress\}))
\end{aligned}
$$

- **Next states:** uncovered (U), covered (C), pre-covered+covered (PC), uncovered+pre-covered+covered (UPC), and uncovered+covered (UC).

2. **uncovered+pre-covered+covered(UPC)**

   - **English description:** w is involved in some open/addressed issue(s) and covered by some open and some passed but no vote-in-progress agreements.
   - **Definition using basic states:** $(U) \wedge (P) \wedge (\neg(V)) \wedge (C)$
   - **Definition in predicate logic:**

$$
\begin{aligned}
&(\exists(i \in I)|(w \in involves(i)) \wedge (state(i) \in \{open, addressed\})) \wedge \\
&(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = open)) \wedge \\
&(\forall(a \in A)|(w \in covers(a)) \rightarrow (state(a) \neq vote-in-progress)) \wedge \\
&(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = passed))
\end{aligned}
$$

- **Next states:** uncovered+pre-covered (UP),

  uncovered+covered (UC),

  uncovered+pre-covered+vote-in-progress+covered (UPVC),

  uncovered+vote-in-progress+covered (UVC),

  pre-covered+covered(PC), and

  uncovered+pre-covered+covered(UPC).

3. **pre-covered+covered(PC)**

   - **English description:** w is not involved in any open/addressed issue and covered by some open and some passed but no vote-in-progress agreements.
   - **Definition using basic states:** $(\neg(U)) \wedge (P) \wedge (\neg(V)) \wedge (C)$
   - **Definition in predicate logic:**

     $$(\forall(i \in I)|(w \in involves(i)) \wedge (state(i) \in \{open, addressed\}))\wedge$$
     $$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = open))\wedge$$
     $$(\forall(a \in A)|(w \in covers(a)) \rightarrow (state(a) \neq vote - in - progress))\wedge$$
     $$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = passed))$$

   - **Next states:** uncovered+pre-covered (UP),

     uncovered+covered (UC), pre-covered (P), covered (C),

     pre-covered+vote-in-progress+covered (PVC),

vote-in-progress+covered (VC), pre-covered+covered (PC), and uncovered+pre-covered+covered (UPC).

6.4.4.5 partially covered(frozen)

- **English description:** among the agreements that cover w, at least one is vote-in-progress and another is passed.
- **Definition using basic states:** $(V) \wedge (C)$
- **Definition in predicate logic:**

$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = vote - in - progress)) \wedge$$
$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = passed))$$

- **Next states:** covered, partially-covered-frozen, partially-covered and bound-frozen.
- **Substates:**

1. **uncovered+vote-in-progress+covered(UVC)**
   - **English description:** w is involved in some open/addressed issue(s) and covered by some vote-in-progress and some passed but no open agreements.
   - **Definition using basic states:** $(U) \wedge (\neg(P)) \wedge (V) \wedge (C)$

- **Definition in predicate logic:**

$$(\exists(i \in I)|(w \in involves(i)) \wedge (state(i) \in \{open, addressed\}))\wedge$$

$$(\forall(a \in A)|(w \in covers(a)) \wedge (state(a) \neq open))\wedge$$

$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = vote - in - progress))\wedge$$

$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = passed))$$

- **Next states:** uncovered+vote-in-progress (UV), vote-in-progress+covered (VC), uncovered+covered (UC), pre-covered+vote-in-progress+covered (PVC), uncovered+pre-covered+vote-in-progress+covered (UPVC), and uncovered+vote-in-progress+covered (UVC).

2. **uncovered+pre-covered+vote-in-progress+covered(UPVC)**

- **English description:** w is involved in some open/addressed issue(s) and covered by some open, some vote-in-progress and some passed agreements.

- **Definition using basic states:** $(U) \wedge (P) \wedge (V) \wedge (C)$

- **Definition in predicate logic:**

$$(\exists(i \in I)|(w \in involves(i)) \wedge (state(i) \in \{open, addressed\}))\wedge$$

$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = open))\wedge$$

$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = vote - in - progress))\wedge$$

$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = passed))$$

- **Next states:** uncovered+pre-covered+vote-in-progress (UPV), uncovered+pre-covered+covered (UPC), uncovered+vote-in-progress+covered (UVC), pre-covered+vote-in-progress+covered (PVC), and uncovered+pre-covered+vote-in-progress+covered (UPVC).

3. **pre-covered+vote-in-progress+covered(PVC)**

   - **English description:** w is not involved in any open/addressed issue and covered by some vote-in-progress and some passed but no open agreements.
   - **Definition using basic states:** $(\neg(U)) \wedge (P) \wedge (V) \wedge (C)$

- **Definition in predicate logic:**

$$(\forall(i \in I)|(w \in involves(i)) \rightarrow (state(i) \notin \{open, addressed\}))\wedge$$
$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = open))\wedge$$
$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = vote - in - progress))\wedge$$
$$(\exists(a \in A)|(w \in covers(a)) \wedge (state(a) = passed))$$

- **Next states:** uncovered+vote-in-progress+covered (UVC), pre-covered+vote-in-progress (PV), pre-covered+covered (PC), vote-in-progress (VC), pre-covered+vote-in-progress (PVC), and uncovered+pre-covered+vote-in-progress (UPVC).

4. **vote-in-progress+covered(VC)**

- **English description:** w is covered by some vote-in-progress and some passed but no open agreements.
- **Definition using basic states:** $(\neg(U)) \wedge (\neg(P)) \wedge (V) \wedge (C)$
- **Definition in predicate logic:**

$$(\forall(i \in I)|((w \in involves(i)) \rightarrow (state(i) \notin \{open, addressed\})))\wedge$$
$$(\exists(a \in A)|((w \in covers(a)) \wedge (state(a) = vote - in - progress)))\wedge$$
$$(\forall(a \in A)|((w \in covers(a)) \wedge (state(a) \neq open)))\wedge$$
$$(\exists(a \in A)|((w \in covers(a)) \wedge (state(a) = passed)))$$

- **Next states:** uncovered+vote-in-progress (UV), uncovered+covered (UC), vote-in-progress (V), vote-in-progress+covered (VC), covered (C), pre-covered+vote-in-progress+covered (PVC), and uncovered+vote-in-progress+covered (UVC).

6.4.4.6 (fully) covered

- **English description:** all issues involving w are resolved and all agreements covering w are passed.
- **Definition using basic states:** $(\neg(U)) \wedge (\neg(P)) \wedge (C)$
- **Definition in predicate logic:**

$$(\forall(a \in A)|(w \in covers(a)) \rightarrow (state(a) = passed)) \wedge$$
$$(\forall(i \in I)|(w \in involves(i)) \rightarrow (state(i) = resolved))$$

  - **On the top level:** partially-covered, free, covered and bound.
  - **On the second level:** uncovered (U), covered (C), free (λ), pre-covered+covered (PC), and uncovered+covered (UC).

6.4.5 Functional description of the states for the win condition

Figure 6.2 summarizes the augmented hierarchical state model of the win condition life cycle. The functional definition for the states on the top level of this model is as follows:

$$state(x):\ W \rightarrow \quad \{free, bound, bound(frozen), partially-covered,$$
$$partially-covered(frozen), covered\}$$

The functional definition for the states in the full expansion of this hierarchical model is as follows:

$$W \rightarrow \{\lambda, U, P, V, C, UP, UV, UC, PV, PC, VC, UPV, UPC, UVC, PVC, UPVC\}$$

$$\begin{aligned}
\lambda: &\quad free \\
U: &\quad uncovered \\
P: &\quad pre-covered \\
V: &\quad vote-in-progress \\
C: &\quad covered \\
UP: &\quad uncovered\&pre-covered \\
UV: &\quad uncovered\&vote-in-progress
\end{aligned}$$

UC : $uncovered \ \& \ covered$

PV : $pre-covered \ \& \ vote-in-progress$

PC : $pre-covered \ \& \ covered$

VC : $vote-in-progress \ \& \ covered$

UPV : $uncovered \ \& \ pre-covered \ \& \ vote-in-progress$

UPC : $uncovered \ \& \ pre-covered \ \& \ covered$

UVC : $uncovered \ \& \ vote-in-progress \ \& \ covered$

PVC : $pre-covered \ \& \ vote-in-progress \ \& \ covered$

$UPVC$: $uncovered \ \& \ pre-covered \ \& \ vote-in-progress \ \& \ covered$

The following shows the output generated by the algorithms defined in Section 6.4.3.

Current state is FREE:

Current state is λ:

(λ +P P)

(λ +U U)

Next States of λ: P U

Next States of FREE: BOUND

Current state is BOUND:

Current state is U:

(U !U U)

(U -U λ)

(U !U+P UP)

(U -U+P P)

(U +P UP)

(U +U U)

Next States of U: λ P UP U

Current state is P:

(P !P+U UP)

(P -P+U U)

(P !P P)

(P -P λ)

(P !P+V PV)

(P -P+V V)

(P +P P)

(P +U UP)

Next States of P: U λ PV V P UP

Current state is UP:

(UP !P+U UP)

(UP -P+U U)

(UP !P UP)

(UP -P U)

(UP !U UP)

(UP -U P)

(UP !P+V UPV)

(UP -P+V UV)

(UP !U+P UP)

(UP -U+P P)

(UP +P UP)

(UP +U UP)

Next States of UP: U UPV UV P UP

Next States of BOUND: BOUND-FROZEN BOUND FREE

Current state is BOUND-FROZEN:

Current state is V:

(V !V+U UV)

(V -V+U U)

(V !V V)

(V -V λ)

(V !V+C VC)

(V -V+C C)

(V +P PV)

(V +U UV)

Next States of V: U V λ VC C PV UV

Current state is UV:

(UV !V+U UV)

(UV -V+U U)

(UV !V UV)

(UV -V U)

(UV !U UV)

(UV -U V)

(UV !V+C UVC)

(UV -V+C UC)

(UV !U+P UPV)

(UV -U+P PV)

(UV +P UPV)

(UV +U UV)

Next States of UV: U V UVC UC PV UPV UV

Current state is PV:

(PV !V+U UPV)

(PV -V+U UP)

(PV !P+U UPV)

(PV -P+U UV)

(PV !V PV)

(PV -V P)

(PV !P PV)

(PV -P V)

(PV !V+C PVC)

(PV -V+C PC)

(PV !P+V PV)

(PV -P+V V)

(PV +P PV)

(PV +U UPV)

Next States of PV: UP UV P PVC PC V PV UPV

Current state is UPV:

(UPV !V+U UPV)

(UPV -V+U UP)

(UPV !P+U UPV)

(UPV -P+U UV)

(UPV !V UPV)

(UPV -V UP)

(UPV !P UPV)

(UPV -P UV)

(UPV !U UPV)

(UPV -U PV)

(UPV !V+C UPVC)

(UPV -V+C UPC)

(UPV !P+V UPV)

(UPV -P+V UV)

(UPV !U+P UPV)

(UPV -U+P PV)

(UPV +P UPV)

(UPV +U UPV)

Next States of UPV: UP UPVC UPC UV PV UPV

Next States of BOUND-FROZEN: PARTIALLY-COVERED COVERED

PARTIALLY-COVERED-FROZEN FREE BOUND-FROZEN BOUND

Current state is PARTIALLY-COVERED:

Current state is UC:

(UC !U UC)

(UC !C+U UC)

(UC -C+U U)

(UC !C UC)

(UC -C U)

(UC -U C)

(UC !U+P UPC)

(UC -U+P PC)

(UC +P UPC)

(UC +U UC)

Next States of UC: U C PC UPC UC

Current state is PC:

(PC !C+U UPC)

(PC -C+U UP)

(PC !P+U UPC)

(PC -P+U UC)

(PC !C PC)

(PC -C P)

(PC !P PC)

(PC -P C)

(PC !P+V PVC)

(PC -P+V VC)

(PC +P PC)

(PC +U UPC)

Next States of PC: UP UC P C PVC VC PC UPC

Current state is UPC:

(UPC !C+U UPC)

(UPC -C+U UP)

(UPC !P+U UPC)

(UPC -P+U UC)

(UPC !C UPC)

(UPC -C UP)

(UPC !P UPC)

(UPC -P UC)

(UPC !U UPC)

(UPC -U PC)

(UPC !P+V UPVC)

(UPC -P+V UVC)

(UPC !U+P UPC)

(UPC -U+P PC)

(UPC +P UPC)

(UPC +U UPC)

Next States of UPC: UP UC UPVC UVC PC UPC

Next States of PARTIALLY-COVERED: PARTIALLY-COVERED-FROZEN

PARTIALLY-COVERED COVERED BOUND

Current state is PARTIALLY-COVERED-FROZEN:

Current state is VC:

(VC !C+U UVC)

(VC -C+U UV)

(VC !V+U UVC)

(VC -V+U UC)

(VC !C VC)

(VC -C V)

(VC !V VC)

(VC -V C)

(VC !V+C VC)

(VC -V+C C)

(VC +P PVC)

(VC +U UVC)

Next States of VC: UV UC V VC C PVC UVC

Current state is UVC:

(UVC !C+U UVC)

(UVC -C+U UV)

(UVC !V+U UVC)

(UVC -V+U UC)

(UVC !C UVC)

(UVC -C UV)

(UVC !V UVC)

(UVC -V UC)

(UVC !U UVC)

(UVC -U VC)

(UVC !V+C UVC)

(UVC -V+C UC)

(UVC !U+P UPVC)

(UVC -U+P PVC)

(UVC +P UPVC)

(UVC +U UVC)

Next States of UVC: UV VC UC PVC UPVC UVC

Current state is PVC:

(PVC !C+U UPVC)

(PVC -C+U UPV)

(PVC !V+U UPVC)

(PVC -V+U UVC)

(PVC !P+U UPVC)

(PVC -P+U UVC)

(PVC !C PVC)

(PVC -C PV)

(PVC !V PVC)

(PVC -V PC)

(PVC !P PVC)

(PVC -P VC)

(PVC !V+C PVC)

(PVC -V+C PC)

(PVC !P+V PVC)

(PVC -P+V VC)

(PVC +P PVC)

(PVC +U UPVC)

Next States of PVC: UVC PV PC VC PVC UPVC UPV

Current state is UPVC:

(UPVC !C+U UPVC)

(UPVC -C+U UVC)

(UPVC !V+U UPVC)

(UPVC -V+U UVC)

(UPVC !P+U UPVC)

(UPVC -P+U UVC)

(UPVC !C UPVC)

(UPVC -C UPV)

(UPVC !V UPVC)

(UPVC -V UPC)

(UPVC !P UPVC)

(UPVC -P UVC)

(UPVC !U UPVC)

(UPVC -U PVC)

(UPVC !V+C UPVC)

(UPVC -V+C UPC)

(UPVC !P+V UPVC)

(UPVC -P+V UVC)

(UPVC !U+P UPVC)

(UPVC -U+P PVC)

(UPVC +P UPVC)

(UPVC +U UPVC)

Next States of UPVC: UPV UPC UVC PVC UPVC

Next States of PARTIALLY-COVERED-FROZEN: COVERED

PARTIALLY-COVERED-FROZEN PARTIALLY-COVERED BOUND-FROZEN

Current state is COVERED:

Current state is C:

(C !C+U UC)

(C -C+U U)

(C !C C)

(C -C λ)

(C +P PC)

(C +U UC)

Next States of C: U C λ PC UC

Next States of COVERED: PARTIALLY-COVERED FREE COVERED BOUND

6.5 The Exhaustiveness and Mutual Exclusiveness of the Artifact States

As defined in Section 6.1, the states of an agreement are exhaustive and mutually exclusive.

Rule 6.5.1 *The states of an agreement are exhaustive.*

$$\forall a \in A,$$

$$state(a) \in \{open, vote-in-progress, passed, failed\}$$

or

$$\forall a \in A,$$

$$(state(a) = open) \vee (state(a) = vote-in-progress)$$

$$\vee (state(a) = passed) \vee (state(a) = failed)$$

Rule 6.5.2 *The states of an agreement are mutually exclusive.*

$$\forall \quad a \in A,$$

$$(state(a) = open) \rightarrow (state(a) \notin \{vote-in-progress, passed, failed\}) \ (1)$$

$$(state(a) = vote-in-progress) \rightarrow (state(a) \notin \{open, passed, failed\}) \ (2)$$

$$(state(a) = passed) \rightarrow (state(a) \notin \{open, vote-in-progress, failed\}) \ (3)$$

$$(state(a) = failed) \rightarrow (state(a) \notin \{open, vote-in-progress, passed\})) \ (4)$$

Given the above, the following rules can be proved.

Rule 6.5.3 *The states of an option are exhaustive.*

$$(\forall(o \in O), (state(o) \in \{unused, pre-used, vote-in-progress, used\})$$

or

$$(\forall(o \in O)$$

$$(state(o) = unused) \vee (state(o) = pre-used) \vee$$

$$(state(o) = vote-in-progress) \vee (state(o) = used)$$

Proof by contradiction:

Assume

$$\exists(o \in O)\ s.t.$$

$$(state(o) \neq unused) \wedge (state(o) \neq pre-used) \wedge$$

$$(state(o) \neq vote-in-progress) \wedge (state(o) \neq used)$$

$$\Rightarrow \quad \exists(a \in A)\ s.t.$$

$$(o \in adopts(a))$$

$$(state(a) \neq open) \wedge (state(a) \neq vote-in-progress) \wedge$$

$$(state(a) \neq passed) \wedge (state(a) \neq failed)$$

$$(\quad definitions\ of\ the\ option\ states)$$

$$\Rightarrow \quad \exists (a \in A)\ s.t.$$

$$(state(a) \neq open) \wedge (state(a) \neq vote - in - progress) \wedge$$

$$(state(a) \neq passed) \wedge (state(a) \neq failed)$$

$$(\quad (A \wedge B) \Rightarrow B)$$

$$\Rightarrow \quad \neg(\forall (a \in A)\ s.t.$$

$$(state(a) = open) \vee (state(a) = vote - in - progress)$$

$$\vee (state(a) = passed) \vee (state(a) = failed))$$

$$(\quad negation)$$

$$(\quad Contradicting\ the\ exhaustiveness\ of\ the\ agreement\ states)$$

Rule 6.5.4 *The states of an option are mutually exclusive.*

$$\forall \quad o \in O,$$

$$(state(o) \ = unused) \rightarrow$$

$$(state(o) \notin \{pre - used, vote - in - progress, used\})\ (1)$$

$$(state(o) \ = pre - used) \rightarrow$$

$$(state(o) \notin \{unused, vote - in - progress, used\})\ (2)$$

$$(state(o) \ = vote - in - progress) \rightarrow$$

$$(state(o) \notin \{unused, pre - used, used\})\ (3)$$

$$(state(o) \ = used) \rightarrow$$

$$(state(o) \notin \{unused, pre - used, vote - in - progress\}))\ (4)$$

Proof of (1) by contradiction:

Assume

$$
\begin{aligned}
& \exists o \in O \; s.t. \\
& ((state(o) = unused) \wedge \\
& (state(o) \in \{pre - used, vote - in - progress, used\})) \\
\Rightarrow \quad & ((\forall a \in A, o \notin adopts(a)) \wedge \\
& (\exists a \in A, \; s.t. \\
& (o \in adopts(a)) \wedge \\
& (state(a) \in \{open, vote - in - progress, passed, failed\}) \; (false) \\
(\quad & (A \wedge \neg A) = false) \\
& \vee \\
& (\exists a \in A \; s.t. \; ((o \in adopts(a) \wedge \\
& (state(a) = failed)) \wedge \\
& (state(a) \in \{open, vote - in - progress, passed\})))) \\
(\quad & definitions \; of \; the \; option \; states) \\
\Rightarrow \quad & (\exists a \in A \; s.t. \; ((o \in adopts(a) \wedge \\
& (state(a) = failed)) \wedge \\
& (state(a) \in \{open, vote - in - progress, passed\})))) \\
(\quad & (false \vee A) = A)
\end{aligned}
$$

$\Rightarrow$ $(\exists a \in A$

$(state(a) = failed) \wedge$

$(state(a) \in \{open, vote - in - progress, passed\})))$

$($ $(A \wedge B) \Rightarrow B)$

$\Rightarrow$ $\neg(\forall a \in A$

$\neg((state(a) = failed) \wedge$

$(state(a) \in \{open, vote - in - progress, passed\})))$

$($ $\neg(\exists A\ s.t.\ B) = (\forall A\ s.t.\ \neg B))$

$\Rightarrow$ $\neg(\forall a \in A$

$\neg(state(a) = failed) \vee$

$\neg(state(a) \in \{open, vote - in - progress, passed\}))$

$($ $\neg(A \wedge B) = (\neg A \vee \neg B))$

$\Rightarrow$ $\neg(\forall a \in A$

$(state(a) = failed) \rightarrow$

$\neg(state(a) \in \{open, vote - in - progress, passed\}))$

$($ $\neg A \vee B) = (A \rightarrow B))$

$\Rightarrow$ $\neg(\forall a \in A$

$((state(a) = failed)) \rightarrow$

$(state(a) \notin \{open, vote - in - progress, passed\})))$

$($ $Contradicting\ the\ mutual\ exclusiveness\ of\ the\ agreement\ states)$

Rule 6.5.5 *The states of an issue are exhaustive.*

$$\forall \quad (i \in I),$$

$$(state(i) \in \{open, addressed, pre-resolved, vote-in-progress, resolved\})$$

Rule 6.5.6 *The states of an issue are mutually exclusive.*

$$\forall \quad i \in I,$$

$$(\quad state(i) = open) \rightarrow$$

$$(state(i) \notin \{addressed, pre-resolved, vote-in-progress, resolved\})(1)$$

$$(\quad state(i) = addressed) \rightarrow$$

$$(state(i) \notin \{open, pre-resolved, vote-in-progress, resolved\})(2)$$

$$(\quad state(i) = pre-resolved) \rightarrow$$

$$(state(i) \notin \{open, addressed, vote-in-progress, resolved\})(3)$$

$$(\quad state(i) = vote-in-progress) \rightarrow$$

$$(state(i) \notin \{open, addressed, pre-resolved, resolved\})(4)$$

$$(\quad state(i) = resolved) \rightarrow$$

$$(state(i) \notin \{open, addressed, pre-resolved, vote-in-progress\})(5)$$

Rule 6.5.7 *The basic states of a win condition* w *in a chain* h *are exhaustive.*

$$
\begin{aligned}
\forall \quad & (w \in W), \\
(\quad & \exists h \in H \; s.t. \; head(h) = w) \wedge \\
(\quad & basic\_state(w) : h \in \\
& \{free, uncovered, pre - covered, vote - in - progress, covered\})
\end{aligned}
$$

Rule 6.5.8 *The basic states of a win condition* w *in a specific artifact chain* h *are mutually exclusive.*

$$
\begin{aligned}
\forall \quad & w \in W, \\
(\quad & \exists h \in H \; s.t. \; head(h) = w) \wedge \\
(\quad & basic\_state(w) : h = free) \rightarrow \\
(\quad & basic\_state(w) : h \notin \\
& \{uncovered, pre - covered, vote - in - progress, covered\})(1) \\
(\quad & basic\_state(w) : h = uncovered) \rightarrow \\
(\quad & basic\_state(w) : h \notin \\
& \{free, pre - covered, vote - in - progress, covered\})(2) \\
(\quad & basic\_state(w) : h = pre - covered) \rightarrow \\
(\quad & basic\_state(w) : h \notin \\
& \{free, uncovered, vote - in - progress, covered\})(3)
\end{aligned}
$$

$$
\begin{array}{ll}
(& basic\_state(w) : h = vote - in - progress) \rightarrow \\
(& basic\_state(w) : h \notin \\
 & \{free, uncovered, pre - covered, covered\})(4) \\
(& basic\_state(w) : h = covered) \rightarrow \\
(& basic\_state(w) : h \notin \\
 & \{free, uncovered, pre - covered, vote - in - progress\})(5)
\end{array}
$$

Rule 6.5.9 *The augmented states of a win condition* w *are exhaustive.*

$$
\begin{array}{lll}
\forall & (w \in W), & \\
(& state(w) \in & \\
 & \{\lambda, U, P, V, C, UP, UV, UC, PV, PC, VC, UPV, UPC, UVC, PVC, UPVC\} &
\end{array}
$$

Rule 6.5.10 *The augmented states of a win condition* w *are mutually exclusive.*

$$
\begin{array}{lll}
\forall & w & \in W, \\
 & \forall & s_1, s_2 \in \\
 & & \{\lambda, U, P, V, C, UP, UV, UC, PV, PC, \\
 & & VC, UPV, UPC, UVC, PVC, UPVC\} \; s.t. \\
 & (& (state(w) = s_1) \wedge (state(w) = s_2)) \rightarrow (s_1 = s_2)
\end{array}
$$

Chapter 7

The WinWin Hierarchical Equilibrium Model

The WinWin Equilibrium Model is an elaboration of the WinWin Spiral Model [BBHL95] to guide the stakeholders throughout the negotiation process. In the WinWin requirements negotiation and renegotiation process, all stakeholders work toward achieving the WinWin Equilibrium state, in which all[1] issues are resolved and all win conditions are fully covered by some passed agreements.

The WinWin Equilibrium state can be destabilized when a new win condition is entered or when a "fully-covered" win condition is dropped. It is characterized by:

1. No issue
2. Resolve single issue
3. Resolve multiple issues

[1] all artifacts in this chapter are with status "active"

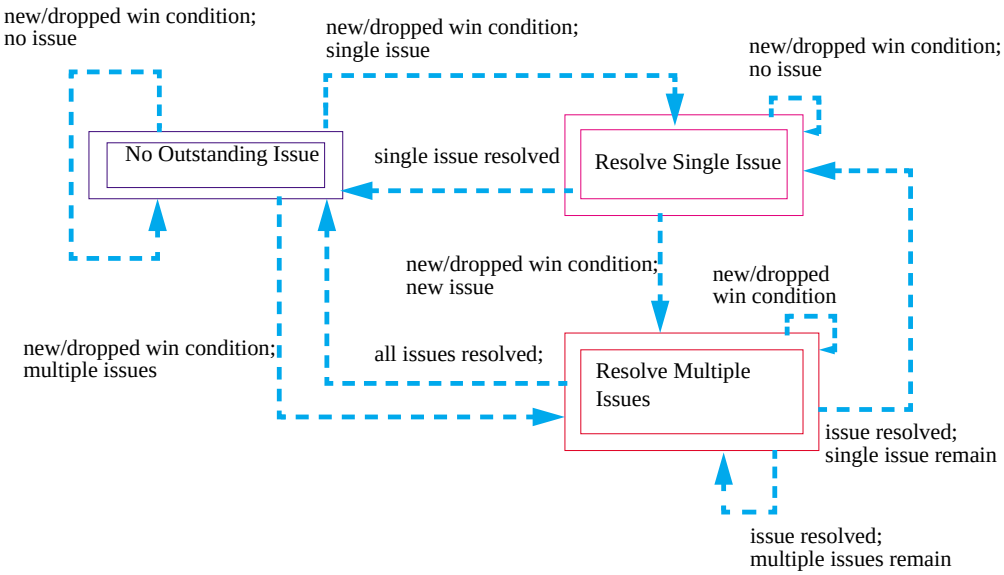


Figure 7.1: Top level model

Figure 7.1 shows the top level of the WinWin Equilibrium Model; Figure 7.2 explains the notation used for the models in this chapter. In [BBHL95], a state transition diagram (STD) for recovering the WinWin equilibrium from a single issue was presented. However, in the real world, stakeholders are facing multiple and inter-locking issues. The hierarchical equilibrium model proposed here is my subsequent work to cover the more complex multiple issues case. The major difficulty encountered here is how to deal with concurrency. Different issues can be in different states simultaneously in the original equilibrium model since, at any given time, a new win condition can be entered and cause a new issue to be raised. To solve this, I divided the states into three mutually exclusive groups: no outstanding issue, resolve single issue, and resolve multiple issues. In addition, I adopt an approach

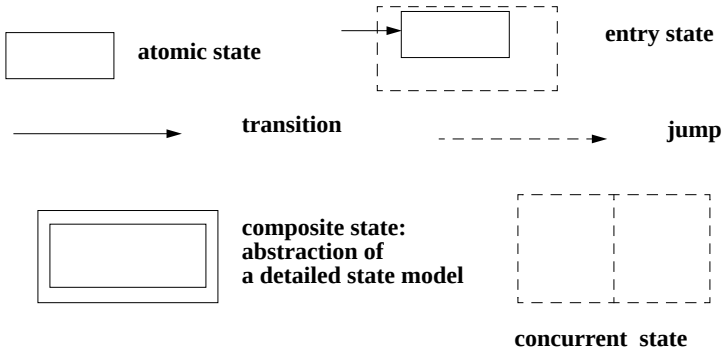


Figure 7.2: Hierarchical state model notation

that accommodates concurrency. The notation used here is a modified version of the dynamic modeling presented in [RBP+91]. Figure 7.2 illustrates the major building blocks. It is a descendant of state transition diagram (STD) that incorporates the concept of concurrency and decomposition. The atomic state, the entry state and the transition in Figure 7.2 are analogous to the state, the starting state and the transition in STD. A composite state contains a group of states that share common conditions. A jump is a special kind of transition. In any sub-state s_1 of a composite state CS_1, a jump causes a transition from s_1 to the starting state of the next composite CS_2. For example, in Figure 7.1, a new win condition can be entered in any sub-state of "No Outstanding Issue." When the new win condition is controversial and raises a new issue, it causes a jump from the current sub-state of "No Outstanding Issue" to the starting state of "Resolve Single Issue." The concurrent state includes several states that can occur simultaneously.

By definitions given previously, an issue is not "resolved" until it is addressed by a "used" option that is adopted by a "passed" agreement. If this condition is not satisfied, this issue is considered *outstanding* and requires negotiation of resolution. When a new win condition comes in the "no outstanding issue" state and does not generate any new issue, the system will stay in "No Outstanding Issue." Otherwise, it will transit from "No Outstanding Issue" to "Resolve Single Issue." In like manner, it can transit to "Resolve Multiple Issues" when this new condition generates more than one issue. "Resolve Single Issue" and "Resolve Multiple Issues" need to be discussed separately since the latter requires considering inter-locking issues.

The following sections show details of the top level states. Every state has a corresponding artifact state combination. Some states in the equilibrium model can be defined using the corresponding artifact state combinations.

It needs to be noted that for every sub-state, the condition of its super-state should always hold and therefore is not restated in the sub-state. For any state, all conditions of its predecessor hold unless otherwise defined.

7.1 No outstanding issue

An outstanding issue is an issue that is not "resolved" by a "passed" agreement. When the negotiation begins, and there are no artifacts created yet, the system is in this "No Outstanding Issue" state. It can also reached if the single issue

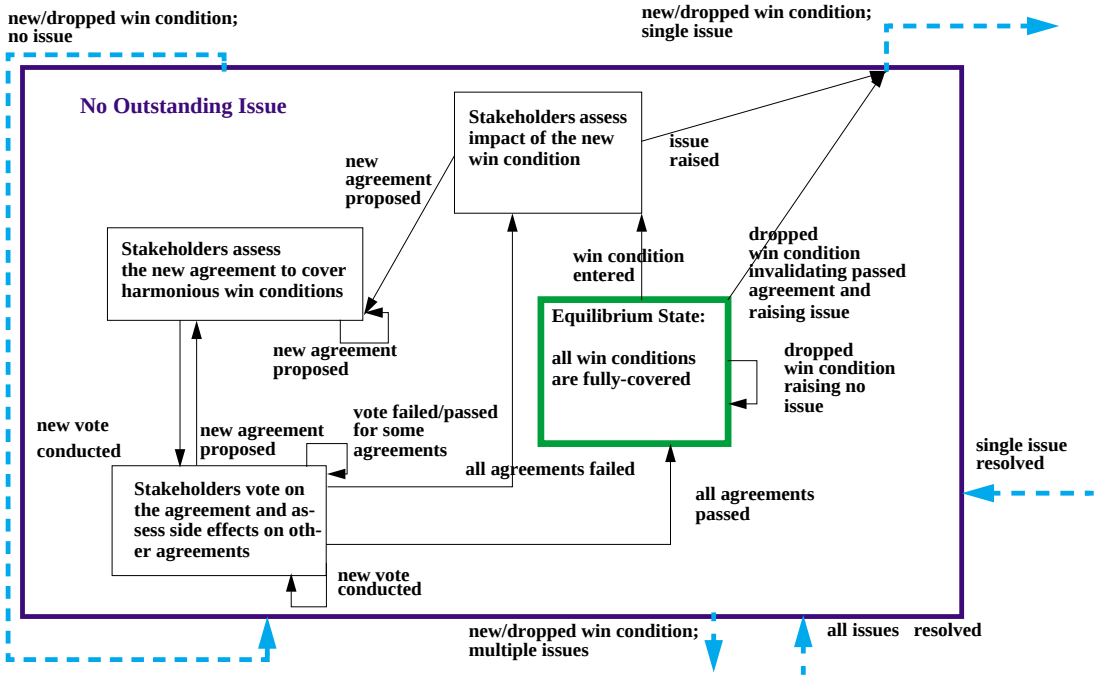


Figure 7.3: No outstanding issue

is resolved/dropped while in the "Resolve Single Issue" state, or if all issues are resolved/dropped while in the "Resolve Multiple Issue" state.

The formal definition of this state is as follows:

$$(\forall(i \in I)|(status(i) = active) \rightarrow (state(i) = resolved))$$

Figure 7.3 shows the sub-states of "No Outstanding Issue." Their definitions will be given in the following sections.

7.1.1 Equilibrium

The WinWin Equilibrium state is the desired goal state for the WinWin equilibrium process in which all win conditions are reconciled and covered by some "passed" agreements. Its definition is as follows:

Definition 7.1.1 *: WinWin Equilibrium State*

$$(\forall(w \in W)|(status(w) = active) \rightarrow (state(w) = fully - covered))$$

It can be proved that the following are implied by Definition 7.1.1.

$$(\forall(i \in I)|(status(i) = active) \rightarrow (state(w) = resolved)) \; -- \; (1)$$

$$(\exists(o \in O)|(status(o) = active) \wedge (state(o) = used)) \; -- \; (2)$$

$$(\forall(a \in A)|(status(a) = active) \wedge (state(a) = passed)) \; -- \; (3)$$

Proof by Contradiction: Assume

$$\exists(i \in I) \; s.t. \; (status(i) = active) \wedge (state(i) \neq resolved)$$

$$\Rightarrow \; \exists(i \in I) \; s.t.$$

$$(status(i) = active) \wedge$$

$$(state(i) \in \{open, addressed, vote - in - progress, pre - resolved\}$$

$$(mutual \; exclusiveness \; of \; states)$$

$\Rightarrow$ $(\exists i \in I)\ s.t.$

$(\exists w \in W) \wedge$

$((w \overset{involves}{\longrightarrow} i) = h) \in CH$

$(state(i) \in \{open, addressed, vote-in-progress, pre-resolved\}$

$(issue\ existence\ rule)$

$\Rightarrow$ $(\exists w \in W)\ s.t.$

$(\exists h \in CH) \wedge$

$(status(w) = active) \wedge$

$(basic\_state(w) : h \in \{uncovered, pre-covered, vote-in-progress\})$

$(definitions\ of\ the\ basic\ win\ condition\ state)$

$\Rightarrow$ $(\exists w \in W)\ s.t.$

$(status(w) = active) \wedge$

$(state(w) \neq fully-covered)$ $[contradicting\ Definition\ 7.1.1]$

$(definitions\ of\ the\ basic\ win\ condition\ state)$

It is straight-forward to prove (2) and (3) using the result and the proving strategy of (1).

As indicated before, the WinWin Equilibrium state can be destabilized either by entering a new win condition or by dropping an existing fully-covered win condition. In the former case, it will transit to the next state. In the latter case, if dropping a win condition invalidates no passed agreements, it will go back to the equilibrium

state. If it does, this invalidated agreement will become inactive and it will transit to "Resolve Single Issue" or "Resolve Multiple Issues" depending on how many issues are raised as a result of the inactive agreement.

7.1.2 Enter win condition

When a new win condition is entered, it is "free" since no other artifacts have references to it.

$$(\exists w \in W | (status(w) = active) \wedge (state(w) = free))$$

Remember that in the equilibrium state, all win conditions are fully-covered. This new win condition now becomes the only exception whose state is not fully-covered. There is actually a stronger condition implied by this argument.

$$(\exists (w \in W) | (state(w) = active) \wedge (state(w) \neq fully - covered))$$

All stakeholders now assess the impact of the new win condition. If no apparent conflicts are found, stakeholders can propose this new win condition to be an open agreement that awaits each stakeholders to commit to this agreement by voting (and this commitment will turn the agreement into a passed one, and the new win condition will become fully covered). Otherwise, some issues will be posted to

address conflicts and options will be proposed to resolve the corresponding issues. In the latter case, it jumps out of the "No outstanding issue" state and goes to either "Resolve Single Issue" or "Resolve Multiple Issues" depending on how many issues are introduced by the new win condition.

7.1.3 Assess new agreement

If the newly entered win condition is not considered controversial, stakeholders can propose one or more agreements to cover that win condition. Then it comes to the "Assess new agreement" state. This state will transit to the next state if a new vote is in progress for some agreement. In this state, there must be at least one new agreement whose state is "open."

$$(\exists a \in A| \quad (status(a) = active) \wedge (state(a) = open))$$

It can be proved that the following is implied by the previous condition and the conditions that are invariant from its preceding state.

$$(\forall(w \in W))| \quad (status(w) = active) \wedge$$
$$((state(w) \neq fully - covered) \rightarrow (state(w) \in \{P, PV, PC, PVC\}))$$

7.1.4 Vote on agreement

When an owner of an "open" agreement finishes his/her draft, he/she can solicit votes on that agreement by entering the voting policy. This action will set the state of the agreement to "vote-in-progress" and come to this "vote on agreement" state.

$$(\exists(a \in A)|(status(a) = active) \wedge (state(a) = vote - in - progress))$$

It can be proved that the following is implied by the previous condition and the conditions invariant from its preceding state.

$$((\exists w \in W)|(status(w) = active) \wedge (state(w) \in \{V, PV, VC, PVC\}))$$

It will move to another state or stay in this state according to the following criteria:

1. if a new vote is conducted, it stays in this state;

2. if all votes are passed, it recovers the "equilibrium" state;

3. if all votes are failed, it goes back to the "assess new win condition" state;

4. if a new agreement is proposed, it goes to the "assess new agreement" state.

7.2 Resolve Single Issue

This state is true when there is only one outstanding issue. It can be reached either when in the "No Outstanding Issue" state, a new/dropped win condition raises one issue, or when in the "Resolve Multiple Issues," all issues but one are resolved/dropped.

In this state, if there is a new/dropped win condition that raises no issue, it will stay in its original state. If the new/dropped win condition causes some new issues, it will go to "Resolve Multiple Issues."

Define $card(s)$ as a function that returns the cardinality of a set s.

The formal definition of this state is as follows to show that there can be only one issue that is not resolved:

$$(I_s = \{i | (status(i) = active) \wedge (state(i) \neq resolved)\}) \wedge (card(I_s) = 1)$$

Define i_s to be the only element in I_s.

$$I_s = \{i_s\}$$

7.2.1 Assess the only issue

In this state, all stakeholders access the only issue to see whether any option can be proposed as a potential resolution to it. Its corresponding artifact life cycle combination is:

$$state(i_s) = open$$

It can be shown that the following is implied:

$$(\exists w \in W | state(w) \in \{U, UP, UV, UC, UPV, UPC, UVC, UPVC\})$$

When options are proposed, stakeholders can move to the next state to negotiate the best feasible option. And if this issue is dropped, it will return to "No Outstanding Issue."

7.2.2 Negotiate the best feasible option

In this state, there are some options proposed to the only issue in the system, waiting to be negotiated. Once stakeholders reach consensus, an agreement will be proposed to adopt the best feasible option(s). If all options are found inapplicable, they will be dropped and the system goes back to the previous state to allow the stakeholders to reassess the issue and repropose options. In this state, no issue can be dropped without dropping all options.

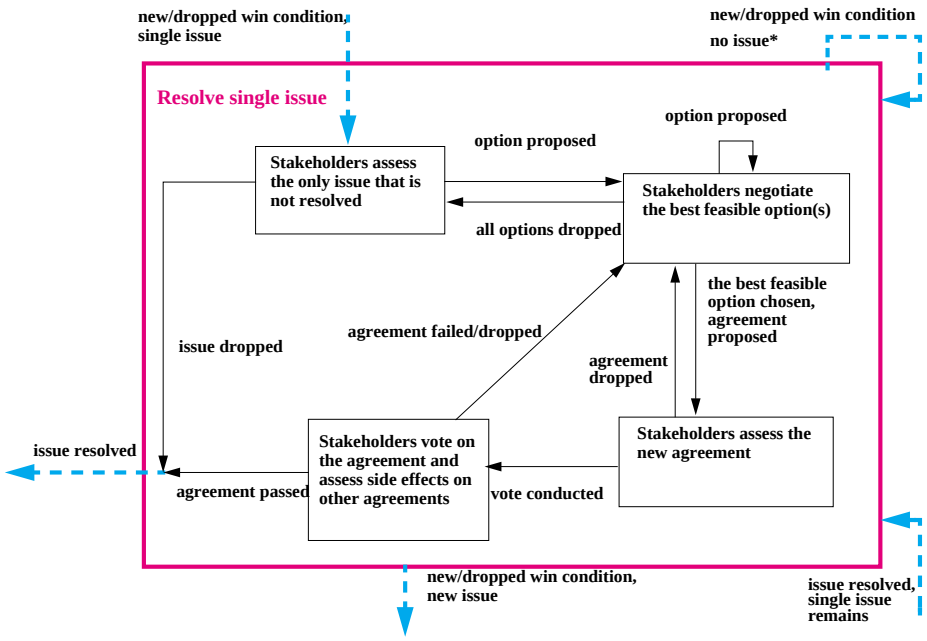


Figure 7.4: Resolve single issue

Its corresponding artifact life cycle combination is:

$$(\exists(o \in O)| \quad (status(o) = active) \wedge$$
$$(state(o) = unused) \wedge (i_s \in addresses(o)))$$

It can be shown the following is implied

$$state(i_s) = addressed$$

7.2.3 Assess agreement

When an agreement is proposed to adopt the best feasible option(s), stakeholders assess the new agreement to see whether it can resolve the only issue. The owner of the agreement can start a vote to solicit stakeholders' consensus and it will move to the next state. If the agreement is not considered viable, it can be dropped and the system will return to the previous state.

Its corresponding artifact life cycle combination is:

$$
\begin{aligned}
(\exists(o \in O), \quad & \exists(a \in A)| \\
& (status(a) = active) \wedge (state(a) = open) \wedge \\
& (status(o) = active) \wedge (o \in adopts(a)) \wedge \\
& (i_s \in addresses(o)))
\end{aligned}
$$

It can be shown that the following are implied:

$$state(i_s) = pre - resolved - -(1)$$

$$(\exists o \in O | state(o) = pre - used) - -(2)$$

$$(\exists w \in W | state(w) \in \{P, PV, PC, PVC\}) - -(3)$$

$$(\forall w \in W | state(w) \notin \{U, UP, UV, UC, UPV, UPC, UVC, UPVC\}) - -(4)$$

7.2.4 Vote on agreement

When an owner of an "open" agreement finishes his/her draft, he/she can solicit votes on that agreement by entering the voting policy. This action will set the state of the agreement to "vote-in-progress" and come to this "vote on agreement" state. If the vote fails or the agreement dropped, the system will return to the previous state. If the vote passes, the only issue is thus resolved and the system will return to the "No Outstanding Issue" state.

Its corresponding artifact life cycle combination is:

$$
\begin{aligned}
&(\exists(o \in O), \quad \exists(a \in A)| \\
&\qquad (status(a) = active) \wedge (state(a) = vote - in - progress) \wedge \\
&\qquad (status(o) = active) \wedge (o \in adopts(a)) \wedge \\
&\qquad (i_s \in addresses(o)))
\end{aligned}
$$

It can be shown that the following are implied:

$$state(i_s) = vote - in - progress - -(1)$$

$$(\exists o \in O | state(o) = vote - in - progress) - -(2)$$

$$(\exists w \in W | state(w) \in \{V, PV, VC, PVC\}) - -(3)$$

$$(\forall w \in W | state(w) \notin \{U, UP, UV, UC, UPV, UPC, UVC, UPVC\}) - -(4)$$

7.3 Resolve Multiple Issues

This state is true when there are multiple outstanding issues. It can be reached from the "No Outstanding Issue" state when a new/dropped win condition raises multiple issues. It can also be reached from the "Resolve Single Issue" state. when one more issue is caused by a new/dropped win condition. After stakeholders negotiate to resolve the many outstanding issues, if no issue remains, the system will return to "No Outstanding Issue." If one issue remains, it will return to "Resolve Single Issue." If multiple issues remain, the system will stay in the "Resolve Multiple Issues."

In this state, if any new/dropped win condition comes in and raises no issue, it does not cause any transition. If any new/dropped win condition causes one or more new issues, it will cause a jump back to the starting state of "Resolve Multiple Issue."

The formal definition of this state is as follows, to show that there must be two distinct issues that are not resolved:

$$
\begin{aligned}
(\exists i_1 \in I, \quad & \exists i_2 \in I \mid \\
& ((status(i_1) = active) \wedge (state(i_1) \neq resolved)) \wedge \\
& ((status(i_2) = active) \wedge (state(i_2) \neq resolved)) \wedge \\
& (i_1 \neq i_2)
\end{aligned}
$$

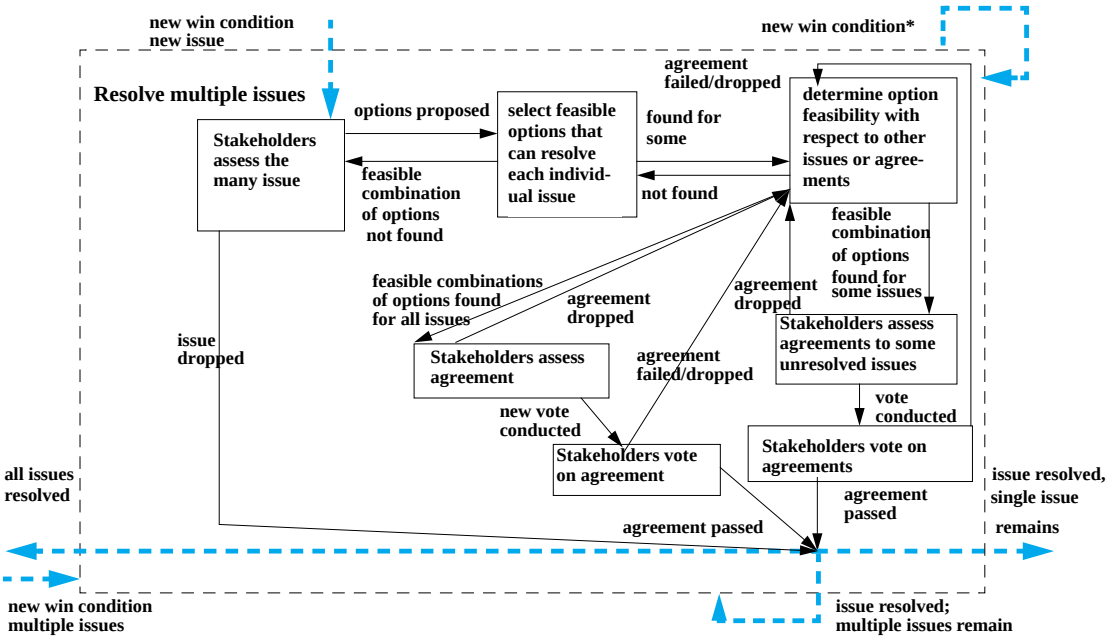


Figure 7.5: Resolve multiple issue

7.3.1 Assess the many issues

This state can be reached by one of the following:

1. in the "No Outstanding Issue," when a new/dropped win condition raises multiple new issues;

2. in the "Resolve Single Issue," when a new/dropped win condition raises some new issue(s);

3. in the "Resolve Multiple Issues," when some agreements are found and passed but they can resolve only part of the issues; and there are still issues that do not have a feasible combination of options which can be used to prepare an agreement;

4. in the "Resolve Multiple Issues," when a new/dropped win condition causes some new issues.

In (1), all the issues must be newly created and of state "open." In (2), the new issue is newly created and "open." In (3), all the remaining outstanding issues must be "open" or "addressed" since they do not have any resolving agreement yet. In (4), the new issue(s) are "open." The following is thus true combining the possibilities discussed above.

$$(\exists i \in I | state(i) \in \{open, addressed\})$$

It can be shown that the following is implied:

$$(\exists w \in W | state(w) \in \{U, UP, UV, UC, UPV, UPC, UVC, UPVC\})$$

In (1), (2) and (4), stakeholders need to address options to the "open" new issue(s). In (3), stakeholders also need to propose new options since the options composed previously do not provide any feasible combination to resolve these issues.

7.3.2 Select feasible options resolving each individual issue

This state can be reached when new option(s) is proposed to an issue. The stakeholders then work on selecting feasible options that can resolve each individual issue.

Another situation is that stakeholders cannot find any feasible combination of options out of the ones selected for each individual one. They thus move back to this state to reselect options for individual issues as alternative candidates. If applicable options are found for some outstanding issues, stakeholders move to "determine option feasibility with respect to other issues and passed agreements." If no options can be applied to any of the issues, the system goes back to the previous state and repropose new options.

As there must be some options addressing each individual issue in discussion, the following is true in this state:

$$\begin{aligned}(\quad & \exists i \in I, \exists o \in O | \\ & (status(o) = active) \wedge (state(o) = unused) \wedge (i \in addresses(o)))\end{aligned}$$

It can be shown the following is implied:

$$(\exists i \in I | state(i) = addressed)$$

7.3.3 Determine option feasibility with respect to other issues or agreements

This state can be reached when stakeholders finish determining feasible options for each individual issue or some previous agreement(s) is/are failed/dropped. Here,

stakeholders need to consider the inter-dependence between artifacts to explore a feasible combination of options that can accommodate the many issues. They do so by comparing the feasible options for different issues. They also need to compare these options with respect to other agreements, especially "passed" agreements. As it is extremely difficult to find a global solution that guarantees all issues to be resolved without violating any previous agreements, a practical way is to try to divide the artifacts into inter-dependent groups and find a partial solution as a stairway to the global solution.

We can use the following hints to help find inter-dependent groups [2]

1. they share common references (for example, two issues involve the same win condition or an issue involve a win condition that is covered by another agreement);

2. one has a reference that is a duplication of the other artifact's reference;

3. stakeholders comments imply their dependency;

4. stakeholders explicitly establish the "related_to";

5. they share taxonomy elements;

6. they share keywords.

[2] The theory in [BI96] of deciding whether two win conditions are in conflict according to their quality attributes can be extended to decide whether two artifacts are inter-dependent

If some feasible combination of options is found to resolve all issues, stakeholders will move to "vote on agreement resolving all issues." If options are only good at resolving some issues, stakeholders will work on "vote on agreement resolving issues." If no feasible combination of options are found, the system will go back to the previous state.

7.3.4 Assess agreements resolving some issue(s)

In the previous state, when a feasible combination of options are found to resolve only some issues, stakeholders can propose agreement(s) to adopt these options. There will be a mix of issues whose options are adopted by some "open" agreements and issues whose options are not adopted by any agreements. Some agreement(s) is(are) then proposed to adopt them.

$$
\begin{aligned}
&(\quad \exists i \in I, \exists a \in A, \exists o \in O | \\
&\quad (status(a) = active) \wedge (state(a) = open) \wedge \\
&\quad (status(o) = active) \wedge (o \in adopts(a)) \wedge (i \in addresses(o))) \wedge \\
&(\quad \exists i \in I | \forall a \in A, \forall o \in O, \\
&\quad (i \in addresses(o)) \rightarrow (o \notin adopts(a)))
\end{aligned}
$$

It can be shown that the following are implied:

$$(\exists i_1 \in I | state(i_1) = pre - resolved) - -(1)$$

$$(\exists o \in O | state(o) = pre - used) - -(2)$$

$$(\exists w \in W | state(w) \in \{P, PV, PC, PVC\}) - -(3)$$

$$(\exists i_2 \in I | state(i_2) \in \{open, addressed\}) - -(4)$$

$$(\exists w \in W | state(w) \in \{U, UP, UV, UC, UPV, UPC, UVC, UPVC\}) - -(5)$$

The stakeholders assess the new agreement(s) to see if it is likely to be a partial solution. Stakeholder(s), when finishing composing the agreement(s), can start a vote to solicit consensus on this partial solution. If the agreement is dropped for some reason, the system will go back to "Determine option feasibility with respect to other issues and agreements" to explore other possible combinations.

7.3.5 Vote on agreements resolving some issues

When vote(s) on "open" agreement(s) resolving some issues is(are) conducted, the agreement(s) is(are) turned to "vote-in-progress." Following the previous state, there are still issues whose options are not adopted by any agreements.

$$
\begin{aligned}
(\quad & \exists i \in I, \exists a_1 \in A, \exists o_1 \in O| \\
& (status(a_1) = active) \wedge (state(a_1) = vote - in - progress) \wedge \\
& (status(o_1) = active) \wedge (o_1 \in adopts(a_1)) \wedge (i \in addresses(o_1))) \wedge \\
(\quad & \exists i \in I | \forall a \in A, \forall o \in O, \\
& (i \in addresses(o)) \rightarrow (o \notin adopts(a)))
\end{aligned}
$$

It can be shown that the following are implied:

$$(\exists i_1 \in I | state(i_1) = vote - in - progress) - -(1)$$

$$(\exists o \in O | state(o) = vote - in - progress) - -(2)$$

$$(\exists w \in W | state(w) \in \{V, PV, VC, PVC\}) - -(3)$$

$$(\exists i_2 \in I | state(i_2) \in \{open, addressed\}) - -(4)$$

$$(\exists w \in W | state(w) \in \{U, UP, UV, UC, UPV, UPC, UVC, UPVC\}) - -(5)$$

If the vote passes and multiple issues remain, the system will go back to the starting state of "Resolve Multiple Issues." If the vote passes and only single issue remains, the system will to "Resolve Single Issue." If the vote fails, stakeholders return to "Determine option feasibility with respect to other agreements and issues" to seek other possible solutions.

7.3.6 Propose agreements resolving all issues

In the previous state, when a feasible combination of options are found to resolve all issues, stakeholders can propose agreement(s) to adopt these options. In this case, all outstanding issues are addressed by options that are adopted by some "open" agreements.

$$
\begin{aligned}
(\quad & \forall i \in I | \exists o \in O, \exists a \in A \; s.t. \\
& (status(o) = active) \wedge (status(a) = active) \wedge \\
& (i \in addresses(i)) \wedge (o \in adopts(a)) \wedge \\
& (state(a) \in in\{open, passed\}))
\end{aligned}
$$

It can be shown that the following are implied:

$$(\forall i \in I | state(i) \in \{resolved, pre-resolved\}) --(1)$$

$$(\exists o \in O | state(o) = pre-used) --(2)$$

$$(\exists w \in W | state(w) \in \{P, PV, PC, PVC\}) --(3)$$

$$(\forall w \in W | state(w) \notin \{U, UP, UV, UC, UPV, UPC, UVC, UPVC\}) --(4)$$

The stakeholders assess the new agreement(s) to see if it is likely to be a global solution. Stakeholder(s), when finishing composing the agreement(s), can start a vote to solicit consensus on this global solution. If the agreement is dropped for

some reason, the system will go back to "Determine option feasibility with respect to other issues and agreements" to explore other possible combinations.

7.3.7 Vote on agreements resolving all issues

When vote(s) on "open" agreement(s) resolving all issues is(are) conducted, the agreement(s) is(are) turned to "vote-in-progress." Following the previous state, all issues are addressed by some options that are adopted by some agreements.

$$
\begin{aligned}
(\quad & \forall i \in I | \exists a \in A, \exists o \in O \ s.t. \\
& (status(a) = active) \wedge (status(o) = active) \wedge \\
& (o_1 \in adopts(a)) \wedge (i \in addresses(o)) \wedge \\
& (state(a) \in \{vote - in - progress, passed\}))
\end{aligned}
$$

It can be shown that the following are implied:

$$(\forall i \in I | state(i) \in \{vote - in - progress, resolved\}) - -(1)$$

$$(\exists o \in O | state(o) = vote - in - progress) - -(2)$$

$$(\exists w \in W | state(w) \in \{V, PV, VC, PVC\}) - -(3)$$

$$(\forall w \in W | state(w) \notin \{U, UP, UV, UC, UPV, UPC, UVC, UPVC\}) - -(4)$$

If the vote passes, all issues should be resolved and the system will jump to "No Outstanding Issue." If the vote fails, stakeholders will return to "Determine option feasibility with respect to other agreements and issues" to seek other possible solutions.

Chapter 8

Integrated Formal Model

Chapters 4 to 7 present the many views of the WinWin requirements engineering process including:

- Win Condition Interaction: a problem space view identifying the Win space of a stakeholder and how it intersects with other stakeholders' Win regions,
- Artifacts and Relationships: schemata and entity-relationship model with functional definitions to support the negotiation infrastructure,
- Artifact Life-Cycles: state diagrams and predicate calculus to demonstrate how artifacts evolve in the negotiation process,
- System Equilibrium: a hierarchical state model and predicate calculus to guide the WinWin users to recover the equilibrium (WinWin) state.

Potential problems with multiple views are inconsistency and confusion. It is possible that the system will tell the user conflicting information in different views.

If the system changes in one view, the changes may not propagate to other views and will result in heterogeneity and cause confusion. Thus, finding a way to integrate the many views becomes a critical issue that should be addressed in the formal modeling. The goal of this chapter is to describe the relationships between the multiple views to establish an integrated model that is consistent and expressive.

Artifact Type	State						
Win Condition	free	uncovered	uncovered	pre-covered	vote-in-progress	covered	uncovered
Issue		unresolved	addressed	pre-resolved	vote-in-progress	resolved	addressed
Option			unused	pre-used	vote-in-progress	used	unused
Agreement				open	vote-in-progress	passed	failed

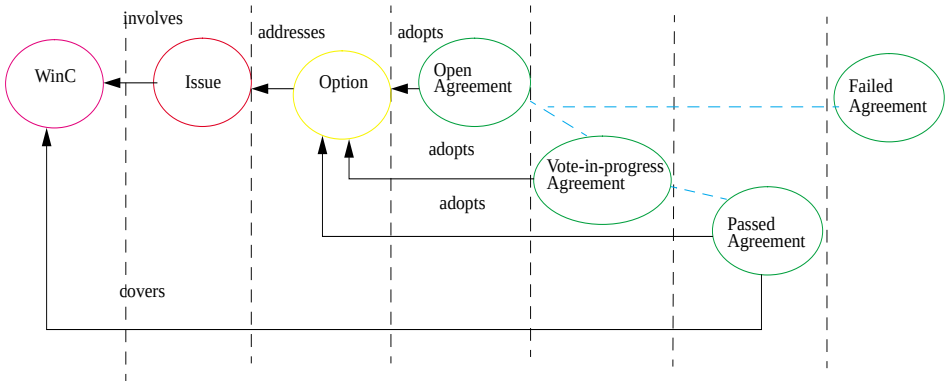


Figure 8.1: Artifact Life Cycle v.s. Artifact Relationships

As described previously, the problem space view models the inter-win-condition relationship and suggests the 4 types of WinWin artifacts with their relationships, which either cover non-controversial win conditions, or reconcile controversial win conditions into passed agreements. Chapter 6 defines the states in the artifact life

cycles as functions of the current negotiation state indicated by the Equilibrium model as well as the inter-artifact relationships. The Problem Space views are identified and correlated to the WinWin Artifacts and their relationships. These Artifact Life cycles, defined as functions reflecting both the inter-artifact relationship and the negotiation state, connect these relationships with the equilibrium model.

Chapter 7 shows that every state in the equilibrium state has a corresponding artifact state combination. Figure 8.1 demonstrates an example of how the state of a win condition evolves with respect to other artifacts that it relates to:

1. when it is just created,
2. when it is involved in an open issue,
3. when the involving issues are addressed by options,
4. when some addressing option is adopted by an open agreement,
5. when that adopting agreement is vote-in-progress (while a vote is being conducted by the stakeholders)
6. when the adopting agreement is voted as passed
7. when the adopting agreement is voted as failed.

For clear illustration, this example intentionally assumes that the win condition is involved in only one issue. A complete enumeration that covers cases when a win

condition is involved in multiple issues and/or covered by multiple agreements is described in Chapter 6.

The state of an artifact shows for the user how far it is from contributing to the equilibrium state. For example, if an agreement is in the state of open, it still needs to be assessed and discussed in order to start a vote. Once it is passed, it indicates that stakeholders have consensus on this agreement and all its covering win conditions are closer to be reconciled.

Figure 8.2 demonstrates a subset of the consolidated model. For every state in the Equilibrium Model, there is a corresponding artifact life cycle combination. This life cycle combination reflects a collection of artifacts connected by relationships (i.e. artifact chains). These artifact chains again represent a specific blending of inter-win-condition relationships. In the WinWin equilibrium state, all win conditions are reconciled and covered by some passed agreements. Its corresponding artifact life cycle combination is discussed in Chapter 7. To make sure that every single win condition is covered and every single issue is resolved, every artifact chain must contain a sub-chain of "Win Condition(covered) $\stackrel{involves}{\longrightarrow}$ Issue (resolved)." By definition of "covered" and "resolved," there must exist a "used" option that is adopted by a "passed" agreement. It can be proved that for all win conditions $w_{i,j}$'s, the intersection of their win regions must be non-empty. The "Resolve single issue" and "Resolve multiple issues" states can be addressed using the same framework as "Equilibrium state."

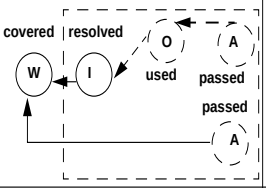
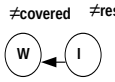
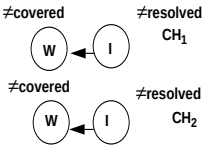
System Equilibrium	Artifact Life Cycles	Inter-Artifact Relationships	Inter-Win-Condition Relationships
Equilibrium state	$\forall w \in W$ $state(w) =$ $C(fully-covered)$	**Every artifact chain CH contains a covered win condition:** 	$\bigcap_{i=1}^{K} \bigcap_{j=1}^{N_i} R(W_{i,j}) \neq \phi$
Resolve single issue	$\exists!\ i \in I$ $state(i) \neq resolved$	**There exists only one artifact chain CH,** 	$\exists!\ I_k$
Resolve multiple issues	$\exists i_1, i_2 \in I, i_1 \neq i_2$ $state(i_1) \neq resolved$ $state(i_2) \neq resolved$	**There exist 2 distinct artifact chains CH_1 and CH_2 containing 2 distinct issues** 	$\exists\ I_1, I_2$ s.t. $I_1 \neq I_2$

Figure 8.2: An integrated model of the many views

Chapter 9

Model Implications

The USC-CSE WinWin requirements negotiation system to date has primarily involved exploratory prototyping. It has evolved from WinWin-0, a version built on top of a COTS(Commercial Off-The-Shelf) tool $CACE/PM^{TM}$ by Perceptronics; through the WinWin-1 version; to the current version WinWin-95. WinWin-1 and WinWin-95 were both developed by USC-CSE. The initial scenario for the WinWin system is to assist stakeholders to achieve the WinWin equilibrium state in which all win conditions are covered by only passed agreements and there are no unresolved issues. A simple equilibrium model in [BBHL95] was developed to guide the WinWin users toward the equilibrium state. In addition, simple artifact life cycles[Hor96] were formulated to signal the current negotiation state.

When the formal modeling was initiated, it indicated that the original equilibrium model and artifact life cycles did not well address concerns such as concurrent issue negotiation and artifact change management. The simple artifact life cycles

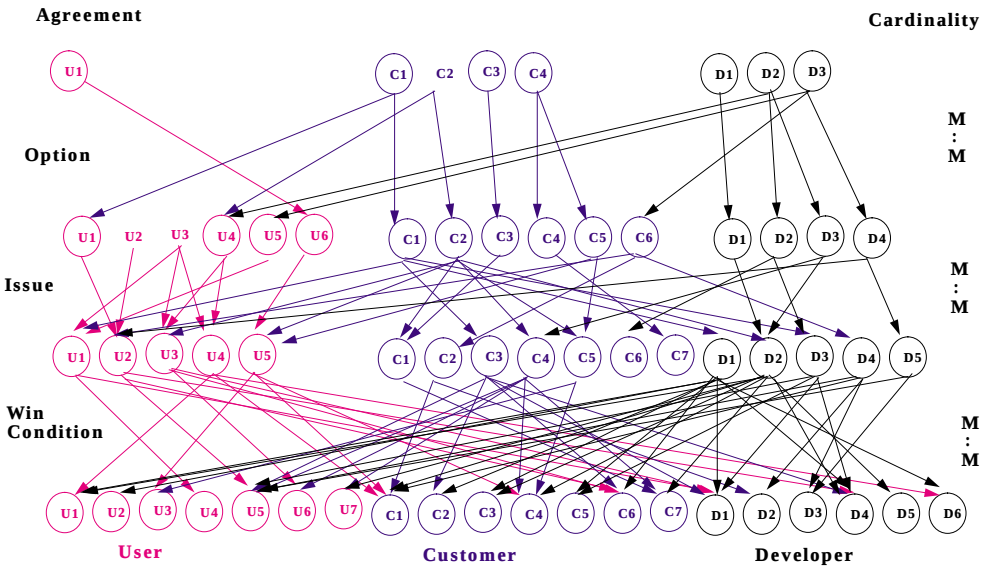


Figure 9.1: Student WinWin Artifact Structure #13

provided very little information about the negotiation state to the WinWin users. This suggested that the WinWin system needed a stronger state summary that highlights artifacts which require the most negotiation.

Moreover, when a WinWin usage analysis of 23 student teams negotiating the requirements for a relatively small-scale library information system using WinWin-95 was performed, it was recognized that many off-nominal cases were over-looked and poorly-managed in the system. An example is as illustrated in Figure 9.1. Consider that the state of a win condition is determined by its involving issues and covering agreement. If the system allows the stakeholders to set up a many-to-many relation between two artifacts, it will result in many possible combinations of situations. How should the state of a win condition under the many possible situations be

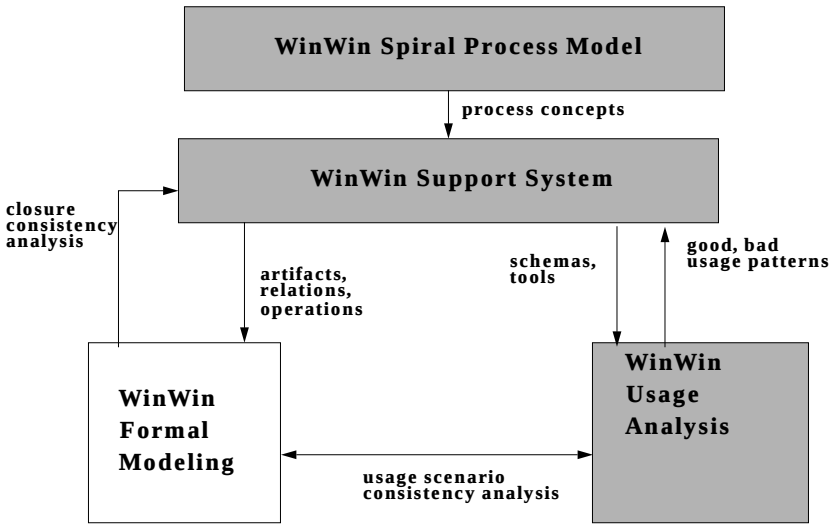


Figure 9.2: Role of the formal modeling

properly determined? And if an option is adopted by two agreements, should its state depend on one agreement being "passed" or both agreements being "passed?" What semantics better models the reality? Answers to these questions are closely connected with our understanding of the system behavior and require further formal analysis. The analysis also demonstrated that users misused the system and became confused about system behavior because some assumptions underlying the system were not well understood.

Figure 9.2 illustrates the role of the formal modeling presented in this thesis. First, it performs consistency closure analysis to the WinWin artifacts, relations, and operations. Second, it stimulates testable hypotheses for the WinWin usage analysis to gather upgrade insights. The following sections outline insights gained

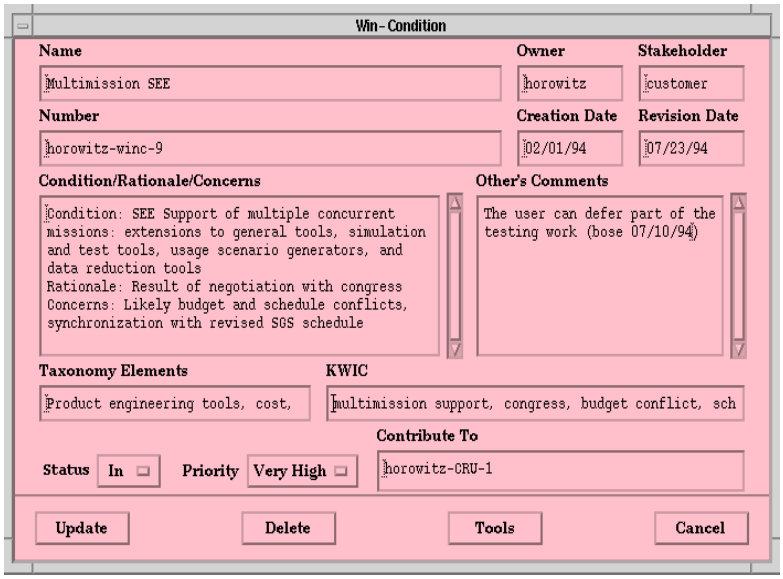


Figure 9.3: WinWin-1 artifact window: Win Condition

from this formal modeling that have helped us upgrade the system and detect aberrant behavior of the system.

9.1 Upgrading insights

9.1.1 Explicit relationships and Referential integrity

Figure 9.3 is a screen dump of a WinWin-1 win condition window. It shows several problems in WinWin-1 for supporting the WinWin process. The first one was ambiguous inter-artifact relationships. In this window, the "contribute_to" slot actually overloaded all possible relationships of a win condition. It confused stakeholders when a win condition could contribute to both a "Point of Agreement (POA: now

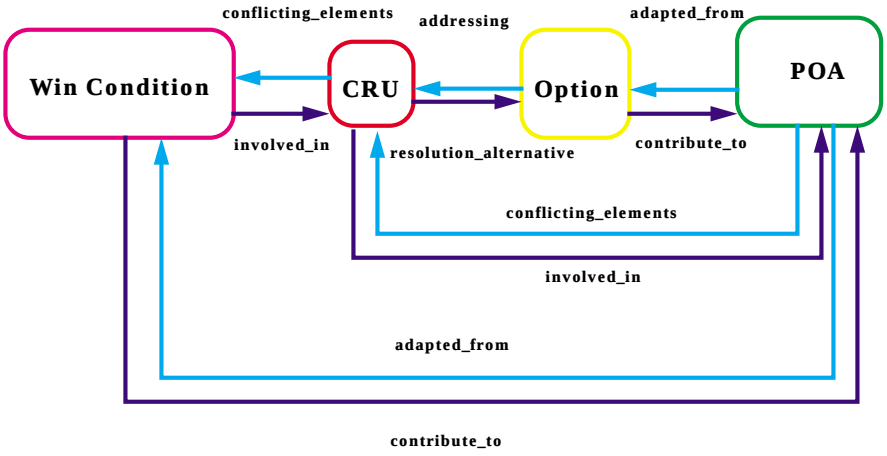


Figure 9.4: First Inter-artifact relationship model

renamed agreement)" and a "Conflict/Risk/Uncertainty(CRU: now renamed issue)" at the same time. Another problem was the value-based reference did not support referential integrity. Stakeholders could get dangling pointers if an artifact was deleted whereas its references were not. Any typing error could result in a dangling pointer. If a stakeholder wanted to look at the content of a particular reference, he/she had to either compose a query or exhaustively search through the artifact menu which was extremely inconvenient and error-prone.

In the proposal of this thesis, the first artifact inter-relationships model was established as shown in Figure 9.4. It provided a good basis for identifying what relationships ought to exist between which pairs of artifacts. It later was revised[1] by the WinWin development team to propose the model in Figure 5.1 that is currently used in WinWin-95. The proposal also made suggestion to change value-based schemas to object-based schemas to enforce referential integrity in the chapter of "Model Implementation." The object-based schemas are now supported by WinWin-95 and facilitate referential integrity and navigation convenience for stakeholders.

9.1.2 Suggesting stronger status summary

In WinWin-1, status and state were mixed in one field to provide a very simple artifact life cycle model. For a win condition, it was *in* or *out* to indicate whether the win condition was still active. For a CRU(issue), it was *resolved* if some option was chosen and a closed POA was reached based on that option or *unresolved*. For an option, it was *open* if it was still under consideration or *dropped*. For an agreement, it was *closed* if stakeholders had reached consensus or *open*. All the above were set by stakeholders manually. In WinWin-95, the original design included only the field status of value "active" and "inactive" for most artifacts and "-," "vote-in-progress," "passed" together with"failed" for an agreement. The formal model advocated the

[1]The relationships were reworded to provide better understanding; and the relationships between agreement(POA) and issue(CRU) were removed to reduce complexity.

necessity of an artifact life cycle model that can reflect the inter-artifact relationships and the negotiation states. Later, an additional attribute—state–for modeling simple artifact life cycles was incorporated to WinWin-95 as follows:

- Agreement
  - —: when an agreement is just proposed
  - vote-in-progress: when an agreement is being conducted a vote
  - passed: when the vote is passed
  - failed: when the vote is failed
- Option
  - unused: when the option is not adopted by any "passed" agreement
  - used: when the option is adopted by a "passed" agreement
- Issue
  - unresolved: when the issue is not addressed by any "used" option
  - resolved: when the issue is addressed by a "used" option
- Win Condition:
  - uncovered: when the win condition is still involved in some unresolved issue or covered by some agreement that has not been passed

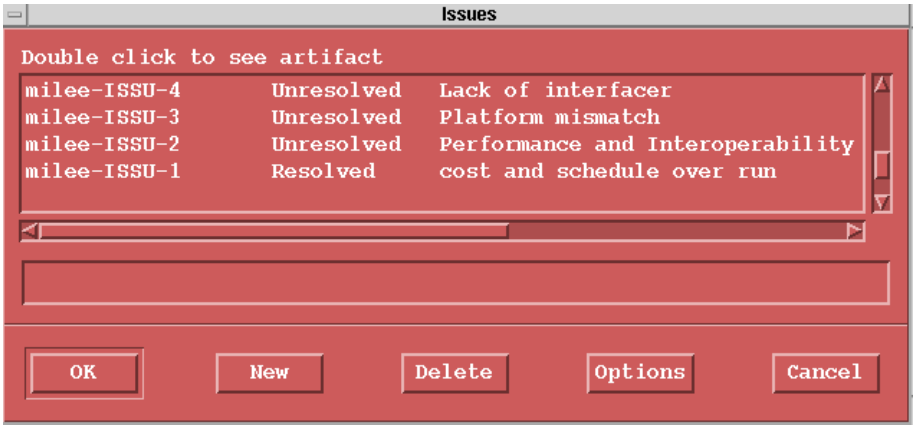


Figure 9.5: WinWin-95 issue state summary

– covered: when the win condition is not involved in any unresolved issue or covered by only "passed" agreements

The formal analysis contends that this model still needs to be extended to suggest a stronger state summary. For example, an issue can be unresolved in many situations. It may be newly created and require proposing options to address this issue. It may have already had options proposed and require stakeholders to negotiate its options. It may be locked because a vote is conducted on the agreement that adopts some option that addresses this issue.

Figures 9.5 and 9.6 show the current issue and win condition summary in WinWin-95. All issues are dichotomized into "resolved" or "unresolved" and all win conditions into "covered" and "uncovered." If a stakeholder wants to know what is a candidate next step to resolve an issue, he/she has to go to that issue, and open up its "referenced_by" menu to see if it is addressed by any options. If there is

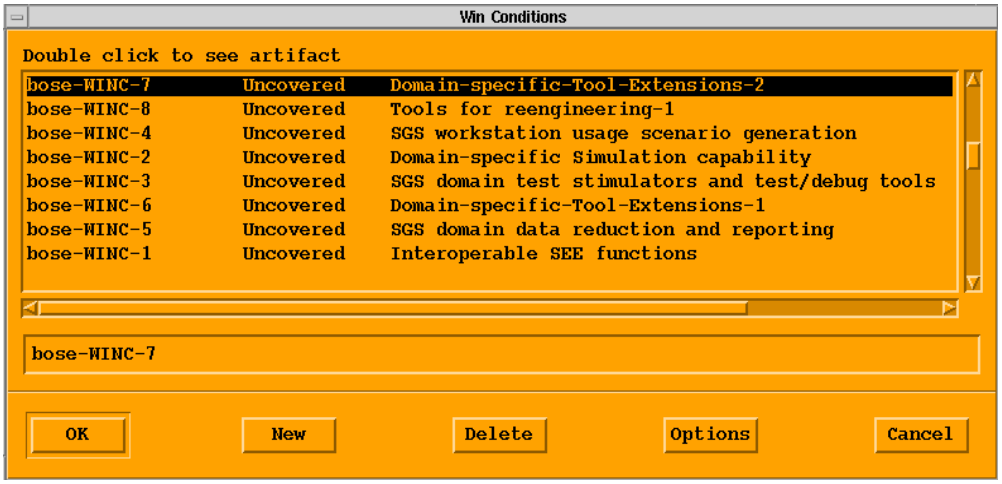


Figure 9.6: WinWin-95 win condition summary

some option addressing it, the stakeholder still needs to assess that option to know if it is adopted by any agreement. And if an issue is locked, the stakeholder will not know the fact until he/she tries to modify its content. If he/she wants to know why this issue is locked, he/she again needs to visit the options and then the agreements to check whether that agreement is "vote-in-progress" or "passed." Stakeholders also cannot prioritize which issue among the many to assess first according to how far it is from being "resolved" by some passed agreement since they do not know what causes an issue to be "unresolved" until they visit the issues one by one and trace the references.

The state summary proposed in Figure 9.7 offers more information to guide stakeholders to recover equilibrium in a more efficient way. It is clear to see

ID	State	Name
milee-ISSU-4	Unesolved	Lack of interfacer
milee-ISSU-3	Pre-resolved	Platform mismatch
milee-ISSU-2	Vote-in-progress	Performance and Interoperability
milee-ISSU-1	Resolved	Cost and schedule over run

Figure 9.7: Suggested issue summary

whether an issue is just newly created(open), has been addressed by an "unused" option(addressed), has been preliminarily resolved by an "open" agreement(pre-resolved), is locked because the resolving agreement is conducted a vote(vote-in-progress) or is locked because the resolving agreement is "passed"(resolved). Stakeholders can also sort the issues according to their states to decide which issue to assess and resolve first.

Similar criteria can be applied to show the inadequacy of the current win condition state summary. The complexity involved in identifying why a win condition is "uncovered" or locked in WinWin-95 is much greater than a single issue, since a win condition can be directly covered by many agreements and involved in many issues, all of different states. Figure 9.8 proposes a win condition state summary that tells stakeholders how many issues without resolving agreements it is involved in, and how many agreements of different states it is covered by, to allow stakeholders to

ID	State				Name
	uncovered(u) [involved in an unresolved issue]	pre-covered(p) [covered by an open agreement]	vote-in-progress(v) [covered by a vote-in-progress agreement]	covered(c) [covered by a passed agreement	
bose-WINC-7				2	Domain-specific-Tool-Extensions-2
bose-WINC-8	3		1	1	Tools for reengineering-1
bose-WINC-4		2		1	SGS workstation usage scenario generation
bose-WINC-2	1			1	Domain-specific Simulation capability
bose-WINC-3	4		2		SGS domain test stimulators and test/debug tools
bose-WINC-6	1				Domain-specific-Tool-Extensions-1
bose-WINC-5	1				SGS domain data reduction and reporting
bose-WINC-1		1			Interoperable SEE functions
egim-WINC-1					test

Figure 9.8: Suggested win condition summary

better sort out which win condition should be visited first and what sequence of actions should be taken to get it fully covered.

9.1.3 Process guidance

The WinWin equilibrium and artifact states suggest process agenda items to lead the WinWin users toward the equilibrium state. Two simple equilibrium models were developed in [BBHL95] and [Hor96]. However, neither one encompassed the multiple-issue aspect. The equilibrium model and the artifact life cycles proposed in this thesis deal with the multiple issue case and offer situated reasoning based on

the current state of the artifacts for the users to determine what sequence of actions to take in order to get to the equilibrium.

9.2 Identifying and preventing potential aberrant behavior

Formalizing the WinWin artifacts and process helps us fully understand system behavior. Analyzing the many possible artifact state combinations formulates a meta-model for generating cases to test if the system responds correctly under different situations. The following are instances that the model was able to identify where WinWin-95 failed to behave as expected:

- Erroneous conditions for locking and unlocking artifacts

  WinWin-95 locked an artifact when it was in the artifact set of a "vote-in-progress" or "passed" agreement and unlocked it when the "vote-in-progress/passed" agreement was inactivated or when the vote was failed. Therefore, when a win condition was covered by a "vote-in-progress" agreement, it was locked. However, if a second agreement covered this win condition, when the vote on the second agreement failed, the system would erroneously unlock the win condition despite that fact that the first agreement still had a vote-in-progress, as shown in Figure 9.9. Artifact life cycles presented in

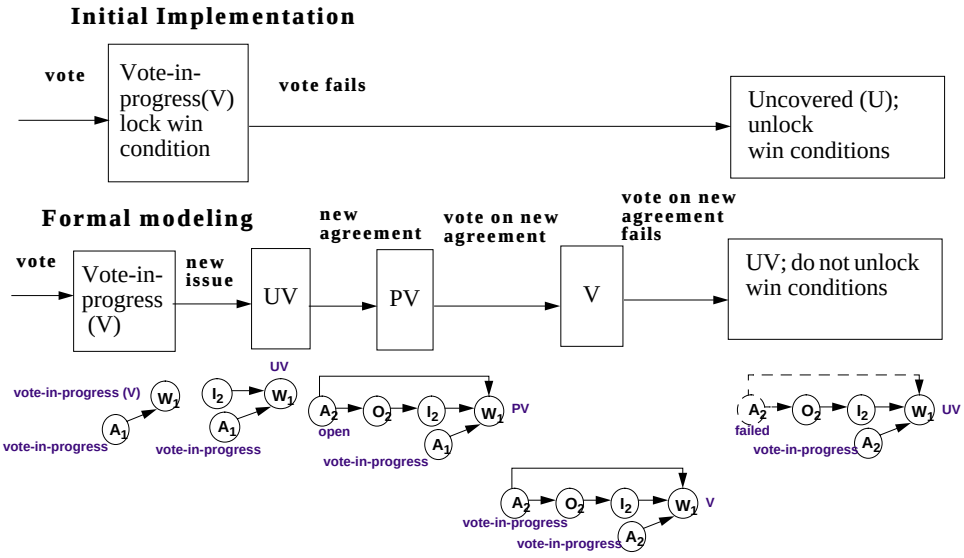


Figure 9.9: Locking problem detected by the model

this thesis offered an exhaustive analysis of all possible states and their corresponding behavior. In this particular case, the resulting "UV(uncovered and vote-in-progress)" state for the win condition indicated that the win condition should remain locked.

- Undesirable state combinations

  In designing WinWin-95, it was assumed every WinWin user would follow the "Win Condition $\stackrel{involves}{\longrightarrow}$ Issue $\stackrel{addresses}{\longrightarrow}$ Option $\stackrel{adopts}{\longrightarrow}$ Agreement" link. That is, they should always create win conditions at first, analyze conflicts between win conditions to suggest an issue involving these controversial win conditions, compose some options to address this issue and propose an agreement

to adopt the best feasible options. However, the WinWin-95 version used for the student project analysis allowed stakeholders to create an issue involving no win conditions, or an option addressing no issue, or an agreement adopting no option or covering no win conditions.

Also, a relationship called "relate_to" was provided to link related artifacts together. Some students used this relationship to replace all the desired relationships for negotiation such as "involves," "addresses," "adopts," and "covers." The system also mistakenly set a win condition to "covered" when all the issues pointing to it by the "relates_to" link were resolved. The rules and artifact state definitions presented before have provided guidance for preventing such undesired state combinations.

Chapter 10

Conclusions

The research described in this thesis models the WinWin requirements negotiation infrastructure and dynamics.

The research involves formal and semi-formal descriptions of multiple views for the WinWin requirements negotiation system, including

- Win Condition Interaction: a problem space view identifying the Win space of a stakeholder and how it intersects with other stakeholders' Win regions;
- Artifacts and Relationships: schemata and an entity-relationship model with functional definitions to support the negotiation infrastructure;
- Artifact Life Cycles: state diagrams and predicate calculus to demonstrate how artifacts evolve in the negotiation process;
- System Equilibrium: a hierarchical state model and predicate calculus to guide the WinWin users to recover the equilibrium (WinWin) state.

As stressed previously, heterogeneous presentations for the same system can cause inconsistency and confusion. Chapter 8 presents how the relationships among the many views are determined to establish an integrated model. The WinWin artifacts and their relationships are mapped onto the problem space view. The artifact life cycles are defined by the WinWin artifact relationships. Each state in the equilibrium model is characterized using the states in the artifact life cycles. Future work can generalize the reconciliation methodology and apply it to other multi-view frameworks.

The many views have also identified initial incompleteness and inconsistency aspects of the WinWin system, and have facilitated artifact change management. The analysis has helped us fully understand the system behavior in the following aspects:

- recognized need to recover equilibrium via the WinWin equilibrium model;
- recognized concurrency-control problems via the WinWin artifact life cycles and the equilibrium model;
- covered off-nominal cases by analyzing the problem space view, the artifact relationships and the artifact life cycles.

In addition, the many possible states identified in the equilibrium and artifact life cycle analyses provided improved summaries for communicating the system status to the WinWin users as follows:

- stronger state summary tables and graphs;
- signals when an artifact is locked.

However, when trying to model the many possible states that should be included in the artifact life cycles and in the WinWin equilibrium model, there are still problems of complexity. To the WinWin users, too many possible states may be hard to understand and make use of. This suggests the need for future work on formulating rules that can simplify the life cycles but still reasonably model the real-world requirements negotiation.

Part III

Bibliography

Reference List

[AC93] W.L. Anderson and W.T. Crocca. Engineering Practice And Codevelopment of Product Prototypes. *Communications of the ACM*, 36(4):49–56, June 1993.

[Alf77] M.W. Alford. A Requirements Engineering Methodology for Real-Time Processing Requirements. *IEEE Transactions on Software Engineering*, 3(1):60–68, January 1977.

[BBHL94a] B.W. Boehm, P. Bose, E. Horowitz, and M.J. Lee. Experimental Results from a Prototype Next-Generation Process Support System. *TRW Systems Integration Group Technology Review*, 2(1):4–27, Summer 1994.

[BBHL94b] B.W. Boehm, P. Bose, E. Horowitz, and M.J. Lee. Software Requirements As Negotiated Win Conditions. In *Proceedings First International Conference on Requirements Engineering*, pages 74–83. IEEE Computer Society Press, April 1994.

[BBHL95] B.W. Boehm, P. Bose, E. Horowitz, and M.J. Lee. Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach. In *Proceedings 17th International Conference on Software Engineering*, pages 243–253. ACM Press, April 1995.

[Ben56] H.D. Benington. Production of Large Computer Programs. In *Proceedings ONR Symposium*, June 1956.

[BGS84] B.W. Boehm, T.E. Gray, and T. Seewaldt. Prototyping Versus Specifying: A Multiproject Experiment Database Systems. *IEEE Transactions on Software Engineering*, 10(3):290–302, May 1984.

[BI96] B. Boehm and H. In. Aids for Identifying Conflicts Among Quality Requirements. *IEEE Software*, 13(2):25–35, March 1996.

[Boe81] B.W. Boehm. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[Boe88] B.W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, 21(5):61–72, May 1988.

[Boz92] J.S. Bozman. Fidelity's Development Plan Leans on JAD and Prototyping. *Computerworld*, pages 73–74, September 21 1992.

[BR89] B.W. Boehm and R. Ross. Theory-W Software Project Management: Principles And Examples. *IEEE Transactions on Software Engineering*, 15(7):902–916, July 1989.

[Buc94] R. Buchness. The WinWin Spiral Model: Rockwell Just in Time Training Course. The Los Angeles Software Process Improvement Network (LA-SPIN) Meeting, November 1994.

[C+88] P. Cook et al. Project Nick: Meetings Augmentation and Analysis. *ACM Transactions on Office Information Systems*, 5(2):132–146, April 1988.

[CY91] E.J. Conklin and K.C.B. Yakemovic. A Process-Oriented Approach to Design Rationale. *Human-Computer Interaction*, 6:357–391, 1991.

[Dav90] A.M. Davis. Specification In a World of Ever-Changing Requirements. In S. Andriole, editor, *Advanced Technologies for Command and Control systems Engineering*, pages 32–47. Fairfax, Va.: AFCEA International Press, 1990.

[EGR91] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware: Some Issues and Experience. *Communications of the ACM*, 34(1):38–58, January 1991.

[F+92] G. Fischer et al. Supporting Software Designers with Integrated Domain-Oriented Design Environment. *IEEE Transactions on Software Engineering*, 18(6):511–522, June 1992.

[FGHW88] F. Flores, M. Graves, B. Hartfield, and T. Winograd. Computer Systems and the Design of Organizational Interaction. *ACM Transactions on Office Information Systems*, 6(2):153–172, April 1988.

[H+92] D. Harris et al. ARIES: The Requirements/Specification Facet for KBSA. Technical Report AL-TR-92-248, USC Information Sciences Institute, Rome Laboratory; Air Force Materiel Command, Griffis Air Force Base, New York, October 1992.

[Hol88] A.W. Holt. Diplans: A New Language for the Study and Implementation of Coordination. *ACM Transactions on Office Information Systems*, 6(2):109–125, April 1988.

[Hor96] E. Horowitz. *WinWin Reference Manual: A System for Collaboration and Negotiation of Requirements.* Center for Software Engineering, University of Southern California, 1.0 edition, March 1996.

[JK94] S. Jacobs and S. Kethers. Improving Communication Decision Making within Quality Function Deployment. In *1st International Conference on Concurrent Engineering, Research and Application*, Pittsburgh, USA, August 1994.

[KBH+92] L. Kawell, S. Beckhardt, T. Halvorsen, R. Ozzie, and I. Greif. Replicated Document Management In A Group Communication System. In D. Marca and G. Bock, editors, *Groupware: Software for Computer-Supported Cooperative Work*, pages 226–235. IEEE Press, 1992.

[KR70] W. Kunz and H. Rittel. Issues as Elements of Information Systems. Technical Report Working Paper No. 131, Institute of Urban and Regional Development, Berkeley, University of California at Berkeley, 1970.

[LYM88] K. Lai, K. Yu, and T.W. Malone. Object Lens: A 'Spreadsheet' for Cooperative Work. *ACM Transactions on Office Information Systems*, 6(4):332–353, October 1988.

[MA93] K.H. Madsen and P.H. Aiken. Experiences Using Cooperative Interactive Storyboard Prototyping. *Communications of the ACM*, 36(4):57–64, June 1993.

[MK93] M.J. Muller and S. Kuhn. Participatory Design: Introduction. *Communications of the ACM*, 36(4):24–28, June 1993.

[NKF94] B. Nuseibeh, J. Kramer, and A. Finkelstein. A Framework for Expressing the Relationships between Multiple Views in Requirements Specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, October 1994.

[P+94] K. Pohl et al. Applying AI Techniques to Requirements Engineering: The NATURE Prototype. In *Proceedings ICSE-Workshop on Research Issues in the Intersection Between Software Engineering and Artificial Intelligence*, May 1994.

[Poh93] K. Pohl. The Three Dimension of Requirements Engineering. In *5th International Conference on Advanced Information Systems engineering*, June 1993.

[PTA94] C. Potts, K. Takahashi, and A.I. Anton. Inquiry-Based Requirements Analysis. *IEEE Software*, 11(2):21–32, March 1994.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Dynamic Modeling*, pages 84–111. Prentice Hall, 1991.

[RD92] B. Ramesh and V. Dhar. Supporting Systems Development by Capturing Deliberations during Requirements Engineering. *IEEE Transactions on Software Engineering*, 18(6):498–510, June 1992.

[Ret93] M. Rettig. Cooperative Software. *Communications of the ACM*, 23(6):23–28, April 1993.

[RKS77] D.T. Ross and Jr. K.E. Schoman. Structured Analysis for Requirements Definition. *IEEE Transactions on Software Engineering*, 3(1):6–15, January 1977.

[Roy70] W. Royce. Managing the Development of Large Software Systems. In *IEEE WESCON*, pages 1–9, August 1970. Reprinted in *Ninth IEEE International Conference on Software Engineering* , Washington D.C.: IEEE Computer Society Press, 1987. pages 328-38.

[S+87] M. Stefik et al. Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. *Communications of the ACM*, 5(2):147–167, January 1987.

[SB82] W. Swartout and R. Balzer. On the Inevitable Intertwining of Specification and Implementation. *Communications of the ACM*, 25(7):438–440, July 1982.

[T+96] K. Takahashi et al. Hypermedia Support for Collaboration in Requirements Analysis. In *Proceedings Second International Conference on Requirements Engineering*, pages 31–40. IEEE Computer Society Press, April 1996.

[TV77] D. Teichroew and K. Vincena. PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. *IEEE Transactions on Software Engineering*, 3(1):41–48, January 1977.

[UC94] USC-CSE. WinWin Spiral Model and Support System Demonstration. USC-CSE Research Review, February 1994.