Requirements Interaction Management

WILLIAM N. ROBINSON
SUZANNE D. PAWLOWSKI
VECHESLAV VOLKOV
Department of Computer Information Systems, Georgia State University, Atlanta, GA 30302; wrobinson@gsu.edu

+1 404 651 3867
http://cis.gsu.edu/~wrobinso

The analysis and management of dependencies among requirements has become a critical area of requirements engineering. This survey reviews this area, herein called Requirements Interaction Management. Requirements interaction management is defined as the *set of activities directed towards the discovery, management, and disposition of critical relationships among sets of requirements*. Using this definition, an evolution of supporting concepts and their related literature is presented. An issues-based framework for reviewing processes and products is presented and applied in the review of the state-of-the-art in requirements interaction management. Finally, seven research projects exemplifying requirements interaction management are presented.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: General—*System architectures, Systems specification methodology*; C.4 [**Performance Of Systems**]—*Modeling techniques, Performance attributes*; D.2.1 [**Requirements/Specifications**]; D.2.2 [**Tools and Techniques**]—*Computer-aided software engineering (CASE)*; D.2.4 [**Program Verification**]; D.2.9 [**Management**]—Life cycle, Software quality assurance (SQA); D.2.10 [**Design**]; H.1.1 [**Models and Principles**]: Systems and Information Theory; I.2.11 [**Artificial Intelligence**]

General Terms: Design, Management, Performance, Reliability, Security, Verification

Additional Key Words and Phrases: Requirements Engineering, system specification, system architecture, Analysis and design, Dependency analysis, Interaction analysis, composite system, WinWin, Telos, distributed intentionality, Viewpoints, KAOS, deficiency driven design, KATE, Oz, Software Cost Reduction (SCR).

GSU CIS Working Paper 99-7

August 30, 1999

TABLE OF CONTENTS

1 INTRODUCTION

An objective of requirements engineering (RE) has been to improve systems modeling and analysis capabilities so that critical aspects of systems can be understood prior to system construction. To that end, RE research spans a wide range of topics. One such topic, of increasing importance, is requirements interaction management.

The purpose of this article is to present a survey of this evolving and important research—work that focuses on the relationships, or interactions, among sets of requirements—an area that the authors have labeled Requirements Interaction Management.[1]

> Requirements Interaction Management is the set of activities directed towards the *discovery, management, and disposition of critical relationships among sets of system or software requirements.*

Although the term Requirements Interaction Management is new, the topics and issues it encompasses have long been recognized as crucial to achieving the objective of a requirements specification that is complete, consistent, and correct.

1.1 Understanding Interactions

Analyzing the extent to which multiple requirements can be satisfied simultaneously has been a challenge for requirements engineering. While single requirement methods have been developed—for example, to minimize network latency or maximize network throughput—such methods typically apply to one or few requirements.

Requirements can interact, often interfering with their achievement. For example, individually two requirements may be achieved on a single processor, but simultaneously achieving both can lead to processor thrashing and the achievement of neither. More generally, a requirement may: (1) deplete a shared resource, (2) remove a pre-condition of another requirement, (3) remove the achieved effect of another requirement, or (4) have other interfering actions.

Modern systems are composed of many components, each of which has many requirements. Moreover, with the growth of object-oriented methods and networked system deployment, component interactions will increase. Satisfactorily achieving all system requirements through component composition is extremely difficult. Achieving certain requirements for one system component can detract from the requirements of other system components. Discovering, tracking, and resolving requirement interactions among system components at requirements definition time remains an open problem.

As Fred R. Brooks indicates, requirements definition is difficult.

> The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements...No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later. [25]

As Peter G. Neumann suggests, such difficulty is only increased when there are multiple interaction requirements.

> The satisfaction of a single requirement is difficult enough, but the simultaneous and continued satisfaction of diverse and possibly conflicting requirements is typically much more difficult. [184]

Despite the importance of the problem, the current state of the art in suffers three major problems, as noted by Axel van Lamsweerde[266].

[1] This is not a survey on the field of requirements engineering. For a general discussion of RE, see [197].

1) The specific kind of interaction being considered is not always clear.
2) There is a lack of systematic techniques for detecting conflicts among non-operational requirements.
3) There is a lack of systematic techniques for resolving negative interactions.

From this brief introduction, Requirements Interaction Management may seem quite similar to the area of Feature Interaction. In fact, both areas are concerned with the detection and resolution of negative interactions. However, the area of Feature Interaction has a narrower view of requirements. Feature Interaction is concerned with the composition and analysis of telephony features, where a *feature* is "an optional unit or increment of functionality."[285] (For example, Call Waiting or Three Way Calling.) In the context of a whole system, features would be composed on multiple requirements. However, Feature Interaction has not considered non-functional requirements. Rather, it has focused on the functional interference among features. As will be described, Requirements Interaction Management includes all types of interactions analysis, as well as how such analysis is managed as part of the overall development life-cycle.

1.2 A History of Problems

In general, requirements errors are: *numerous*, ranging from 25 percent to 70 percent of total software errors—in US companies, averaging one per function point[120]; *persistent*, 2/3 of them are detected after delivery; and *expensive*, fixing requirements errors can cost up to 1/3 of the total production cost[20]. Moreover, many systems failures have been are attributed to poor analysis of requirement interactions[119][155][184].

In general, failures of system interactions can be distinguished from simple component failures[196]. As Nancy G. Leveson observes,

> Whereas in the past, component failure was cited as the major factor in accidents, today more accidents result from dangerous design characteristics and interaction among components[94].—p. 9 [146]

Leveson documents several cases where safety requirements were given lesser priority than other requirements. For example, for the Therac-25 computer controlled radiation therapy machine, system efficiency was given a higher consideration than some safety features.

> The software did not contain self-checks or other error-detection and error-handling features that would have detected the inconsistencies and coding errors. Audit trails were eliminated because of a lack of memory. —p. 550 [146]

The Therac-25 reused software from the prior Therac versions; however, the newer Therac-25 did not have the same hardware characteristics. This, in part, lead to the overdosing of six people from 1985 to 1987[146]. Leveson reminders us that,

> A naive assumption is often made that reusing software or using commercial off-the-self software will increase safety because the software will have been exercised extensively. Reusing software modules does not guarantee safety in the new system to which they are transferred and sometimes leads to awkward and dangerous designs.—p. 552 [146]

Unfortunately, a similar reuse of software led to the destruction of the Ariane 5 launcher[149][188]. Often, the rush to complete a system leads to the dropping of "lesser" non-functional requirements, such as safety or reliability.

Sometimes, interactions between requirements are explicitly considered, yet failures still occur. For example, consider the tradeoff between flight safety, where braking is not to be allowed, and landing safety, where braking is a must. To ensure that pilots did not inadvertently engage the A320's braking system, the software required that the airplane's weight be detected on the wheels.

However, this failed when a pilot attempted to land on a wet, wind-blown runway where the full weight of the plane was not detected[138][139].

> [...] the spoilers, brakes and reverse thrust were disabled for up to 9 seconds after landing in a storm on a waterlogged runway, and the airplane ran off the end of the runway and into a conveniently placed earth bank, with resulting injuries and loss of life.—[139]

System failures, such as the Therac-25, the Lufthansa A320 in Warsaw, and many others arose from (often implicit) undesirable interactions among requirements. A goal of requirement interaction management is to make such critical requirement interactions more obvious during the requirements analysis phase.

1.3 Influencing Factors[2]

At first glance, it may appear simple to support requirement interaction analysis. One need simply formalize the requirements, or at least structure them. Then a computer aided software engineering tool can check the syntax and consistency of the requirements. While Computer Aided Software Engineering (CASE) tools have been successful in providing support for modeling and code generation[33][143][185], they have been less successful in supporting requirements analysis[143].[3] Moreover, requirements analysis is not just about checking the consistency of descriptions. In fact, it is generally desirable to represent inconsistent requirements. This is because requirements represent the needs of system stakeholders—needs that are often inconsistent.

Inconsistency[187], conflict, break-down[278], cognitive dissonance[63]—these are terms that characterize aspects of uncovering unexpected ideas during problem solving. These, and other terms, have been used throughout the requirements engineering literature. (See section 4.2.1.) The general concept of “conflict” has been characterized as a key driver of group communication and productive work[212]. Conflict has been empirically shown to be a driver of systems development[155][163][231] and, more specifically, requirements development[12][128][160][222].

Two basic forces give rise to requirements “conflicts”. First, the technical nature of constructing a requirements document gives rise to *inconsistency*—“any situation in which two parts of a [requirements] specification do not obey some relationship that should hold between them”[53]. Second, the social nature of constructing a requirements document gives rise to *conflict*—requirements held by two or more stakeholders that cause an inconsistency. Applying the general convention in RE, we use the term *conflict* to indicate all types of requirements inconsistencies and conflicts unless the context calls for the use of a more specific term.

Technical and social forces give rise to conflicts that drive essential difficulties of requirements engineering. By managing conflicts, one can manage a key aspect of requirements engineering. Specific problem types will illustrate.

Consider three technical problems that lead to requirements conflicts.

- **Voluminous requirements.** The shear size of a requirements document can lead to conflicts, such as varied used of terminology. This is especially true as requirements are modified.
- **Changing requirements and analysts.** As a requirements document is developed, new requirements are added, older ones are updated. One change request can lead to a cascade of other change requests until the requirements reach a more consistent state. As a result, the document is typically in a transitory state where many semantic conflicts exist, of which most are

[2] This subsection based on a subsection in [226].

[3] In fact, the downstream life-cycle successes of these tools may be one of the reasons that systems analysts spend a greater percent of the time on requirements analysis than ever before[89].

expected to be resolved simply by bringing them to the current state as (implicitly) understood by the analysts. Unfortunately, the implicit current state of requirements is lost when analysts leave a long term project. Moreover, requirement concepts and their expressions vary with the composition of team members. Such changes introduce conflicts.

- **Complex requirements.** The complexity of the domain or software specification can make it difficult to understand exactly what has been specified or how components interact. Implicit requirement dependencies often hide requirements conflicts.

Consider three social problems that lead to requirement conflicts:

- **Conflicting stakeholder requirements.** Different stakeholders often seek different requirements that cannot be mutually achieved. This problem is exacerbated by changing stakeholders.
- **Changing and unidentified stakeholders.** In the attempt to understand system requirements, analysts often seek new stakeholders for an ongoing project. Analysts report that they can understand system requirements when interacting with actual users; however, such access can be difficult to come by[153]. Moreover, one department of an organization may claim to be "the" customer; however, when it comes to the final purchase decision, it may be another department[153]. Such organizational interactions can lead to drastic changes in the requirements.
- **Changing expectations.** In addition to the technical problem of tracking changed requirements, there is the social problem of informing stakeholders of the consequences of changes, as well as managing stakeholders' requests and their expectations of change. Research shows that user behavioral participation and psychological involvement have a positive effect on user satisfaction of development products[10]. User participation is particularly effective during requirements development[128][145][150][160][167].

Requirements interaction management attempts to address such technical and social problems as part of a strategy to manage the conflicts that contribute to the essential difficulties of requirements engineering. Requirements interaction management can address many of the problems by supporting requirements tractability in a dynamic, multi-stakeholder environment. For example, by tracking the statements asserted by analysts and stakeholders as they enact a requirements dialog, one can manage voluminous requirements, and visualize the changes in requirements, the analyst team, or system stakeholders. Problems that are more social can also be addressed. For example, by tracking stakeholder statements, one can find trends (e.g., convergence or divergence) of expectations. Requirements interaction management tools can even support the detection and resolution of multi-stakeholder requirement conflicts (see section 6).

1.4 Article Overview

This survey is presented in three major themes. First, requirements interaction management is defined based on its terms (§2) and history (§3). Second, basic research themes involving the processes (§4) and products (§5) of requirements interaction management are presented. Third, research projects illustrative of requirements interaction management are summarized (§6). Finally, the article concludes (§7) that requirements interaction management has become a critical area of requirements engineering. whose methods will lead to the development of systems with higher stakeholder satisfaction and fewer failures.

2 REQUIREMENTS INTERACTION MANAGEMENT - DEFINITION AND SCOPE

Requirements are descriptions of needs. Components are implementations that can satisfy requirements. Requirement interactions can be understood through direct comparisons of requirements descriptions, or analyses of their underlying components. Requirements interaction management concerns the management of activities that uncover and resolve requirement interactions. In this section we elaborate on the definition of requirements interaction management, beginning with definitions used for the three words that compose this term.

2.1 Requirements

The term, *requirement*, has been defined a number of ways, each emphasizing different aspects of requirements engineering[284]. Central to the definition is a user, or stakeholder, need. For example, Davis states that a requirement is "a user need or a necessary feature, function, or attribute of a system that can be sensed from a position external to that system"[45]. The IEEE standard 610 (1990) has a similar definition:

- A condition or capacity needed by a user to solve a problem or achieve an objective
- A condition or capacity that must be met or possessed by a system or systems component to satisfy a contract, standard, specification or other formally imposed documents
- A documented representation of a condition or capability as in 1 or 2

Such a broad definition includes important specialized requirement types. Each specialized requirement type distinguishes an important aspect of a requirement definition. Below, are common distinctions that can be used to categorize requirements. (For a more refined description, see [197].)

- *System*. Requirements may be distinguished based on the type of system they describe. For example, hardware requirements, software requirements, and requirements on system users. There may even be development requirements concerning the development process (e.g., cost effective, timely) or development aspects of the resulting product (e.g., reusable, maintainable, portable).
- *Functional*. Requirements may be distinguished based on the form of computation that they describe. Functional requirements describe a function (or process) as a relationship between inputs and the resulting outputs.
- *Non-functional*. Requirements may also be distinguished in that they do not define a function, but instead they describe attributes of the execution of a function, such a efficiency and reliability. Non-functional requirements are sometimes referred to as system (or software) qualities.
- *Informational*. Requirements may be distinguished in that they define information, or data, rather than how data is manipulated through functions.
- *Abstraction*. Requirements may be distinguished based on the abstraction level of their description. A requirement may be further defined by add new details defined in more specialized sub-requirements. Through specialization of abstract requirements, or generalization of detailed requirement, a requirement abstraction hierarchy can be defined.
- *Development properties*. Requirements may be distinguished based on their development properties. For example, a requirement may have just been proposed. Later, it may be accepted or rejected.
- *Representational properties*. Requirements may be distinguished based on their representation. A requirement may begin as an informal sketch, then become a natural language sentence (e.g., "The system shall ..."). Finally, more formal representations, such as UML, Z, or predicate cal-

culus, may be used to express a requirement.

2.1.1 Requirement Satisfaction

A requirement is satisfied by a component if the (implemented) component exhibits all the properties specified in the requirement—components operationalize requirements. Consider the following simple requirement.[4]

```
Requirement UseMSWindowSystemPlatform with
InformalDef
   id : "The system shall run on MS Windows 95."
End
```

Now, consider a component that satisfies the requirement.

```
Object NEC6260WindowsNotebook with
InformalDef
   id : "The NEC 6260 Notebook running MS Windows 95."
End
```

The component is a concrete instantiation that satisfies the requirement; e.g., NEC6260WindowsNotebook satisfies UseMSWindowSystemPlatform.

Requirements can be partially satisfied. The degree of satisfaction that a requirement, R, derives from a component, C, can be mapped onto a range:

$Sat_R : C \rightarrow [0,1]$

The above characterization is a utility function for requirement R. Utility theory provides a means to formalize the *value* a requirement obtains from a component[126]. In terms of requirement satisfaction, a 100 percent utility means that a component, C, satisfies requirement R completely.

Fuzzy set theory provides a means to formalize requirement satisfaction into linguistic terms, such as high, or low. Using fuzzy set theory, a mapping from component satisfaction to fuzzy sets can be defined.

$FuzzySat_R : \mu_B(C)$

Above, the satisfaction that a requirement R obtains from component C is mapped onto the fuzzy set *B* (e.g., *B* is the fuzzy set HighSatisfaction[150][151][282]). Both utility theory and fuzzy set theory provide techniques to aggregate requirement satisfaction across a variety of attributes, such as cost, or reliability.

2.2 Interaction

Two requirements, labeled R_1 and R_2, are said to *interact* if (and only if) the satisfaction of one requirement affects the satisfaction of the other. As indicated above, a component is said to *satisfy* a requirement if the component exhibits all the properties specified in the requirement. Thus, if component C_1 satisfies R_1 and component C_2 satisfies R_2, and the run-time behavior of C_1 affects the run-time behavior of C_2, then C_1 interacts with C_2, and indirectly, R_1 interacts with R_2.

As an illustrative example, consider the above UseMSWindowSystemPlatform requirement and the following requirement:[5]

[4] The syntax for requirements is a ConceptBase implementation of KAOS (see sections 5 and 6.4). The language allows for both informal and formal descriptions. Formal aspects of the descriptions will not be considered until section 5.

[5] While these requirements are “low-level” and concern system interface components, they are still requirements and provide a widely accessible introduction to the ideas.

Requirement UseX11R6Libraries **with**
InformalDef
id : "The system shall have an interface that is built on the X11R6 windowing libraries."
End

Now, consider two components that individually satisfy each requirement, the above NEC6260WindowsNotebook component and the following component.

Object GNUX11R6SunLibrary **In** X11R6Library **with**
InformalDef
id : "The GNU X11R6 library for Sun UltraSparc Solaris."
End

Each component is a concrete instantiation that satisfies its corresponding requirement; e.g., GNUX11R6SunLibrary satisfies UseX11R6Libraries. However, together, the system cannot run. The GNU X11R6 Sun library for Sun UltraSparc Solaris consists of binary components that cannot be executed on MS Windows 95—particularly, the NEC 6260 Notebook. Requirements UseMSWindowSystemPlatform and UseX11R6Libraries interact because their corresponding components have a (negative) interaction.

Interactions may be negative, as illustrative above. Requirement R_1 negatively interacts, or conflicts, with requirement R_2 if the satisfaction of R_2 is reduced as the result of satisfying requirement R_1. Conversely, interactions may be positive, in which case the satisfaction of one requirement increases the satisfaction of another requirement.

Circumstances may determine how requirements interact. Reconsider the above two requirements in relation to the following new component, GNUX11R6WindowsLibrary, to replace GNUX11R6SunLibrary.

Object GNUX11R6WindowsLibrary **In** X11R6Library **with**
InformalDef
id : "The Xserver X11R6 library for MS Windows 95."
End

The binary libraries provided with the Xserver X11R6 library for MS Windows 95 can be executed on a MS Windows 95—in fact, that is its purpose. Since many requirements are non-operational abstractions to be satisfied through components, requirements themselves seldom directly interact. Rather, requirements interact indirectly through the components that can satisfy them. (Computer science terms this the *intertwining between specification and design*[258]. Decision science terms this the *means-ends interdependency*[286]. Similarly, AI planning distinguishes between goals and plans[276].)

Actual requirement interactions are always conditional. Consider two requirements, each a logical negation of the other: $R_x \equiv X$ and $R_y \equiv \neg X$. While the two requirements appear to interact (negatively), it may be possible to find component and a context, in which the components do not interact. A simple solution is turn taking. First, satisfy R_x, then later satisfy R_y.

The above discussion of requirements interaction leads to a basic characterization of requirement interaction based on implied operational interactions:

- *Perceived interaction*. Requirements may be perceived to interact, as their descriptions (seem to) imply conditions in which the satisfaction of one requirement affects the satisfaction of another requirement.
- *Operational interaction*. Requirements do interact as components selected to satisfy them affect one another.

Thus, while requirements may appear to interact, their actual interactions can only be determined through the operational components used to satisfy them.

Interaction is not “all or nothing”. Just as requirement satisfaction can vary in degree, requirement interaction can vary in degree, under certain conditions. These other dimensions can further characterize interaction, as described below.

2.2.1 Basis of interaction

Requirements interact, positively or negatively, through some common dependency. An interaction may concern the requirements description; two requirements may share some common description, but still have differences in meaning.[6] For example, two requirements may assign different values to a common object (e.g., $R_{z1} \equiv$ “Z = 5” and $R_{z2} \equiv$ “Z = 6”). An interaction may also concern operational dependencies among requirements. For example, two requirements may imply the consumption of a common scare resource. Individually, each can be satisfied, but there may not be enough of a resource to satisfy both requirements as the system runs. For example, consider two requirements of a file server:

```
Requirement QuickServerResponse with
InformalDef
   id : "The file server shall respond to requests within 0.05 seconds."
End

Requirement Serve200Connections with
InformalDef
   id : "The file server shall serve up to 200 connections simultaneously."
End
```

The file server may be able to respond to an individual request quickly or it may be able to serve 200 connections simultaneously. However, the limited resource of CPU cycles may prevent both requirements from being satisfied simultaneously.

The above interaction is an example of a non-functional performance interaction. That is, the manner in which the file server responses to requests can become undesirable as the volume of requests increase. Thus, the types of requirements (§2.1) involved in an interaction can help to characterize the type of interaction. In general, discovering, categorizing, and resolving such dependencies in an on-going research problem in requirements interaction management.

2.2.2 Degree of interaction

Some interactions are more problematic than others. For example, the above components NEC6260WindowsNotebook and GNUX11R6SunLibrary, result in complete system failure. On the other hand, if component GNUX11R6SunLibrary is replaced by SunSoftMSWindows95Emulator, then the system can run, albeit somewhat slowly.

```
Object SunSoftMSWindows95Emulator with
InformalDef
   id : "The SunSoft MS Windows 95 emulator."
End
```

To see this explicitly, add the non-functional requirement, QuickSystemResponse.

[6] Conversely, they may not share a common description, but may be intended to describe the same concept[247].

```
Requirement QuickSystemResponse with
InformalDef
   id : "The system shall have a quick response time."
End
```

All of the above requirements can be satisfied; however, the use of SunSoftMSWindows95Emulator to satisfy UseX11R6Libraries implies that UseX11R6Libraries somewhat negatively interacts with QuickSystemResponse. Thus, interactions have a range of affect, from total disabling of another requirement, to a moderate reduction of requirement satisfaction. (The same can be said for positive requirement interaction.) To express such degrees of interaction, researchers have used qualitative and fuzzy values. (See section 4.2.1.)

2.2.3 Probability of interaction

The above interactions are static and do not vary with the execution of the system. Conversely, some interactions will depend on the state of system or the system input. Reconsider the above two file server requirements, QuickServerResponse and Serve200Connections. Now consider that, most of the time, the file server can satisfy both requirements. However, if it receives many requests for large files, it may not have the resources to simultaneously respond to all requests within the specified time period. Alternatively, the server may have internal conditions that lead to the failure of the requirements. For example, the server may also have to satisfy a requirement to archive all files (e.g., to tape) at night. If many file requests occur during the archival process, then again, the requirements QuickServerResponse and Serve200Connections may not be satisfied. To express such conditions of interaction, researchers can use probabilities to express the likelihood of interaction.

While the above dimensions of interaction were expressed in terms of a binary interaction between two requirements, *n*-ary interaction can exist, where *n* is greater than two. As an example, reconsider the above file server requirements, QuickServerResponse and Serve200Connections. It may not be the case that QuickServerResponse and Serve200Connections fully consume the CPU cycles; however, as more requirements are added, there may become a point at which the sum total of a subset of requirements does fully consume the CPU cycles, after which any new requirement may create a negative run-time interaction. Of course, expressing, detecting, and resolving *n*-ary interaction becomes more difficult with the number of requirements.

2.3 Requirements Interaction Management

The management aspect of Requirements Interaction Management (RIM) concerns the strategic application of activities for identifying, analyzing, monitoring, documenting, communicating, and restructuring requirement interactions. The activities may be applied within an *ad hoc* or defined process; they may involve the use of special tools and techniques; they may be conducted solely by analysts or include other stakeholders. In any case, the overall goals of the activities include:

- Early detection and resolution of requirement conflicts.
- Increased system efficiency through resource sharing and utilization.
- Increased and varied stakeholder involvement.

Satisfying these goals leads to an overall reduction in system errors and costs, while increasing system effectiveness and stakeholder satisfaction.

Requirements interaction management is still being defined. The major subtopics are: representation, processing, perspectives, and theoretical basis.

- *Representation of requirements, interactions, resolutions, and other products*. Researchers are

defining the set of terms, or ontology, that can be used to describe system products.

- *Processes for discovery, management, and disposition of interactions*. Researchers are defining techniques, some automated, that can be used to manage interactions. A common goal is to provide early life-cycle (a.k.a. design-time) analysis rather than address interactions at system run-time.
- *Perspectives, or views of the process and product*s. Researchers are providing varying views of requirements and their interactions to different systems stakeholders. Additionally, they are integrating their products and processes into the encompassing software development life-cycle.
- *Theoretic basis for representation, process, and perspectives*. Researchers are augmenting traditional requirements engineering theories with theories from Database, Artificial Intelligence, Knowledge Acquisition and Representation, and Social Conflict and Negotiation to establishing a theoretical basis for the support and application of requirements interaction management.

These major areas are summarized in figure 1.

Figure 1 provides a means to classify, or frame, requirements interaction management research. The framework is interpreted as follows. The theories provide a basis for developing new specialized techniques. They are drawn from a variety of disciplines. They include concepts that can be adapted to requirements interaction management, such as database schema integration. Other concepts include models, ontologies, and formalisms for requirement engineering.

The goals of figure 1 define the scope of the requirements interaction management. The perspectives define the interface to the processes and products that define the technological component of requirements interaction management. For example, an analyst typically has access to all the processes and their products during development, while a system user may be provided with a more limited view. The processes define the common activities applied to analyze and modify require-

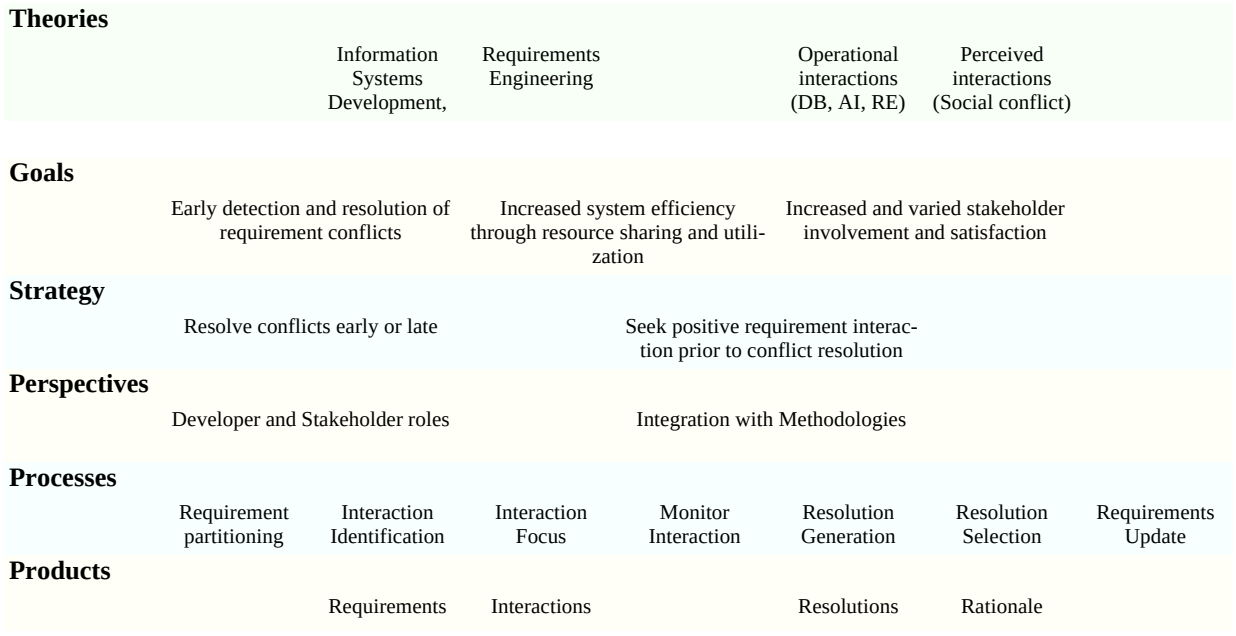


Figure 1. A descriptive framework of requirements interaction management research.

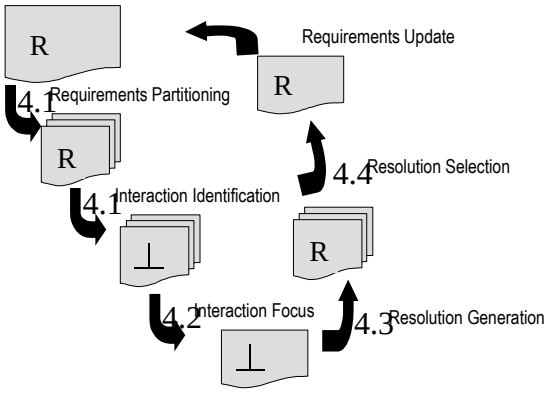


Figure 2. An illustration of the activities which are managed as part of the requirements interaction management life-cycle. Each number indicates the section where an activity is introduced.

ments according to their interactions. The products include intermediate and final results used during the processes.

Figure 2 illustrates the processes of requirements interaction management. It is derived from our model of automated negotiation, which in turn was derived from a survey of tools and theories[230]. The description begins with unstructured requirements which may be divided into partitions. Next, interaction identification may provide conflicts which must be resolved; however, only a subset of interactions will be considered at a time through interaction focusing. Resolution generation provides alternative ways in which each conflict can be resolved. Then, resolution selection determines which will become a change request for the requirements document.

The following section 3 introduces common theories and terms of the related theoretical basis. The details of how specific processes or products have be adapted are deferred until sections 4 - 5. Section 6 reviews specific projects that illustrate common themes or techniques from requirements interaction management. Finally, section 7 presents conclusions.

3 A HISTORICAL PERSPECTIVE OF REQUIREMENTS INTERACTION MANAGEMENT

Requirements interaction management has a narrow “systems” focus on interaction management. However, many of its theories and techniques have been adapted from other disciplines. A description of the evolution of the prominent concepts, and their related disciplines, is presented next.

3.1 Evolution of Concepts Supporting Requirements Interaction Management

Table 1 provides a summary of prominent concepts and their evolution into the emerging discipline of requirements interaction management. The concepts are divided into the five categories of theory, strategy, perspectives, processes, and products.

Many of the theoretical concepts of requirements interaction management theory, such as preference, conflict, negotiation, and resolution, are derived from human negotiation[203] and group decision making[109]. A general overarching tenet of RIM is analogous to that of group decision-making:

Table 1. Evolution of Concepts for Requirements Interaction Management

	Before 1970	1970 s	1980 s	1990 s
Theory		Codified negotiation techniques: "log rolling", condition restructuring[203]	Requirements Negotiation Behavior[12][222] Negotiation experts: case-based[260], rule-based [130]	Domain independent resolution generation[228]
Strategy (Goals)		Programming goals[274] Software qualities[15]	Software development goal structure[20]	Software quality attributes[8]
	Management by Objectives[49]	Software metrics[86]		Software quality interaction experts[20]
	MAUT[205]	MCDM programming[286]	MCDM for requirements[221]	Software quality architecting[125]
			QFD[95]	QFD for requirements[108] Non-functional framework[178]
Perspectives		Multi-view specification[174]	Method Engineering[137] Parallel elaboration[58]	Requirement Viewpoints[187]
			Process programming[195]	Process compliance[56] Interaction monitoring[69]
Products			Requirements Modeling Language[88] Requirements traceability[198]	Goal oriented requirements[42] Agent oriented requirements[177]
Processes		Goal-based design[123]	Goal-based requirements negotiation[220] Program slicing[106]	Goal regression for requirements[215][266]
			Schema integration[11] Inconsistency dialog[76]	Inconsistency reasoning[189] Inconsistency framework[40]
			Multi-agent planning[85]	Agent negotiation[238]

> Specifying stakeholder perspectives on system requirements, followed by their negotiated integration, will result in systems that are both technically better, but are also more accepted by system stakeholders.

In RIM, techniques from human negotiations have been adapted to assist in the detection and resolution of conflicts that arise among requirements. Those domain-specific automated techniques of the 1980's, including case-based and rule-base methods, have been generalized[228] and formalized[266].

A strategy makes the above tenet of group requirements description operational. For the most part, current strategies of requirements interaction management consist of attempting to satisfy system and developmental goals.

Early on, it was recognized that different software development goals: 1) often interact, even conflict, with each other, and 2) developers can directly satisfy an individual goal, but satisfaction of multiple goals is more difficult. In 1974, Weinberg and Schulman demonstrated this experimentally[274]. Teams were given one goal to satisfy, including: minimize effort, minimize lines of code, minimize memory usage, maximize program clarity, and maximize clarity of program output. Each team did best on their given goal—second best in one case. Since then, many software development goals, and their relationships, have been specified[8][9][20][35][125]. Most recently, models and tool support have been created to aid in the analysis of software development goal interactions[19].

Reasoning about requirements goals has evolved concurrent with the evolution of software development goals. Multiple Attribute Utility Theory (MAUT)[205], and later Multiple Criteria Decision Making (MCDM)[221] have provided general decision theoretic techniques that aid in the elicitation of criteria (a.k.a. goals) and the tradeoffs among them during a decision process. The more specialized decision technique of Quality Function Deployment[95] has been applied to requirements analysis (e.g., [108]).

Perspectives, or views, on sets of related requirements have similarly evolved. Approaches, such as CORE[174], ETHICS[173], and later MultiView[6] and Soft Systems[29] combined both social and technical aspects of development in the representation of various views of system requirements. Algorithmic aspects of representing, comparing, and combining various system views have been described in Feather's parallel elaboration work[58], the ViewPoints project[187], and an ever growing body of related research[73].

Requirements products have also evolved. To support reasoning about requirements interactions among requirements views, requirements definition languages has evolved. For example, the Requirements Modeling Language (RML)[88] has given rise to agent[177] and goal[42] oriented requirements languages. Such languages allow for analyses concerning how actions of external agents and system agents affect the satisfaction of system requirements.

Finally, requirements interaction analysis processes have also evolved. Many of the requirements interaction reasoning techniques have been borrowed from related fields. For example, goal regression, an AI planning techniques, can be used to uncover certain requirements that are the root cause of a conflict[266]. Similarly, database schema integration ideas[11] have been adapted to combine requirement views[254]. Likewise, logics and frameworks for reasoning about logical inconsistencies have been derived from non-monotonic reasoning[107].

Interaction analysis of Distributed Artificial Intelligence (DAI) agents share some similarity to the analysis of multiple viewpoint requirements. In the context of a requirements management, each DAI agent represents a requirement viewpoint and the agent knowledge-base represents the requirement viewpoint description. Thus, when DAI agents interact to complete shared tasks, their representation and reasoning is similar to that found in the integration of multiple requirement viewpoints.

DAI agents can cooperative to complete shared plans[85]. If they reach an inconsistent state, they may cooperatively negotiate to satisfy other plans[38]. To do so, they may use economic models to guide their decision making in order to efficiently manage their resources[238]. Such analysis can be applied to integrate multiple requirement viewpoints.

3.2 Disciplines Influencing Interaction Management Research

Researchers on interaction management can be found within a wide variety of disciplines. Most of the research is concerned with identifying and managing negative interactions, or conflicts.

To characterize influences on requirements interaction management, we surveyed over 60 published articles that were referenced by requirements interaction management articles. These referenced works mainly described theories, techniques, and tools for the management of conflicts from disciplines ranging from the computer and information sciences to the cognitive and social sciences. From the articles, we identified seven distinct areas of influence, each of which is further refined. The presentation of disciplines is based on the relative influence of the disciplines on RIM, rather than the overall scope or maturity of the discipline. Thus, the relatively small area of schema integration has as much, if not more, influence as the more established economic theory.

Table 2 summarizes disciplines of influence for research on interaction management. Each discipline is described in the following subsections.

Table 2. Disciplines Influencing Requirements Interaction Management

Software Development • Requirements engineering • Concurrent engineering • Quality architecting • Feature interaction
Database Research • Schema integration • Schema re-engineering
Knowledge Acquisition and Representation • Knowledge integration • Information integration
Artificial Intelligence and Distributed Artificial Intelligence • Reasoning with inconsistency and incompleteness • Distributed problem solving, coordination, collaboration
Negotiation Support Systems • Coordination • Collaboration • Group issues: dominance, anonymity
Social Conflict and Negotiation • Negotiation theory and models • Negotiation strategies and tactics • Bargaining and arbitration • Political negotiation • Social economics
Individual Decision Making • Cognitive dissonance theory • Utility theory

3.2.1 Software Development

Researchers in software development are addressing interaction management in a number of contexts, including:

- *Requirement consistency.* Detection and resolution of requirement inconsistency is a growing theme of requirements engineering—this theme is further expanded throughout this article. As exemplars of this type of research, consider the Viewpoints project (section 6.3) that provides a framework in which rules detect logical inconsistencies among, and within, views of a system requirements specification.
- *Concurrent engineering*. Detection and resolution of design differences among the designs of multifunctional and multidisciplinary teams has been a concern of concurrent engineering[135]. Quality Function Deployment (QFD) is commonly used to identify interactions, from those among system requirements, to those among lower level design or production requirements[135]. However, other methods, such as heuristic conflict classification and resolution, have been used to identify and resolve undesirable design interactions[130].
- *Feature interaction.* Detection and resolution of undesirable functional (feature) interaction is

an established part of telephony software development. For example, consider the conflict between “Caller Number Identification” and “Unlisted Number”. Applying “Caller Number Identification” provides the receiver of a telephone call with the originating telephone number. Conversely, applying “Unlisted Number” prevents the originator from supplying the originating telephone number. Using an AI planning-based scheme of goal hierarchies, it is possible to generate a resolution where the callee receives the caller’s name, but not their telephone number. This resolution resolves the conflict among the telephony features.[268]

- *Quality architecting*. Analysis of how different system architectures affects tradeoffs among system qualities is a concern of software architects[125]. A method, such as ATAM[125], analyzes system qualities, such as performance, security, and reliability as a means to determine if multiple interacting system qualities can be satisfied by a system architecture. If all quality goals cannot be satisfied, the method helps select an architecture that satisfies most qualities.

3.2.2 Database Research

Researchers in database development are addressing interactions as they arise in *schema consistency*. Traditionally, a relational database is designed first with multiple views of data, then the view are combined into a global data schema. As part of the schema integration process, conflicts among the views are identified[11]. Typically, differences in *name* or *structure* are recognized. Generally, this process has been supported with a defined methodologies[11]; however, tool support is growing[82][117][118][206].

3.2.3 Knowledge Acquisition and Representation

Researchers in knowledge acquisition and representation are addressing interactions in a number of contexts, including the following.

- *Knowledge integration*. Knowledge-bases, such as found in expert systems, should be consistent if they are to support logical reasoning. In support of knowledge consistency, knowledge gained from multiple experts must be combined and made consistent within the computerized knowledge-base. Common knowledge integration techniques include the use of metaknowledge, set theoretic analysis, consensus theory, repertory grid analysis, cluster analysis, and decision theory[24]. For example, a tool have been developed based on repertory grid and personal construct theory aimed at supporting the derivation of terminological consistency among experts[246][247].
- *Information integration*. A knowledge-base system, during its execution, may receive a variety of inconsistent inputs. To solve its overall task, the system must appropriately deal with these inconsistencies[96][140]. For example, in a meeting scheduling system, a single person may be referenced in a number of schedule databases using slightly different names; an information integration agent can reconcile this by recognizing naming differences[259].

3.2.4 Artificial Intelligence

Researchers in artificial intelligence (AI) and distributed artificial intelligence (DAI) are addressing interactions in a number of contexts, including the following.

- *Planning*. AI planners have had to deal with the conflict between goal satisfaction and the availability of resources to satisfy a goal. Traditionally, a plan for goal satisfaction has been constructed through goal decomposition—the refinement of abstract goals into disjunctions of conjunctions of more specific subgoals. However, a planner may not be able to construct a plan for a particular subgoal. To overcome such an obstacle, a planner would traditionally reconsider higher-level decompositions and then replace the failed goal decomposition with an alter-

native goal decomposition[276].

- *Distributed agent negotiation*. Distributed artificial intelligence has expanded the role of planning. Multi-agent planning systems identify and resolve plan failures that occur among sets of loosely coordinated agents (e.g., [38][50][85][134][270]). Again, the method of resolution is typically subgoal replanning; however, the subgoal failure and replanning is complicated in the absence of global information.
- *Negotiating agen*ts. Researchers in AI, and subsequently DAI, have defined negotiating agents. Two complimentary research areas have arisen:
    - *Negotiation analysis knowledge*. Case-based reasoning and rule-base programming, have been used to codify and apply analyses for identifying and resolving conflicts that arise in a variety of domains, including labor negotiation[261], design integration[130], and specification integration[229].
    - *Negotiation protocol knowledge*. Frameworks[39][250] and communication protocols[133][141][165][191][238] have been defined for the purpose of coordinating the sequences of messages among distributed negotiating agents.

3.2.5 Negotiation Support Systems

Researchers in negotiation support systems (NSS) are addressing interactions in the context of advising a human negotiator or supporting humans gathered around a negotiation table[116][148]. Research ranges from developing a NSS "shell" aimed at supporting the construction of negotiation systems[127][164], to specialized domain support such as airline buyout[243], product marketing[210], or electronic marketplace[281]. Some negotiation support systems have the intelligent reasoning features of artificial intelligence. Negotiation support systems often focus on human aspects of negotiation including dominance and anonymity of participants.

3.2.6 Social Conflict and Negotiation

Much of the above research has been influenced by theories and studies of conflict and negotiation among humans. Such knowledge provides a background in persuasion[79], negotiation[203][204], and decision making[109][205] from which computerized models can be derived.

3.2.7 Individual Decision Making

The general discipline of individual decision making has influenced the basic theories of interaction management research. Multi-attribute utility theory suggest how an individual can trade-off various interacting goal to maximize their overall utility[205]. Some decision models consider the dynamic aspects of this process. For example, as individual learns of the trade-offs among goals, that individual may reconsider the value, or weight, they place on individual goals[286].

4 PROCESSES OF REQUIREMENT INTERACTION MANAGEMENT

The overall process of requirements interaction management may be subdivided into six activities, as illustrated previously in figure 2. In the initial process, unstructured requirements may be divided into partitions. Next, interaction identification may provide conflicts that must be resolved; however, a subset of interactions will be considered at a time through interaction focusing. Resolution generation provides alternative ways in which each conflict can be resolved, then resolution selection determines which will become a change request for the requirements. Research issues arising from each of these activities are presented in subsequent subsections (4.1 through 4.6).

4.1 Requirements Partitioning

Given a document with a great many requirements, requirements partitioning seeks to focus interaction analysis on manageable requirement subsets. This is important, as analysis of all interactions among all requirements can involve significant computation; for n requirements, there are $n(n\text{-}1)/2$ binary comparisons. Partitioning seeks to divide this problem into a set (or tree) or smaller problems. Requirement partitioning gives rise to the following issues.

I 1. By what measure should a requirements document be partitioned into subsets which enhance analysis?

Problem partitioning, based on subgoals or function decomposition, has been considered by most computer science communities (e.g., distributed artificial intelligence [83]). In composite system design, the cross-product of function partitioning and agent responsibility has used to partition requirements[58]. Similarly, one could apply the concept of the machine-environment boundary for partitioning[284]. Other requirements attributes can be used. For example, analysts can attribute requirements according to a predefined classification of common non-functional attributes of software. Then, one can restrict comparisons to requirements with similar quality attributes[19]. Partitioning based on stakeholder perspectives (or views) is a commonly used natural partitioning based on the originating source of the stated requirements[29][87]. Other partitions been based on scenarios[201], and the use of *requirement subsumption* to derive *root requirements*[225]. Finally, partitions have been based on some *a priori* analysis (e.g., issues [279]). For example, Easterbrook has defined partitions based on consistency—a type of truth maintenance system for requirements; however, since it is based on requirements consistency (a type of interaction), the partition is the *result* of interaction analysis and not the *input* to it[51].

I 2. How can an analyst select requirements with certain characteristics?

Given some characterization of each partition, database and keyword search technology has been used to partition the requirements. For example, commercial tools can apply database technology to select subsets of requirements based on requirement attributes. Difficulties arise when the requirements are not attributed *a priori* with necessary characteristics. In such cases, requirements can be partitioned based on keywords found in each requirement. However, the presence of a keyword does not necessarily indicate that the key characteristic is present in the requirement. Some of such limitations of keyword retrieval have been overcome by concept-based information retrieval[30][31]. Sometimes requirement subsets are constructed with different terminology, that can further confound partitioning; for example, where requirements are developed by different people. In such cases, one can fall back on statistical measures of usage to generate mappings among terms[247].

I 3. Given a partitioning of requirements, how can analysis of the partitions be ordered to enhance analysis?

Strategies for analyzing a set of requirements partitions are rare. One can apply a general software life-cycle strategy; for example, the Spiral Model would consider the riskiest partitions first[21].

4.2 Interaction Identification

Given a set of requirements, requirements interaction identification seeks to determine requirements which are mutually incompatible in that they imply systemic failures in the resulting operational system. A search for such interactions gives rise to the following issues.

I 4. What are the kinds of requirements interactions that occur?

A variety of fields have contributed to a growing ontology of interaction types. Artificial intelligence planning concepts often serve as the basis, including types such as: goal/subgoal and goal conflict. Goal conflict itself is explained in terms of conditions, or resources, of operators that attempt to achieve a goal; for example, goals may conflict because operators, which satisfy the goals individually, have interfering preconditions when their simultaneous achievement is sought. Similarly, two requirements may conflict because they mutually deplete the available resources of an operational system. Many of such planning terms have also been applied to characterize interactions among requirements[202][220][228].

A simple ontology of supports/detracts (i.e., +/-), that simply summarizes the details on *how* requirements interact, has been found to be a practical solution[19][35][42][207][279][225]. Such work has been extended to incorporate fuzzy logic concerning the degree of conflict[282]. More recently, fault-trees have been considered as the basis for an ontology of interactions types. A fault tree, such as that concerning human-computer interaction requirements[157], can be considered an *a priori* enumeration of common operational failures that occur among human-computer interfaces. In a similar fashion, a classification of common interactions among non-functional attributes of software development has been used to annotate the ways in which requirements conflict[19]. Finally, relationships other than conflict have been considered, such as the cost/benefit of requirements[124].

4.2.1 Uses of the Term Conflict

Table 3 summarizes the most general types of interactions found in the requirements engineering related literature. As would be expected, they are divided into correlation types of positive, negative, unspecified, and independent.

Table 3. Types of Requirement Relationships

Type	Description	Example
Positive correlation	Increasing the satisfaction of $\mathbf{R}_1$ increases the satisfaction of $\mathbf{R}_2$.	Some, +, ++,[35] Influence +[92]
Negative correlation	Increasing the satisfaction of $\mathbf{R}_1$ decreases the satisfaction of $\mathbf{R}_2$.	Hurts, -, --, [35] Contradictory Influence -[92]
Unspecified correlation	Changing the satisfaction of $\mathbf{R}_1$ has an unspecified effect on the satisfaction of $\mathbf{R}_2$.	Impacts on Interdependent
No correlation	Increasing the satisfaction of $\mathbf{R}_1$ has no effect on the satisfaction of $\mathbf{R}_2$.	Neutral

A more refined analysis of the literature reveals the bases of most interaction relationships. These include interactions over structure, resources, task, casualty, and time. Table 4 summarizes these more refined interaction types.

Table 4. Basis of Requirement Relationships

Type	Description	Example
Structure	$\mathbf{R}_1$ is similar to $\mathbf{R}_2$.	Duplicate, Alternative.
Resource	$\mathbf{R}_1$ and $\mathbf{R}_2$ depend on the same resource.	Resource utilization/ contention
Task	$\mathbf{R}_1$ describes a dependent task of $\mathbf{R}_2$.	Subtask, Means/Ends, Operationalized by, Pre/ Post condition
Causality	$\mathbf{R}_1$ describes a consequence of $\mathbf{R}_2$.	Results in
Temporal	$\mathbf{R}_1$ has a temporal relation to $\mathbf{R}_2$.	Coincident state, simultaneity constraint, pre/ post time relation

The interaction types of tables 3 and 4 were derived from our survey of conflicts that are considered in related literature. A the total of 46 different conflict terms were encountered. These 46 terms were reduced to eighteen different conflict types. These eighteen types were from two broad categories: syntactic conflicts and semantic conflicts. Syntactic conflicts are those caused by terminology inaccuracies or improper grammar. Semantic conflicts concern the meaning of the concepts; they were divided further into four subcategories. Each sub-category contains a set of related conflicts that were found in the literature. Table 5 describes these interaction types.

I 5. What kinds of analyses can be applied to requirements to uncover requirements interactions?

Techniques for comparing requirements and classifying their interaction type have been automated. If one has operational requirements, then program slicing can be used to show semantic differences in versions of a common root specification[106][280]. If the requirements are only represented as non-operational systems goals, then one can apply planning techniques to derive a plan for the conjunction of the requirements set: 1) if a plan can be found from a given operator set, then the requirements *can* be achieved simultaneously, 2) if the planner fails to find a plan, one can use goal regression to find the reason for the requirements conflict[4][215][263]. Finally, it may be the case that a set of requirements can be achieved and can fail within the same environment. One can check for this by planning for the conjunction of some requirements with the negation of others; if such a plan succeeds, then the requirements can fail as demonstrated by the plan[4]. Of course, such analysis is only possible if some portion of the system environment has been formalized in terms of operations and other environmental resources[4]. Such a planning approach to goal interaction has also been applied to scenario analysis: requirements become plan goals and the operators of the plan becomes the actions of the scenario. Scenarios can be generated by considering a variety of plan failures (e.g., precondition failure via resource depletion)[4][157]. Such interaction analysis has also been carried out in the distributed artificial intelligence[83]; for example, [268][270].

It is not always feasible to rely on a formalized model of the environment to check plan-based requirements interactions. Sometimes, one only has the semantics inherent in a structured requirements document. In such cases, one can use *a priori* knowledge of commonly occurring requirements elements, called an ontology. Such ontological information specifies commonly occurring entities and their relationships[81][90][192][255]. Using such an ontology, interaction analysis has two stages: 1) classify the requirements as instances of the ontology, 2) compare the classified requirements. Such analysis has been automated[253].

Table 5. Uses of the Term Conflict from the Literature

Interaction Type			Description
Syntax Conflicts	***Synonyms***		Multiple terms refer to a single concept.
	Homonyms		A single term refers to multiple concepts.
	Mistake		Conflict caused by a syntax error which is readily recognized as such.
Semantic Conflicts	***Perception***	*goal*	High-level goals of agents conflict.
		assumption	Agents' assumptions conflict.
		domain boundary	Agents perceive the boundary of the decision-making domain differently. This leads to missing essential concepts for one agent, and irrelevant present concepts for another.
		cognitive conflict	Agents judge issues according to different criteria, or assign different weights to the same criteria.
	Communication	*abstraction level*	Agents positions cannot be compared or matched because of different levels of abstraction.
		accuracy	Agent's positions have different levels of precision.
		circular justification	Agent's justification appears circular to another agent.
	Resource	*quantity*	Resource limit is exceeded, or there is a mismatch between the desired resource level and provided level.
		quality	Resource quality attributes are not met.
		availability	Resource is not available. Conflict varies with consumable/non-consumable resources, as well as with divisible/indivisible resource. Divisible resources can be shared, and non-consumable resources can be re-used.
		redundancy	Multiple resources satisfy the same goal.
	Behavior	*deadlock*	Conflict stalls a process.
		deadline	Deadline is violated or cannot be agreed on.
		commitment	Agents disagree on the commitment level (from breakable to binding), or a commitment is violated.
		normative conflict	An agent does not behave in expected or prescribed way. For example, court decisions should be predictable based on the past judgments of similar cases.

Even with the manual attribution of a limited interaction ontology to requirements interactions, significant benefits can be gained. For example, an efficient resolution strategy can be automated based on analyses of a simple five term interaction ontology[225].

4.2.2 Conflict Detection Methods

As indicated above, requirements have been analyzed for their interactions using a variety methods. Table 6 summarizes five categories of methods. These methods are described in the next subsections.

Table 6. Interaction Analysis Methods

Method	Description	Example
Domain Model	Requirement interactions are found and classified by comparing requirements against an *a prior* model of requirement interactions.	WinWin[19], CDE[130]
Theorem proving	Requirement interactions are found by proving assertions about requirements.	SCR[101], RSML[147], PVS[256]
Scenario Analysis	Requirement interactions are demonstrated by simulating a sequence of events that represents a narrow aspect of a system's required behavior.	SCR[101], Spin[104], CREWS-SAVRE[159][257]
Modeling checking	Requirement interactions are found by exhaustively searching a state-based model of requirements for specified properties.	Murphi[47], SMV[168], Spin[104]
Executing monitoring	Requirement interactions are found by monitoring a system implementation for certain events that indicate specified requirement properties.	FLEA[57][62]

4.2.2.1 Domain Model of Binary Interactions

To identify interactions at the requirement level, a domain model of system requirement interactions is necessary. The domain model captures commonly occurring requirement interactions, such as Completeness Increases Effort; this is illustrated in figure 5 as a thick link between the Completeness and Effort classes.[7] A hierarchy of such *binary requirement interactions* typically defines the domain model. Most work on quality interactions uses such binary relationships to indicate interactions (cf. [19][35][42][207][225]).

Using a domain model of requirement interactions, interaction analysis has two phases: 1) classify requirements as instances of the domain model, and 2) instantiate the model level interactions to the requirements. For example, if the domain model indicates that Completeness Increases Effort, and Completeness and Effort are qualities of two requirements, then there is an Increases interaction between their Completeness and Effort. This process can be more formally described as follows:

InferQualityInteractions (R1,R2) ≡
∀ R1,R2 : REQUIREMENT, ∃ QC1,QC2 : QUALITYCLASS, ∃ Q1,Q2 : QUALITY, ∃ INT1 : INTERACTION,
• (R1 QUALITY Q1) ∧ (Q1 ∈ QC1) ∧ (QC1 INT1 QC2) ∧ (Q2 ∈ QC2) ∧ (R2 QUALITY Q2)
⇒ ∃ INT2 : INTERACTION (Q1 INT2 Q2)

The WinWin tool is probably the most well know requirements tool that uses a domain model of binary interaction relationships to notify stakeholders of new requirement interactions[19][54]. (Section 6 summarizes the WinWin project, among others.)

4.2.2.2 Domain Model Interaction Patterns

A domain model of binary interactions can lead to inconsistent inferences about interactions. Reconsider the impact of complete and accurate information (ScheduleInformationCompleteAccurate) on participant scheduling effort (SchedulerMinimizesParticipantEffort). In general, it may be that Completeness Increases Effort. However, it may also be that Automation Decreases Effort. For example, computerized selection and scheduling of a meeting room can both: 1) *decrease* the effort of meeting participants, and 2) *increase* schedule information completeness. However, increased completeness implies increased participant effort, which is not consistent with the *a priori* interaction,

[7] Such links in figure 5 on page 33 are interaction summaries; the complete ontology uses the Interaction class to model all interactions, both at the quality class level and at the requirement instance level.

Completeness Increases Effort. Thus, a domain model of binary interactions is simple to construct, but its lack of precision can lead to inconsistencies.

A more precise domain model can be created by using interactions patterns. Such patterns provide context that can be used to augment more generic quality relationships. As an example, consider the following (simplified) interaction pattern.

> For all instances of ACTIVITY1, where POSITION1 has a role in ACTIVITY1
> and AGENT1 fills POSITION1 and AGENT1 isA COMPUTER and ACTIVITY1 has a post-condition of INFORMATION and
> AGENT2 fills POSITION2 and AGENT2 isA HUMAN and POSITION2 has a role in ACTIVITY2
> and ACTIVITY2 has a pre-condition of INFORMATION
> POSITION1's ACTIVITY1 decreases POSITION2 ACTIVITY2's Effort.

Informally, the pattern indicates that an activity performed by a computer agent that produces information used by a human agent decreases the human agent's effort. Of course, a multiplicity of patterns leads to the same problem of inconsistency as found in domain models of binary interactions. However, the precision of the patterns reduces the problem.

The specification tool, Critic[71], and the concurrent requirements analysis tool, CDE[130], used interactions patterns to detect conflicts. (Section 6.2 summarizes a project that uses domain independent patterns to classify requirement interactions.)

4.2.2.3 Proof Checking

Correctness and completeness can be verified in requirement specifications described as deterministic state machines. Two popular tools, SCR[101] and RSML[147], demonstrate how analytic techniques can proof safety and liveness properties for such specifications. For example, SCR (§6.6) has been used to uncover inconsistencies between requirements[13]. Heitmeyer and Mandroili present an overview of the current state-of-the-art in formal modeling and analysis tools for software specifications[98].

4.2.2.4 Scenario Analysis

As defined in[267]

> A *scenario* is ... a temporal sequence of interaction events among different agents in the restricted context of achieving some implicit purpose(s). ... A scenario captures just one particular, fragmentary instance of behavior of a system.

Since scenarios are representative execution fragments of a system, their outcome may be evaluated relative to requirements. *Positive* scenarios satisfy requirements, while *negative* scenarios illustrate the violation of requirements. A scenario that is both positive and negative can illustrate a negative interaction among requirements.

Scenario analysis can be used to detect requirements interactions. The user may select a subset of requirements to be analyzed. Then, a particular scenario can be executed to determine if the selected requirements are satisfied. If the requirements are not satisfied, then a undesirable interaction has been discovered. For temporal logic requirements and state-based scenarios, such analysis can be automated using model checking (e.g., Spin[104]). Alternatively, a knowledge-based approach can suggest scenarios that are likely to generate requirement interactions[159].

4.2.2.5 Model Checking

Model checking provides an intermediate level of analysis between monitoring an actual system execution and proving general system properties. Model checking is an operational exploration of state-based models. Thus, it is amenable to the analysis of state-based requirements[161]. Such

analyses can prove that specified logic conditions will, or will not, occur in a modeled state of a system satisfying the requirements.

Modeling checking of requirements is typically applied as follows: 1) a portion of the requirements specification is translated into a formal automata-based model, 2) important requirements properties, such as liveness, are defined as logical properties of the model, 3) a model checker (e.g., Spin[104]) is used to exhaustively check all states of the model for violations of the specified properties.

Model checking has been used to verify functional requirements; specifically safety, precedence, or liveness requirements. It has found missing, ambiguous, and erroneous requirements[13][97][251]. Moreover, this work can be applied to analysis of implementations. By instrumenting an implementation to log interesting state changes, a model checker can check the resulting log files to verify properties of the implementation[28].

4.2.2.6 Execution Monitoring

Requirement level descriptions can be directly analyzed for static properties; however, to analyze some dynamic properties, the described system behavior can be execute or simulated.

Execution monitoring of requirements is a technique that tracks the run-time behavior of a system and notes when it deviates from its design-time specification. Requirements monitoring is useful when it is too difficult to prove system properties. To aid analysis, assumptions are made as part of the requirements definition process. These assumptions are monitored at run-time. Should the assumptions fail, a predefined procedure is invoked (e.g., notification to the designer). Note, such monitoring is different from exception handling in that it: (1) considers the combined behavior of events occurring in multiple threads or processes, (2) links run-time behavior with the actual design-time requirements, and (3) provides sufficient information to allow for the run-time reconfiguration of software or software components.

Fickas and Feather proposed requirements monitoring to track the achievement of requirements during system execution as part of an architecture to allow the dynamic reconfiguration of component software[69]. Feather has produced a working system, called FLEA, that allows one to monitor events defined in a requirements monitoring language[57][62]. FLEA captures interesting events as assertions in a database. (An external system inserts the assertions into the AP5 database.) When a monitored condition occurs, its defined action is executed. Thus, monitoring mainly consists of the translation of requirements monitoring descriptions to database triggered actions.

Fickas and Feather illustrate the execution monitoring of requirements in the context of monitoring the requirements of a software license server. When the license server fails to satisfy its requirements (e.g., a user shall be granted a license in 90% of their requests), due to a change in the system environment, the system notifies an administrator. Emmerich *et. al.* (and others [226]) have since illustrated how the technique may be used to monitor process compliance[56]; for example, organizational compliance to ISO 9000 or IEEE process descriptions[166].

4.3 Interaction Focus

I 6. Given a number of requirements interactions, how can the interactions be partitioned to enhance the analysis?

Some requirements interactions depend on other requirements interactions. For example, the resolution of one conflict may introduce new conflicts into the requirements set; conversely, one resolution may remove multiple conflicts.

Efficient resolution seeks to focus on key interactions. There are a number of measures by which interactions can be partitioned or ordered for consideration by the resolution generation process. In general, one seeks to decrease the overall conflict and to minimize rework. To satisfy these goals, Robinson has ordered requirements by their degree of *contentiousness*—the percentage of conflicting interactions that a particular requirement has among all requirements[225]. Next, one always focuses resolution generation on conflicts which involve requirements with the greatest total contention. Robinson has shown this to be a strategy that monotonically decreases the overall conflict in a small requirements document[225]. However, this strategy only considers conflict dependencies. Other factors, such as requirements importance, are not considered. It may be the case that a less contentious requirement should be considered first because it *must be* achieved *exactly* as stated (i.e., there is no room for negotiation). Such a case would lead to the strategy of always considering conflicts that involve requirements with the greatest total importance. However, the *conflict context* of low contention requirements is small—there are only a few other requirements to be considered. Thus, resolutions of low contention requirements may be myopic, and result in the introduction of new requirements which further exacerbate other conflicts. For example, in a limited resource environment, assigning all resources to important requirements solves a few important conflicts, but can introduce many other conflicts. Thus, one may need to reconsider the importance of a requirement in the face of trade-offs among many other requirements. Research into cost-value trade-offs of requirements can assist this process[124].

Requirements interaction can also be partitioned using the same means to partition a large set of requirements. (See section 4.1.) Thus, one could consider all interactions found in a partition of a: stakeholder, scenario, requirement subsumption hierarchy, etc. In fact, decision science suggests that individuals can maximize their own benefit by first understanding and specifying their own preferences prior to negotiating with others[286]. This suggests that one use stakeholder partitioning to resolves intra-partition conflicts prior to inter-partition conflicts. Moreover, negotiation literature suggests that, in social contexts, the simplest conflicts be resolved first, thereby building trust among the negotiating participants[203].

4.4 Resolution Generation

I 7. Given requirements interactions, how can resolutions be generated?

Techniques for conflict resolution have been automated[230]. Conflict resolution can be characterized as multiple goal planning problem: given goal sets G_1 and G_2 held by agents A_1 and A_2, respectively, the resolution process attempts to find a combined goal set similar to $\{G_1, G_2\}$ that can be achieved without conflict. Resolution is commonly characterized as a three-tuple: (agents, goals, environment), where multiple agents seek to achieve goals within an environment—the environment specifies available operators, resources, and other constraints of the domain. A classic value-oriented approach to conflict resolution considers alternative goals in order to find a non-conflicting substitute goal set. If the substitute goals are ordered, then the process of considering lesser desired goals is called *lexicographical ordering*[286]. This approach characterizes resolution generation as a constraint relaxation problem[39][275]. If the goals are arranged in an AND/OR hierarchy, one can apply replanning to generate alternative goal values[2][268][270].

In addition to altering goal values, one can consider the environment—specifically, the operators, resources, and other constraints of the domain. Using such a structure-oriented approach, leads to the consideration of new operators and resources, as well as distributing resource usage over time (i.e., sharing), and other interesting resolutions. This approach, sometimes referred to as

"lateral thinking" or "out of the box thinking"[79], is considered more likely to lead to an optimal resolution, than the value-oriented approach[127][203]. Thus, it is not surprising that a number of knowledge-based agents use some form of problem restructuring to generate resolutions[127][130][164][260][261]. [8]

The structure-oriented approach has be implemented by matching new conflicts with a domain-dependent case-base containing associations of previous conflicts and their resolutions[130][260][261]. New resolutions can be derived from previous resolutions of similar cases. Another implementation approach encodes structure-oriented resolution knowledge into a domain-dependent rule-based system[127][164]. Such restructuring transformations can be generalized using basic negotiation principles. A theory-based, rather than domain-based, approach overcomes problems attributed to the narrow expertise of expert systems[27]. Negotiation theory-based domain-independent transformations apply across application domains and still apply when faced with new, unforeseen circumstances. This is the approach has been recently explored[218][228][266]. (For example, see section 6.4.)

4.4.1 Conflict Resolution Methods

Table 7 summarizes conflict resolution methods found in the requirements engineering related literature. The six categories were derived from approximately 29 methods mentioned in the literature, of which approximately 11 unique methods were identified. (Section 6 summarizes projects that use these methods; see table 9 in section 6.)

4.5 Resolution Selection

Resolution generation can present a great many ways to remove a requirements conflict. Resolution selection seeks to determine how one can select an appropriate resolution, as well as incorporate the selection measures into an efficient resolution generation technique.

I 8. Given requirements resolutions, how can one select the best resolutions?

Decision science theories suggest how one can select the best alternative from a set of alternatives. The classic approach is based on *utility*—the benefit one derives from an alternative[205]. An overall utility can be decomposed into multiple criteria, to form a multiple criteria utility function[286].[9] If requirements are attributed with scaled non-functional attributes, then stakeholders can use the attributes to specify that they seek to maximize, minimize, or reach specific attribute values. Such stakeholder multiple criteria utility functions can be used to derive a preference ordering among requirements or conflict resolutions. Robinson has demonstrated the value of such an approach in two different research projects[215][221][218]. (A specialization of this general approach, based the House of Quality methodology, has also been demonstrated[108].) In addition, Robinson has demonstrated the value of Zeleny's Interactive Decision Evolution Aid (IDEA[286])—an interactive decision procedure which provides feedback on criteria trade-offs among decision alternatives[215][221][218]. Using the procedure, an analyst does not explicitly

[8] Many distributed artificial intelligence projects use the term "negotiation" to describe their work. Typically, the aspect of negotiation explored concerns explicit[38][39][50][133][165][235][237][242][250] or implicit[232][233][234] communication protocols. Such communication often involves the collaborative sharing of local aspects of a global problem. However, strategic communication in competitive environments has been explored[191][235]. Nevertheless, the generation of resolutions to conflict typically involves some form of compromise, goal relaxation, or goal dropping[1][122][136][172][240][275]. (The type of resolution outcome can be characterized independently of the bargaining architecture which produces it[211].) In general, the main focus of these projects is not on defining new resolution generation techniques, but on incorporating resolution into a distributed artificial intelligence architecture.

[9] An *attribute* of a requirement is called a *criteria* in decision science.

Table 7. Conflict Resolution Methods

Method	Description
Relaxation *generalization* *value-range extension*	Conflicting requirements are relaxed to expand the range of mutually-satisfactory options beyond what the original requirements specify. Generalization involves replacing the conflicting concept with a more general concept. Value-range extension changes the reservation values (prices) of the stakeholders.
Refinement *specialization*	Conflicting requirements are partially satisfied. Assumes that the requirements can be decomposed (refined) into specialized sub-requirements, some of which can be satisfied.
Compromise	Given a conflict over a value that exists within a domain of values, compromise finds another substitute value from that domain.
Restructuring *related resource* *related requirement* *distribution*	Restructuring refers to a set of methods that attempt to change the conflict context; they may alter contextual objects in addition to the conflicting requirements. Restructuring (modifying) resources or requirements that affect the conflicting requirements can reduce constraining relationships and allow a wider range of resolution options. Distribution predicates requirement satisfaction on time, duration, or other contextual resources to allow contextual requirement satisfaction.
Failure Recovery *re-enforcement* *re-planning*	Failure recovery attempts to restructure the context and to avoid the conflict entirely. Re-enforcement refers to restructuring the precondition of the conflict. For example, a non-response conflict can be avoided by sending multiple notifications to the agent. Re-planing refers to choosing an alternative set of requirements in order to achieve a subordinate requirement (a.k.a., goal).
Other *postponement* *abandonment*	Conflict resolution can be postponed. In complex interactions, many conflicts and requirements are interrelated. By resolving postponing a conflict and resolving other conflicts, the postponed conflict might cease to exist. Alternatively, conflicting requirements can be abandoned.

specify trade-offs between criteria. Instead, the analyst simply seeks to improve values of the current solution along specific criteria. As part of the search for a good resolution, resolution generation is invoked to create new alternatives. As this process continues, the analyst will focuses on a succeedingly narrower solution set. As such, solution optimally is subjectively determined by the analyst[63][109]. The result can be interpreted as a settlement of trade-offs among stakeholder positions[221][286].

I 9. Can resolution selection be integrated into resolution generation?

Given knowledge of which resolutions are preferred, it would be efficient to include such information into the resolution generation process so that resolution search is limited to the most promising resolutions. A match-based approach provides for such narrowed search. For example, a case-base approach can select resolutions based on similar conflict contexts which include similar preferences[130][260][261]. Similarly, a neural network can learn strategies for generating good resolutions[191]. However, explanation of the reasoning that lead to a specific resolution can be difficult under these match-based approaches. Alternatively, the transformations in a transformation-based approach can provide textual explanations of the reasoning[183] and formal analyses of the transformations applied (e.g., refinement[43]). Moreover, given selection preferences, one can incorporate those preferences into preconditions of transformation to make generation more efficient[172].

4.6 Requirements Interaction Management

The above five subsections are activities that are part of the identification and resolution of requirements conflicts. These activities themselves raise methodological issues concerning the context of their use.

I 10.When should the activities (requirements selection through resolution selection) take place?

Methodological guidance as to when and why one should engage in requirements management activities is rare. Traditional approaches, such as the classic software life-cycle, suggest that interactions should be checked and resolved after any substantial change to a document—especially, after each phase in the software life-cycle[23]. However, some methodologies explicitly seek independent, and possibly inconsistent, partitions [29][174][187][228][241]. Nuseibeh summarizes ways in which various software methodologies address conflicts in descriptions, including: ignoring, circumventing, removing, and ameliorating [186]. He goes on to suggest metrics which should be tracked as part of inconsistency management, including: likelihood of failures due to unresolved conflict, and dependent decisions which may be effected by the status of a conflict. Additionally, it is suggested that non-intrusive "reminders" aid in tracking the status of conflicts. Unfortunately, if one is to tolerate conflicts for a period of time, the circumstances as to when conflict detection and resolution should be applied remains largely unexplored.

I 11.How can automated support for multiple analysts, engaged in the interaction management activities, be provided?

A number of projects address the management of requirements interactions within a multiple analyst environment. Chen and Nunamaker have proposed a collaborative CASE environment, tailoring GroupSystems decision room software, to facilitate requirements development[32]. Using C-CASE, one can track and develop requirements consensus. Potts e*t. al.*, have defined the Inquiry Cycle Model of development to instill some order into analyst dialogs concerning requirements interactions—specifically, interactions that arise as part of scenario analysis[201]. Requirements are developed in response to discussions consisting of questions, answers, and assumptions. By tracking these types of dialog elements (and their refinements), dialog is maintained, but inconsistency, ambiguity, and incompleteness are kept in check through specific development operations and requirements analysis (e.g., scenario analysis). The ViewPoints project has stimulated substantial research into the management of multiple requirements representations[73][77][52][174]. At its core, is the representation of multiple development documents which can use different languages (e.g., dataflow diagrams, petri nets), as well as different stakeholder perspectives (e.g., Manger, Employee). The WinWin tool provides groupware support for tracking team development of requirements, including conflict detection and resolution[19][54]. In addition to issue tracking, the tool aids conflict characterization with its hierarchy of common requirements conflict criteria. There are still other collaborative CASE efforts which use metamodels to aid analysis across stakeholder perspectives[181][207][93].

In addition to direct support of the analysis *of the requirements*, a number of projects are indirectly supporting such analysis by giving analyst tools to aid their dialogs *about the requirements*. Basic problems of collaborative CASE include: information control, sharing, and monitoring[269]. Collaboration problems include supporting: task, team, and group level analysis[269]. Collaborative tools, such as electronic white-boards and video conferences can capture the dialog surrounding analysis. Such rationale can be linked to the requirements dialog in a manner similar

to linking source documents to specific requirements[34]. Such collaborative tools can be adapted to facilitate conflict resolution, as well as capture rationale for selected resolutions.

I 12.How can the status of the activities be monitored: through development? through system operation?

Most collaborative case tools support the tracking of annotations associated with documents. As such, one can update a document's annotated status as it passes from one activity to another. Fewer tools support the explicit specification, achievement, and tracking of methodology goals. For example, consider the goal of having all requirements have a defined user priority.

$$\text{HavePriority} \equiv \forall\, R \in \text{Requirement}, \exists\, P \in \text{UserPriority} \bullet (\text{HasPriority } R\ P)$$

(This could be used to support standard PSS05, which specifies that under incremental development, all requirements will have a user defined priority[166].) It is desirable to support analysts in their specification, achievement and tracking of such goals. Emmerich *et. al.* has illustrated how a tool can monitor the violation of such methodology goals as part of a process compliance checking technique[56]. Their work, is built upon Fickas and Feather's requirements monitoring concept[69]. (See section 4.2.2.6.)

Outside of requirements engineering, workflow and process modeling provide some solutions for the management of requirements development[248]. It is possible, for example, to generate a work environment from a hierarchical multi-agent process specification[170]. There has been some attempt to incorporate such process models into CASE tools[169]. However, these tools generally aid process enactment, through constraint enforcement. As Leo Osterweil notes:

> Experience in studying actual processes, and in attempting to define them, has convinced us that much of the sequencing of tasks in processes consists of reactions to contingencies, both foreseen and unexpected.[194]

In response, new research tends to downplay process enforcement and supports the expression and monitoring of process goals[56][226].

5 PRODUCTS OF REQUIREMENT INTERACTION MANAGEMENT

The products of requirements interaction management are largely descriptions of requirements and their interactions. Defining a language for such descriptions is a major research effort. In this section, we present a single requirement ontology that represents an amalgamation of ideas from projects aimed at addressing requirement interaction management through language. (Section 6 summarizes projects that define such ontologies; see for example, KAOS or *i** .)

An ontology is needed to formally define requirements. To support the interaction analyses, the ontology should allow for the description of functional and non-functional requirements, as well as other aspects introduced in section 2.1. In addition to the descriptions of requirements, it is important to capture domain definitions. For example, it may be known *a priori* that satisfaction of a particular requirement increases network bandwidth. An important aspect of interaction analysis involves the application of such *a priori* domain knowledge to uncover requirement interactions. (See section 4.2.2.1.)

5.1 Requirements for a Distributed Meeting Scheduler

We introduce an sketch of Requirements for a Distributed Meeting Scheduler as a means by which to illustrate a requirement ontology and requirement interaction analysis. The meeting scheduler requirements were chosen because of: (1) the complex multi-stakeholder interactions (e.g., pri-

vacy, responsibility, efficiency), (2) the availability of a widely circulated compact, yet rich, requirements document[265], and (3) the publication of prior analyses of the case[201][264].

The general problem of the meeting scheduler can be summarized by the introduction to the requirements[265]

> The purpose of a *meeting scheduler* is to support the organization of meetings—that is, to determine, for each meeting request, a meeting *date* and *location* so that most of the intended participants will effectively participate. The meeting date and location should thus be as convenient as possible to all participants. Information about the meeting should also be made available as early as possible to all potential participants. ...

The remaining requirements of the four-page baseline description refine the roles of the meeting scheduler and participants. However, this introduction will be sufficient to understand the examples that follow.

5.2 A Simple Ontology for Requirement Interaction

Figure 3 illustrates a simple requirement ontology that provides for the expression of requirements and their interactions.[10] Most objects are classified as resources. Resource subclasses include agent, position, and activity; other subclasses, such as organization and information are not shown. An activity is carried out by an agent who fills a position within an organization. A position may have associated roles in which an agent does activities. Resources may be defined through their attributes, constraints, and qualities. We call such a requirement ontology, a *agent-based requirement ontology* because it expresses requirements as goal states to be achieved by system agents. It is derived from a synthesis of agent-based requirements ontologies, including the widely applied ontologies of REMAP[207], KAOS[42] and *i** ("distributed intentionality")[177].

The requirement ontology can be used to express requirements of the distributed meeting scheduler. Requirements are expressed in terms of constraints on resources. For example, consider the following requirement.

```
Requirement InitiatorKnowsConstraints with
Mode
   m : Achieve
InformalDef
   id : "A meeting initiator shall know the schedules of the various participants invited to the meeting within
2 days after the meeting initiation."
FormalDef
   fd : $ forall m/Meeting,  p/Participant, mi/Role, t1/TimePoint
   ((mi Initiator m) (p Invitee m) (m CreationDate t1)) ==> (Eventually[(mi Knows p!Schedule)/prop,t1+2/
lastDay]) $
End
```

The requirement InitiatorKnowsConstraints has both an informal definition and a formal definition. The formal definition. is a set of constraining relationships over the requirement ontology.[11] Formula types, variables, and attributes become classes, class instances, and class attributes in the requirement ontology. For example, in the InitiatorKnowsConstraints requirement, a meeting initia-

[10] The ontology is illustrated using the Unified Modeling Language (UML) notation.

[11] The language defining the formal definitions of these requirements is that of ConceptBase[111]. It is a deductive database that provides techniques, such as such arbitrary relationships between objects, recursive processing of objects, and parameterized query classes. The requirement, InitiatorKnowsConstraints, references the QueryClass Eventually—it implements the temporal logical operator ◊. A QueryClass is itself an instance of one or more classes. Through its constraint and isA specification, a QueryClass defines necessary and sufficient conditions for objects that are instances of it. Such conditions are used to compute the objects that answer the query.

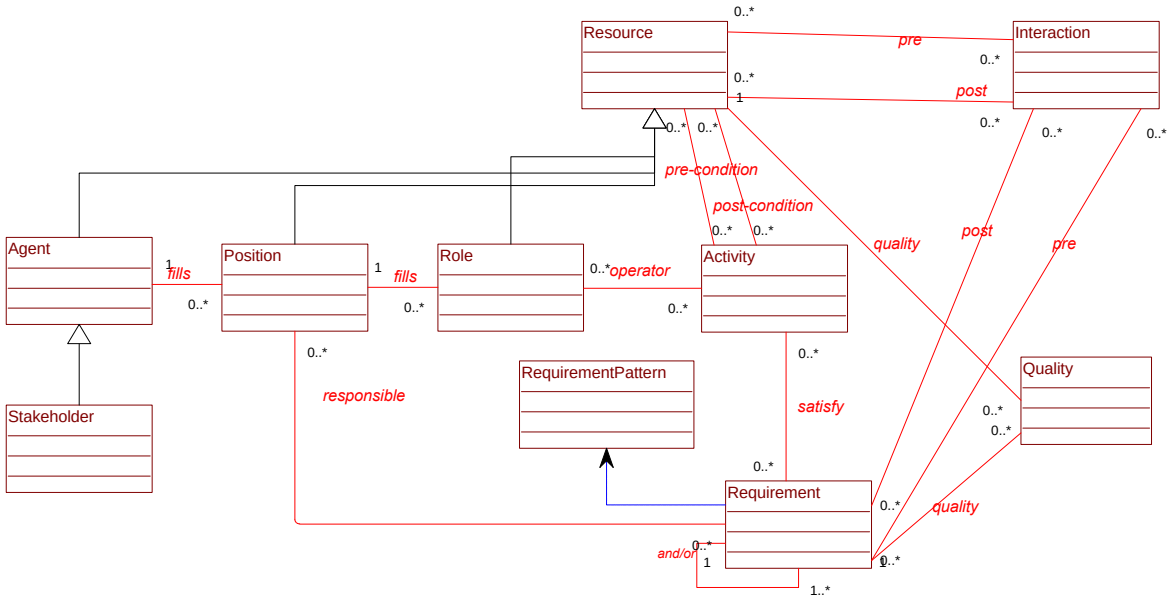


Figure 3. An illustrative agent-oriented requirement ontology.

tor, *mi*, is an instance of the Role class, while a participant instance, *p*, is an instance of a Participant class, which in turn, is a subclass of the Role class.

A requirement is formally define through the propositions of its formal definition. As shown in figure 3, a requirement can be instantiated from a requirement pattern. Moreover, requirements can be specialized (IsA) and decomposed (and/or).

The requirement ontology can be used to express agent-oriented qualitative requirement goals, as illustrated in the following requirement.

Requirement MinimizeParticipantSchedulingEffort **with**
InformalDef
id : "The scheduler shall minimize participant scheduling effort.
End

The above informal requirement can be formalized using a *requirement pattern*, PositionOptimize-sAnotherPositionActivityQuality. As shown below, the semantics of such patterns can be expressed in terms of logical propositions over the requirement ontology.

```
RequirementPattern PositionOptimizesAnotherPositionActivityQuality with
parameter
   position1, position2 : Role;
   activity1 : Activity;
   quality1 : Quality
InformalDef
   id : "For all instances of POSITION1 and POSITION2, where POSITION2 has a role in ACTIVITY,
POSITION1 should operate another activity (ACTIVITY2) that is part of an interaction that supports
POSITION2's ACTIVITY's QUALITY in reaching its optimum."
FormalDef
   fd       : $ forall position1,position2/Role
              ((activity1 Operator position2))
                 ==> (exist activity2/Activity interaction1/Interaction
                        (activity2 Operator position1)
                        and (interaction1 PreCondition activity2) and (interaction1 PostCondition activity1)
                        (interaction1 quality quality1) and (interaction1 deltaQuality Optimum))  $
end
```

As shown above, parameters are used to apply the pattern to specific requirements. For example, the following instantiates the parameters to define a formal requirement for MinimizeParticipantSchedulingEffort.

```
PositionOptimizesAnotherPositionActivityQuality SchedulerOptimizesParticipantSchedulingEffort
      [Scheduler/position1,Participant/position2,MeetingScheduling/activity1,Effort/quality1]
End
```

The above parameterized requirements can be interpreted as follows.

```
For all instances of SCHEDULER and PARTICIPANT, where PARTICPANT has a role in
MEETINGSCHEDULING, the SCHEDULER should operate another activity that is part of an interaction
that supports
PARTICPANT's MEETINGSCHEDULING EFFORT in reaching its optimum.
```

Requirements formalized using the ontology can be directly analyzed, as illustrated in section 4.2.2. To describe the resulting requirement interactions, an ontology for requirement interactions can be used. Next, we present a simple requirement interaction ontology.

5.3 Anatomy of an Interaction

As introduced in section 2.2, a requirement interaction describes a certain type of relationship between requirements. In part (a) of figure 4, the elements of a requirement interaction are illustrated. In part (b) of figure 4, an example requirement interaction is shown. It specifies that how a requirement on complete information increases participant scheduling effort. That is, SchedulerOptimizesParticipantSchedulingEffort interacts with the following requirement.

```
InformationGoal CompleteScheduleInformation with
Mode
   m : Optimize
InformalDef
   id : "A scheduling decision shall be based on complete information."
FormalDef
fd : $ forall s/Scheduler m/Meeting  d/Decision i/Information completeness/CompletenessQuality
      ((s Scheduling m) and (s Decision d)  and (PreCondition d i) and (i Quality completeness))
      ==> (Maximum[completeness/prop])$
End
```

Each element of a requirement interaction is described below.

5.3.1 Interaction and definition

Two requirements may be associated in a relationship. As illustrated in figure 4, interaction is simply the relationship name (e.g., Increases satisfaction). The definition specifies exactly how the two requirements are related. In the case of Increases satisfaction, the definition may be a function relating the values of one requirement to another. As illustrated in figure 4, the definition may relate the two requirements informally or qualitatively (e.g., somewhat of section 4.2.1). However, the interaction may be defined more precisely as illustrated below.

Increases satisfaction (R_1, R_2) ≡ $U(R_1.\text{value}) = R_2.\text{value}$

Here, Increases satisfaction is defined with a utility function. Such a representation assumes that the satisfaction of a requirement can be quantified over a range of values (§2.1.1). Then, the degree of satisfaction of one requirement can be related to another through a function.

5.3.2 Condition

An interaction may not always be applicable. The condition specifies the circumstances under which the interaction holds. As the example illustrates in part (b) of figure 4, SchedulerOptimizesParticipantSchedulingEffort is only increased if participant input is required to increase CompleteScheduleInformation.

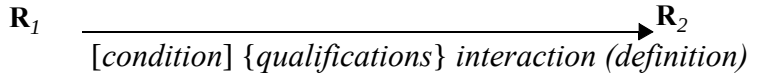
5.3.3 Qualifications

Probabilities, belief values, importance and other qualities may be assigned to an interaction. Figure 4 illustrates this as interaction *qualities*. For example, if SchedulerOptimizesParticipantSchedulingEffort increases CompleteScheduleInformation 80 percent of the time, then it could be represented with a probability of 0.8.

5.4 Instantiating the Requirement Interaction Ontology

Figure 5 illustrates the interaction of figure 4 (part b) as represented in the requirement ontology of figure 3. The interaction in figure 5 makes the qualities (completeness and effort) more explicit, while leaving out the conditional aspect of the interaction.[12] Given the two requirements, CompleteScheduleInformation and SchedulerOptimizesParticipantSchedulingEffort, and the *a prior* model that indicates Completeness increases Effort (described in 4.2.2.1, illustrated in figure 5), an inference can be made that increased satisfaction of CompleteScheduleInformation will increase the effort in SchedulerOptimizesParticipantSchedulingEffort.

(a)



(b)

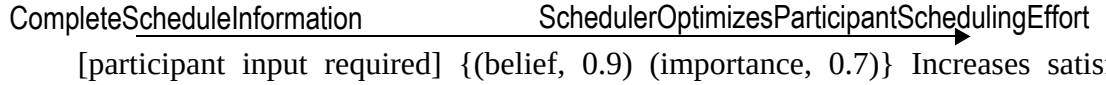


Figure 4. Illustrated anatomy of an interaction.

[12] The ontology of figure 3 does not show the details of an interaction. Figure 5 illustrates a limited extension of that ontology. The figure clarifies qualities, such as completeness and effort, by explicitly representing qualities as attributes of requirements. Each quality of the interaction has a separate function (in the Quality class) that defines how changes in scheduling information effect each quality.

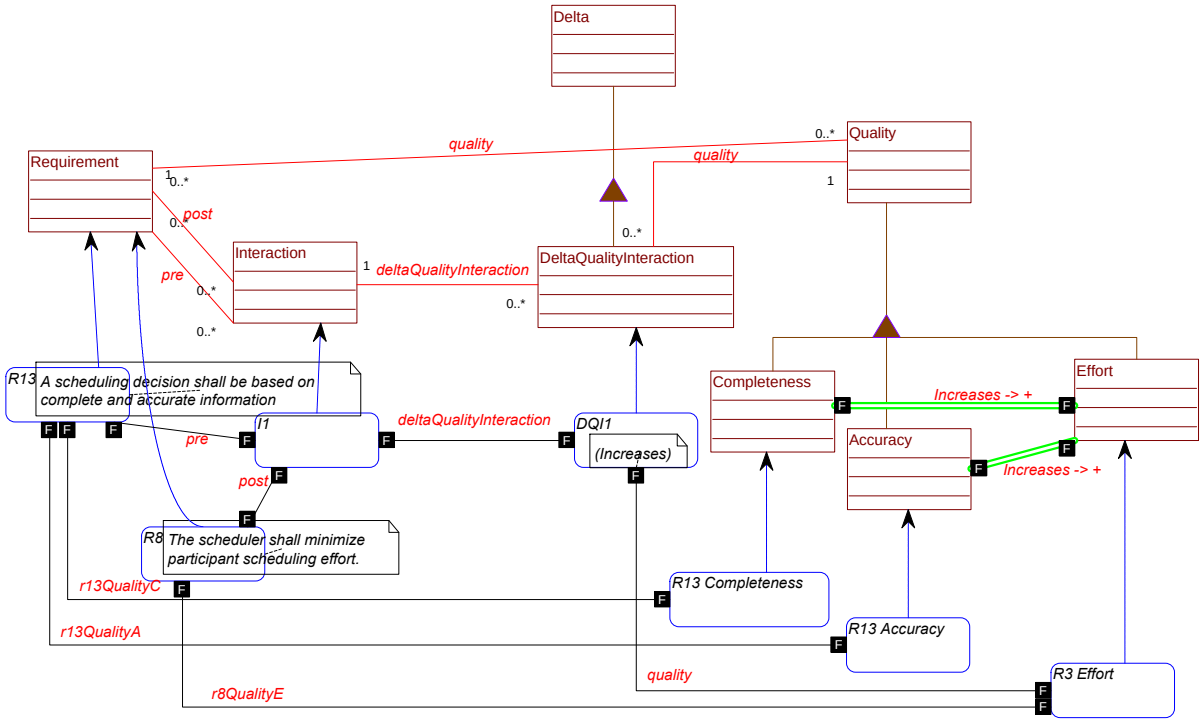


Figure 5. An illustration of a requirement level interaction.

5.5 Summary

A requirement interaction ontology provides for the description of how one or more requirements affect other requirements. Typically, certain properties of a requirement are distinguished as having an affect other requirements. For example, satisfaction of security for one requirement may affect the efficiency of another requirement. Researchers are developing ontologies that include a useful set of requirement properties and their relationships. Additionally, they are developing *a prior* models and algorithms to aid interaction analysis of requirements.

6 PROJECTS ILLUSTRATIVE OF REQUIREMENTS INTERACTION MANAGEMENT

This section summarizes seven projects that illustrate some aspect of requirements interaction management[13] The summaries are not comprehensive. Moreover, the projects are not directly comparable, as they seek to achieve different research goals. Nevertheless, each project does provide 1) representations of interactions among requirements, 2) reasoning about interactions, and the removal of conflicts through requirement modifications, and 3) some computer support for analysis. Together, the seven project provide an overview of the variety of support that can be provided for requirements interaction management.

Below, tables 8 and 9, provide a comparative overview of the conflict detection and resolution support provided the projects. (See sections 4.2.1 and 4.4.1 for a description of the issues.) For each issue considered, a value of A+, A, M, or N is provided. These values indicate (decreasing) levels of automated support. For example, deficiency drive design requirements analysis (DDRA) directly addresses semantic conflicts of assumptions through the Critic tool[71], so it is marked

[13] Summaries are current as of 8/1/99.

with a “A+”. In contrast, SCR lacks a domain model by which to analyze goal assumptions, so it is marked with an “N”. Unfortunately, the projects do not directly indicate their support for all issues. So, some of the values had to be (subjectively) inferred. Thus, the tables should be considered as the authors opinion. As such, the tables can be used as an index into the literature for readers who seek more detailed information.

Table 8. Conflict Detection Methods by Project[a]

Interaction Type			Win-Win	NFRs	View-Points	KAOS[b]	DDRA	SCR	M Telos
Syntax Conflicts	***Synonyms***		N	N	M	M	N	A	N
	Homonyms		N	N	M	M	N	N	N
	Mistake		N	N	M	N	N	N	N
Semantic Conflicts	***Perception***	*goal*	A+	A+	A+	A+	A+	A	A
		assumption	N	M	N	M	A+	N	N
		domain boundary	A+	N	A+	N	A+	N	A+
		cognitive conflict	A	A	N	N	A	N	N
	Communication	*abstraction level*	A	A	N	N	A	N	A
		accuracy	A	A	N	N	A	N	A
		circular justification	N	N	N	N	N	N	N
	Resource	*quantity*	A	A	N	A	A	A	N
		quality	A+	A+	N	A	A	N	N
		availability	A	A+	N	A	A	A	N
		redundancy	A	A+	N	A	N	N	N
	Behavior	*deadlock*	N	N	N	A+	A+	A+	N
		deadline	N	N	N	A	A	A+	N
		commitment	N	N	N	N	N	N	N
		normative conflict	A	A	N	N	A	N	N

a.*Key::* M = some support manual, A = some computer automation, A+ = computer automation specifically designed to solve the problem, N = no support described.
b.KAOS automation is described rather than implemented.

Next, in each of the following subsections, a project is summarized in terms of the process and products of sections 4 and 5, respectively.

Table 9. Conflict Resolution Methods By Project[a]

Method	Win-Win	NFRs	View-Points	KAOS[b]	DDRA	SCR	M Telos
Relaxation *generalization value-range extension*	M	N	N	A+	A+	N	N
Refinement *specialization*	M	N	N	A+	A+	N	N
Compromise	M	N	N	A+	A+	N	N
Restructuring *related resource related requirement distribution*	M	N	N	A+	A+	N	N
Failure Recovery *re-enforcement re-planning*	M	N	N	A+	A+	N	N
Other *postponement abandonment*	M	N	N	A+	A+	N	N

a.*Key::* M = some support manual, A = some computer automation, A+ = computer automation specifically designed to solve the problem, N = no support described.
b.KAOS automation is described rather than implemented.

6.1 WinWin

The WinWin project supports collaboration among a wide set of system stakeholders as a means to improve software development outcomes. Too often, software development solutions satisfy only a subset of all stakeholders, the "winners". Table 10 illustrates the distribution of winners and losers for some typical software development solutions.

Table 10. Frequent Software Development Win-Lose Patterns[a]

Proposed Solution	Winning stakeholders	Loosing Stakeholders
Quick, cheap, sloppy product	Developer, Customer	User
Lots of features (a.k.a., "bells & whistles")	Developer, User	Customer
Driving too hard a bargain	Customer, User	Developer

a.Table reproduced from [15].

As a means to have a winning outcome for all stakeholders, the WinWin project is developing software support for multi-stakeholder requirements analysis and is integrating such analysis into the larger software development life-cycle.

In WinWin, development of winning stakeholder requirements is a process. Figure 6 illustrates the overall WinWin process. The process model combines the risk reduction strategy of the spiral model[21] with the negotiation-oriented philosophy of Theory-W[22]. On each cycle through the processes in figure 6:

- Stakeholders are identified
- Requirements of each individual stakeholder are identified, called *win-conditions*.

- Requirements interactions are identified, called Conflict/Risk/Uncertainty Specifications (*CRU's*), and their resolutions are captured as Point of Agreements (*POA's*).
- The product and process descriptions are elaborated according to the new requirements. Alternative means of satisfying the new collaborative requirements are considered and selected based on risk reduction.
- The next cycle of the project is planned, validated, and reviewed.

Three major milestones have been defined for the overall WinWin spiral process model: Life-Cycle Objectives (LCO), Life-Cycle Architecture (LCA), and Initial Operational Capability (IOC). Requirements are among the six attributes that characterize each milestone. Stakeholders must commit to milestones between project inception, elaboration, and construction[16].

6.1.1 Processes

The WinWin project has developed the WinWin tool which provides computer support for collaborative stakeholder requirements analysis. Its support can be considered in terms of the processes described in section 4, as shown below.

- ***Partitioning***. WinWin allows for the partitioning of according to attributes attached to requirements. These include, stakeholder ownership and attribute types from a project-defined taxonomy of requirement attribute types. Figure 7, presents a summary of one such taxonomy (from [17]). Each attribute type is also linked to 1) stakeholder roles, 2) inter-attribute relationships, and 3) strategies for reducing conflict.

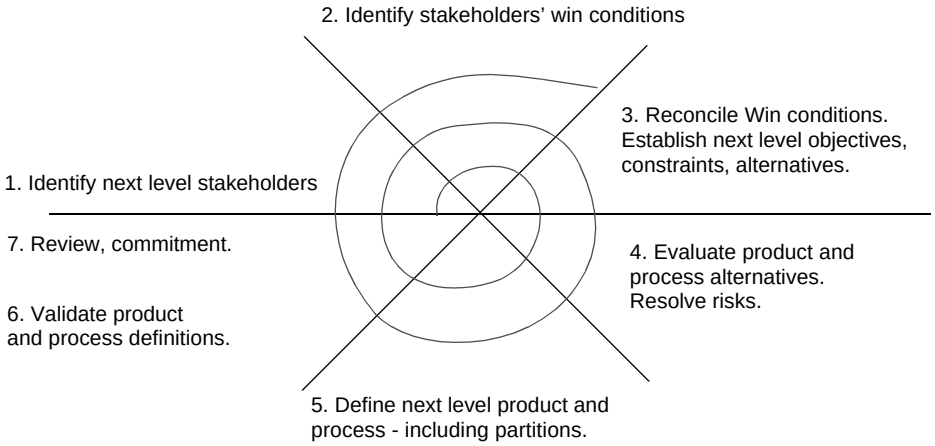


Figure 6. The WinWin spiral process model.

1 Media operations	2.2 Database (File Access)	4.1 Response Time
1.1 Query/Search/Browse	2.3 User/Admin. Interface	4.2 Reliability
1.2 Access Control	2.9 Others	4.3 Security
1.3 Audio/Video Operation	3 Administration	4.4 Usability
1.4 Update/Input	3.1 User Management	4.5 Interoperability
1.9 Others	3.2 Usage Monitoring	4.6 Workload
2 Interface	3.9 Others	4.7 Cost
2.1 COTS (SIRSI, etc.)	4 Quality	4.8 Schedule
		4.9 Others

Figure 7. A WinWin project-specific taxonomy of attribute types.

  - Stakeholders vary in the importance they place on attribute types. QARCC, a component of WinWin, pre-defines an association of stakeholder roles to attribute types. For example, the User stakeholder role cares about Usability and Performance, while the Developer stakeholder role cares more about Cost and Schedule[19].
  - It may be known *a priori* that certain attribute types conflict with other attribute types. (See section 4.2, especially figure 4.) In QARCC, each attribute type has an associated set of supportive and detracting attribute types. These sets are used in identifying potential requirement conflicts.
  - It may be known *a priori* that certain attribute types, when in conflict, can have their conflict reduced through a set of general strategies. In QARCC, each attribute type has an associated set of textual process and product strategies that stakeholders may consider when confronted with a conflict.
- ***Identification***. When a requirement (a.k.a. win-condition) is entered into the QARCC component of WinWin, a list of potentially conflicting requirements is presented. This list is generated by: 1) retrieving the attribute types associated with the new requirement, 2) retrieving the associated set of potentially conflicting attribute types, and 3) finding the set of other requirements that have those types[19]. For each such conflict, a Conflict Advisor Note message is sent to stakeholders who have indicated concern about the attribute type.
- ***Focus***. WinWin artifacts, such as requirements and resolutions, can be sorted by artifact attributes, such as owner, status, priority, revision date, *etc*.
- ***Resolution***. When a potential conflict is identified, QARCC can present users with a list of predefined (text) strategies that may apply to the given attribute type conflict. A user may use this information to define a resolution Option.
- ***Selection***. An resolution Option can be selected from the set of Options (defined by users of WinWin).

6.1.2 Products

In the WinWin project, the major requirement artifacts include: requirements, conflicts, resolutions, and agreements. Figure 8 illustrates the relationship among them. These classes are also reflected in the simple requirement ontology of figure 3, section 5.[14] In contrast, WinWin has a limited a *priori* requirement ontology. For example, in figure 3, classes are used to define the elements of a requirement (e.g., Activity, Position, Resource) independent of a project. Software tools can rely on the instantiation of such classes to reason about requirements. (See for example, the *i** project in section 6.2.) In WinWin, such terms are defined in the taxonomy on a project-specific basis. However, QARCC does have an attribute type taxonomy that includes an *a priori* model of requirement interactions and resolution strategies. (These elements are also reflected in the requirement interaction of figure 5, section 5.) This model enables QARCC to provide lists of potential requirement conflicts and suggestions on how one might resolve them.

6.1.3 Case-Study Results

Since 1995, the WinWin project has published articles describing case-studies of software analysis using WinWin. Several results of those case-studies are particularly noteworthy for requirements interaction management. They include the following observations.

[14] In figure 3, a WinWin requirement or resolution is a Requirement, a WinWin conflict is an Interaction, and a WinWin agreement is a link from a Requirement to a Resolution (not illustrated in figure 3). For a reference on rationale models, see [208].

- *Between 40% and 60% of requirements involve conflicts.* In a two year comparison of projects involving 37 student teams, a significant number of requirements raised conflict[55].
- *Most conflicts are simple to resolve.* This result seems to depend on the complexity of the project, domain experience of the stakeholders, and the resource constraints placed on development. Moreover, the time to create even a single resolution can be significant. Nevertheless, in 37 WinWin projects, between 45% and 69% of conflicts only required one resolution option before an agreement was reached[55].
- *Developers contribute the most to identifying and resolving conflicts*. The stakeholder roles of User, Customer, and Developer contribute varying ways to the project. Users and Customers contribute more to requirement identification, while Developers contributed more to conflict and resolution identification[55].
- *System qualities (non-functional requirements) involved the greatest conflict.* The next most controversial attribute type of the taxonomy in figure 7, operations, had nearly half as many conflicts as did the system qualities[17].

6.2 Non-Functional Agent Oriented Requirements

Since 1992 researchers at (and from) the University of Toronto have published papers on the modeling and analysis of non-functional requirements (NFRs) and agent-oriented requirements (*i**)[178]. Generally, these two topics are presented separately; however, here both are presented in relation to the overall topic of requirements interaction management.

Formal modeling of high-level business requirements has been the mainstay of this work. With the Requirements Modeling Language (RML)[88] as a precursor, the Toronto group has developed and formalized semantics of non-functional requirements, methods for elaborating requirements, correlations among requirements, as well as link that analysis into an agent-oriented view of business requirements.

6.2.1 Products

Figure 9 illustrates functional and non-functional requirements in relation to an *i** model of actor-dependences. Functional requirements, such as ScheduleMeeting, are represented as an AND/OR hierarchy[179]. Thus, figure 9 illustrates that to achieve ScheduleMeeting, both schedules must be obtained and a scheduled match must be found.

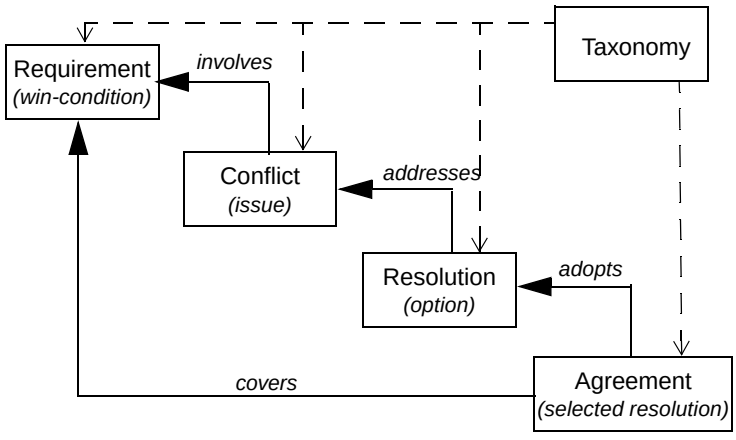


Figure 8. WinWin artifacts. (Adapted from [17].)

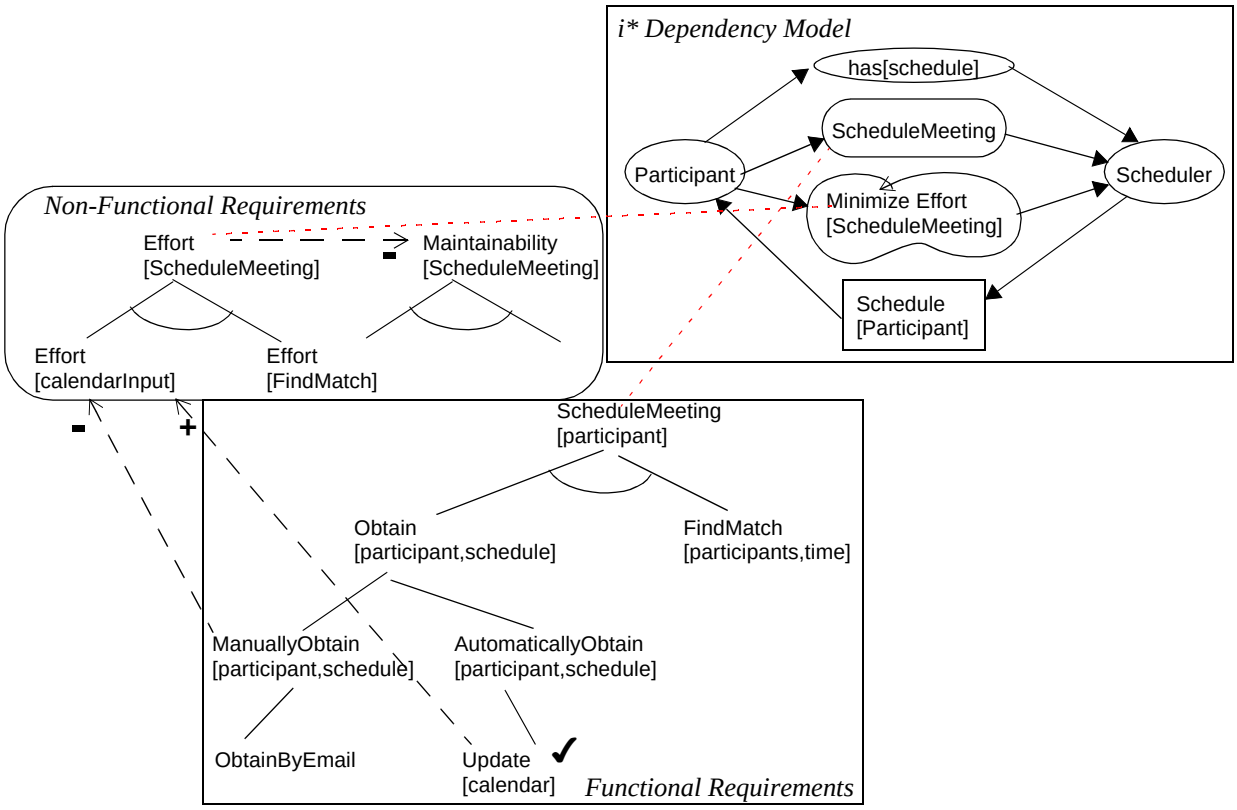


Figure 9. An illustration of functional, non-functional requirements, and the *i** model.

Non-functional requirements are illustrated in the middle part of figure 9. Such requirements are also represented as an AND/OR hierarchy. Non-functional requirements describe qualities of the functional requirements. Thus, the non-functional requirements of figure 9 are parameterized by functional requirements.[15] For example, the Effort of ScheduleMeeting is determined by the Effort to obtain calenderInput and the Effort to FindMatching schedules.

Non-functional requirements are linked to the functional requirements via supports (+) or detracts (-) relationships. For example, Update[calendar] supports the non-functional requirement of (minimizing) Effort. This can be useful in determining if a non-functional requirement has been achieved. For example, in the figure, Update[calendar] has been satisfied (indicated with a "✓"). If all AND subrequirements have similarly been satisfied, then the overall requirement is said to be satisfied. Of course, the satisfaction of other requirements may interfere. From figure 9, it can be seen that ManuallyObtain[participant,schedule] detracts from the satisfaction of Effort. However, the figure does not show that ManuallyObtain[participant,schedule] (or its descendents) has been satisfied. So Effort[calenderInput] is indeed satisfied.

In general, determining the satisfaction of a non-functional requirement can be quite difficult. Some functional requirements can detract from a NFR while others support it. Moreover, the satisfaction of one NFR can detract from another NFR. This is illustrated in figure 9 with the negative link from Effort to Maintainability. It means that satisfying Effort (via minimizing) will detract from the satisfaction of Maintainability. Despite such complexity, a qualitative label propagation algo-

[15] More specifically, non-functional requirements are parameterized by topic. A *topic* is a class that may be used in, or constrained by, a requirement. In figure 9, requirement names are used, rather than the more indirect topic names to reinforce the relationship between non-functional requirements and functional requirements.

rithm has been developed that can determine if a non-functional requirement has been satisfied, denied, or undetermined[178].

Catalogues of non-functional requirements interactions have been specified[35][37]. The catalogues indicate how the satisfaction of one NFR can detract from the satisfaction of another. (See section 4.2, especially figure 4.) Like most research projects, the catalogues have been project specific descriptions. However, since the NFR interactions are arranged in a hierarchy of NFR abstraction parameterized by topic, one may reuse the more abstract NFR descriptions across projects.

The top of figure 9 illustrates an *i** actor-dependency model for the requirements[177]. The model "views an organization as a network of *intentional* dependencies among actors in a social environment."[283].[16] The circles of Participant and Scheduler indicate actors in the model.[17] The four links between the actors indicate their dependencies:

- *Goal dependency.* The Participant depends on the Scheduler to have the goal has[schedule]. It is unspecified how the Scheduler may satisfy this goal, but simply that the Participant relies on the Scheduler to do so.
- *Task dependency.* The Participant depends on the Scheduler to carry out the task ScheduleMeeting. One may infer that ScheduleMeeting is a task that implements the functional requirement ScheduleMeeting[participant] that satisfies the goal has[schedule]; however, such inter-model relationships have not been defined as of yet.
- *Softgoal dependency.* The Participant depends on the Scheduler to perform some task that satisfies the softgoal Effort[ScheduleMeeting]. Like the goal dependency, the softgoal dependency does not indicate which task the Scheduler should do in order to satisfy Effort[ScheduleMeeting].
- *Resource dependency.* The Scheduler depends on the Participant to make the resource Schedule[participant] available.

The dependencies of the *i** model provide a context in which functional and non-functional requirement interactions can be understood in relation to organizational intentions.

6.2.2 Process

The University of Toronto projects have resulted in a number of software prototypes. The collective support that they could provide can be considered in terms of the processes described in section 4, as shown below.

- ***Partitioning***. The NFR framework allows for the partitioning of requirements into functional and non-functional type hierarchies parameterized by *topic* classes. Moreover, requirements can be stored in an object-oriented database[111], so other attributes can be used to partition requirements. Finally, the interrelationship of the requirements and the *i** model can be used to partition requirements.
- ***Identification***. When a requirement is entered into a NFR support system and associated with existing NFRs, it can be determined, via the *a prior* supports and detracts links, which other requirements are affected. Moreover, these effects can be understood within the overall *i** organizational model. Additionally, label propagation from selected requirements can determine the cumulative effect of requirements on non-functional requirement satisfaction.
- ***Focus***. NFR framework requirements can be sorted by their attributes and relationships. Inter-

[16] *i** denotes the "distributed intentionality" of the model.

[17] The *i** model is intended to show the *intentional* relationships of organizational actors. Thus, a computerized component, such as the Scheduler, is rarely shown as it is awkward (although not incorrect) to indicate its intentions. However, showing the Scheduler in figure 9 completes the illustration of the example.

actions are represented as unattributed relationships, thus interactions can be sorted by the interaction type and the associated requirements of the interaction.

- ***Resolution***. When a potential conflict is identified within the NFR framework, alternative resolutions can be generated by an analyst and added to the requirements as an OR node.
- ***Selection***. A requirement may be selected as a resolution by marking it as selected (indicated with a "✓"). There is support for recording and analyzing claims, in support or against, a particular requirement decomposition. Such argumentation may be used to select a requirement[36].

6.3 Viewpoints

Since 1992 researchers at (and from) Imperial College have published papers on modeling and analysis of system descriptions from multiple viewpoints[75]. Their ViewPoint framework provides a means to partition requirements and analyze relationships between partitions[187].

6.3.1 Products

The ViewPoint framework supports multiple perspectives, or *views,* of requirements. A view typically captures only a portion of the overall system description (i.e., a *partial specification*). Moreover, views of a system can vary in scope, representation, stakeholder ownership, or other dimensions. Easterbrook and Nuseibeh summarize ViewPoints, as follows[53]:

> ViewPoints are loosely coupled, locally managed, distributable objects which encapsulate partial knowledge about a system and its domain, specified in a particular, suitable representation scheme, and partial knowledge of the process of development.
>
> Each ViewPoint has the following slots:
>
> - a representation style, the scheme and notation by which the ViewPoint expresses what it can see;
> - a domain, which defines the area of concern addressed by the ViewPoint;
> - a specification, the statements expressed in the ViewPoint's style describing the domain;
> - a work plan, which comprises the set of actions by which the specification can be built, and a process model [74] to guide application of these actions;
> - a work record, which contains an annotated history of actions performed on the ViewPoint.

Relationships among ViewPoints can be determined by the application of consistency rules. These rules are used to detect inconsistencies between ViewPoints. The rules typically apply to ViewPoints using a common representation, such as data flow diagrams[187] or state transition diagrams[74]. Each rule has the following form (see [52]):

$$\exists\ VP_D(t,d) \bullet \{VP_S : ps_1 \Re\ VP_D\ ps_2\}$$

In words, it can be used to determine if there exists a ViewPoint in the destination, VP_D, whose type is *t* and domain is *d*, such that a partial specification of a source ViewPoint, VP_S, is related to VP_D by relation $\Re$. As an example, the following rule expresses that "Process names must be unique across all DFDs"[52]:

$$\exists\ VP_D(DFD,D_a) \bullet \{VP_S : \text{Process.Name} \neq VP_D : \text{Process.Name}\}$$

Complex rules, and interrule relationships, can be expressed within the ViewPoints framework.

6.3.2 Process

As part of the ViewPoints project, a variety of computerized support has been specified in addition to the software prototype, called the √iewer[190]. The collective support that ViewPoints could provide can be considered in terms of the processes described in section 4, as shown below.

- ***Partitioning***. The ViewPoints framework allows for the partitioning of requirements into any subsets an analyst chooses. The framework itself does not provide any categories; rather, it provides the views into which one can place requirements.
- ***Identification***. Once requirements are represented in ViewPoints, an analyst can apply the consistency rules to determine inconsistencies between ViewPoints.
- ***Focus***. Inconsistencies are associated as postconditions with the rule that generated them. Thus, inconsistencies are referenced by their consistency rule. It has not been demonstrated how the framework could prioritize the further analysis or the resolution process. However, interrule relationships can be used to specify dependencies in the ordered application of consistency rules [52].
- ***Resolution***. An analyst may generate a resolution by applying a resolution rule that is associated with the corresponding consistency rule that was violated[52]. Alternatives may be considered in the scope of a hierarchy of consistent ViewPoints[51]; however, in general, resolution alternatives are not explicitly represented.
- ***Selection***. A resolution is implicitly selected by the application of resolution rule.

6.4 KAOS

Since 1991 researchers at Université catholique de Louvain have published papers on the KAOS project (Knowledge Acquisition in autOmated Specification of software) for modeling and analysis system requirements[262]. The project is broad in its scope, and includes: meta-modeling, specification methodology, learning, and reuse. Here, we summarize the KAOS meta-model (ontology) and the KAOS paradigm of support for requirements interaction management.

6.4.1 Products

The KAOS language provides two basic levels of descriptions: “an outer semantic net layer for *declaring* a concept, its attributes and its various links to other concepts; an inner formal assertion layer for *formally defining* the concept.”[43] At the semantic layer, KAOS provides an ontology similar to that introduced in section 5.2. The KAOS classes are summarized below (cf., [42][266]).

- *Object*. An object is a thing whose instances may evolve from state to state. (An object is similar to a UML class, in that it has instances whose attributes values can vary over time.) An object can be specialized to be an *entity, relationship*, or *event* depending if the object is autonomous, subordinate, or instantaneous, respectively.
- *Operation*. An operation is an input-output relation over objects; it defines state transitions.
- *Agent*. An agent is an autonomous object that can perform the operations assigned to it.
- *Goal*. A goal represents state that a system should meet; sometimes called an objective. Goals can be refined into an AND/OR directed acyclic graph.
- *Requisite, requirement, assumption*. A requisite is a goal that can be formulated in terms of states controllable by an agent. In other words, a requisite is a goal that *can be* assigned to agent with the expectation that the agent can satisfy the goal through performing operations. A requirement is a requisite that has been assigned to a system agent, while an assumption is a requisite that has been assigned to an environmental agent.

- *Scenario*. Typically, scenarios show how goals can be achieved. A scenario is a composition of operation applications. The composition of scenario operations must satisfy the pre- and post-condition constraints of operations. Additionally, objects can have associated domain invariants that must be satisfied.

6.4.2 Process

As part of the KAOS project, a variety of computerized support has been specified in addition to the construction of the software prototype, called the GRAIL[44]. The collective support that KAOS could provide can be considered in terms of the processes described in section 4, as shown below.

- ***Partitioning***. All KAOS objects, including requirements, are store in an object-oriented database, so object type and associated attributes can be used to partition requirements[44].
- ***Identification***. A variety of requirements interactions types can be identified within the KAOS framework. Table 11 summarizes the KAOS inconsistency types. (These are the formalized and specialized counter parts of the conflict types found in table 5, of section 4.2.1.) In the table, conflict is defined as a logical inconsistency among assertions. In KAOS, assertions are the logical formulas that formally defined objects, including requirements. Divergence is perhaps the most interesting of the inconsistency types. As an example, consider a requirement divergence in resource management requirements (from [266]). A user requirement might state that “if a

Table 11. KAOS Inconsistency Types

Inconsistency	Illustration	Description
Process-level deviation	aProcesRule ≡ ∀ r:Requirement, Prop(r) *and in requirements definition:* ∃r:Requirement ¬ Prop(r)	A state transition in the RE process that results in an inconsistency between a RE process rule and a state of the RE process.
Instance-level deviation	aRequirement ≡ ∀ x::X, Prop(r) *and in running system:* $\exists x_i$:X ¬ Prop(x_i)	A state transition in the running system that results in an inconsistency between a product level requirement and a state of the running system.
Terminology Clash	Attends(participant, meeting) ∈ SRS_1, ∧ Participates(participant, meeting) ∈ SRS_1	A single real-world concept is given different syntactic names in the requirements.
Designation Clash	Attends ≡ "attending meeting m until the end" ∈ SRS_1, ∧ Attends ≡ "attending part of meeting m" ∈ SRS_1,	A single syntactic name in the requirements specification designates different real-world concepts[284].
Structure Clash	ExcludedDates ≡ SetOf[TimePoint] ∈ SRS_1, ∧ ExcludedDates ≡ SetOf[TimeInterval] ∈ SRS_1,	A single real-world concept is represented with different structures in the requirements specification.
Conflict	(1) {DomainTheory, $\wedge_{1\le n}$ A_i} |– false (2) ∀i {DomainTheory, $\wedge_{j\neq i}$ A_j} |⊬ false	A conflict among assertions (elements of formalized requirements) occurs within a domain theory when: (1) the set of assertions are logically inconsistent within the domain, (2) removing any one of the assertions removes the inconsistency.
Divergence	(1) {DomainTheory, *B*, $\wedge_{1\le n}$ A_i} |– false (2) ∀i {DomainTheory,*B*, $\wedge_{j\neq i}$ A_j} |⊬ false (3) ∃ S:Scenario S ⊨ *B*	A divergence among assertions (elements of formalized requirements) occurs within a domain theory iff there exists a *boundary condition B*, such that: (1) the set of assertions are logically inconsistent within the domain including B, (2) removing any one of the assertions removes the inconsistency, and (3) there exists a feasible scenario S that satisfies B.
Competition	aRequirement ≡ ∀ (x:X) A[x] (1) {DomainTheory, *B*, $\wedge_{i\in I}$ A[x_i]} |– false, for some I (i.e., require instance) (2) ∀i {DomainTheory,*B*, $\wedge_{i\in J}$ A[x_i]} |⊬ false, for any J ⊂ I (3) ∃ S:Scenario S ⊨ *B*	Competition is a particular type of divergence that occurs when different instances A[x_i] of the same universally quantified requirement ∀ x: A[x] are divergent.
Obstruction	(1) {DomainTheory, *B*, A} |– false (2) ∀i {DomainTheory,*B*} |⊬ false (3) ∃ S:Scenario S ⊨ *B*	An obstruction is a borderline case of divergence in that is only involves one assertion. It amounts to an obstacle to the satisfaction of a requirement[200].

user is using a resource, then they will continue to use the resource until is no longer needed".[18]

∀ u:User, r:Resource
Using(u,r) ⇒ o [Needs(u,r) → Using(u,r)]

In contrast, library staff might state a requirement that "if a user is using a resource, then she

[18] KAOS formal definitions use temporal logic operators[161]. Here, o means in the next state. Other operators are included for the previous state (•), some time in the future (◊), some time in the past (◆), always in the future (□), always in the past (■).

will return the resource after some *d* days."

$$\forall\ u{:}User,\ r{:}Resource$$
$$Using(u,r) \Rightarrow \Diamond_{\leq d} \neg\ Using(u,r)$$

While the two requirements are not logically inconsistent, a problem does arise when a user needs (and uses) a resource for more than *d* days.

$$\Diamond\ (\exists\ u'{:}User,\ r'{:}Resource)\ [Using(u',r') \wedge \Box_{\leq d}\ Needs(u',r')]$$

The expression of this problem is a *boundary condition*—an expression that makes the requirements logically inconsistent. Boundary conditions are also be expressed where two instantiations of a requirement compete or where a requirement competes with the environment (i.e., an obstacle[200]).

Identification techniques. The KAOS framework defines several means of identifying inconsistencies; table 12 summarizes those techniques. In KAOS, detection is a manual procedure; however, other projects have automated some of the detection techniques[266].

Assertion regression (a.k.a., goal regression[271]) is a particularly interesting divergence detection technique. It is used to identify a boundary condition, *B*, that leads to the divergence. Given rules of the form $X \Rightarrow Y$, regression determines what must be true if the rule is to be applied to satisfy a specific assertion A. Regression is essentially a backward application of a rule. It is only useful when the right-hand side of the rule, Y, unifies with the assertion, A. That is, the rule can satisfy A. For example, consider the following rule and goal:

Rule: $o\ Needs(u,r) \rightarrow o\ Using(u,r)]$
Goal: $\Diamond\ \exists\ u{:}User,\ r{:}Resource\ Using(u,r) \wedge \Box_{\leq d}\ o\ Using(u,r)]$
Goal regressed through rule: $\Diamond\ \exists\ u{:}User,\ r{:}Resource\ Using(u,r) \wedge \Box_{\leq d}\ o\ Needs(u,r)$

Above, regression amounts to replacing the part of the goal that unifies with the right-hand side of the rule with the left-hand side of the rule. In general, unifying Y with part of an assertion $\neg A_i$ produces a match μ that can be substituted into $\neg A_i$ to produce a description *B* that if applied to $X \Rightarrow Y$, will result in $\neg A_i$. That is, it produces a boundary condition. Thus, given a set of requirements, R, one can apply assertion regression to R to find various boundary conditions that lead to divergence.[19] One critical difficulty is knowing which subsets of R to consider, as not all subset of R will have a divergence.

Other identification techniques include *detection patterns* and *detection heuristics*.

[19] Different boundary conditions can be had by: (1) selecting different assertions $\neg A_i$, (2) selecting different rules $X \Rightarrow Y$, (3) backchaining through the rules (e.g., $X \Rightarrow Y$, $W \Rightarrow X$) and (4) selecting different unifications in the case that there is more than one maximally general unification.

Table 12. KAOS Techniques for Inconsistency Detection

Method	Illustration	Description
Assertion Regressing	(1) Consider {DomainTheory,*B*, $\wedge_{j \neq i}$ A_j} $\vdash \neg A_i$ (2) Construct *B* by starting with $\neg A_i$ (3) Select a definition of the form $X \Rightarrow Y$ (4) Let μ be the MostGeneralUnification (f,Y) where f is some subformula in $\neg A_i$ (5) Construct *B* by substituting X.μ for f in $\neg A_i$: Let $B := \neg A_i[f / X.\mu]$	Given a divergence that involves assertions $\wedge_{1 \leq n} A_i$ and a domain theory of assertions, one can derive a boundary condition *B* by regressing the negation of an assertion $\neg A_i$ through rules of the form $X \Rightarrow Y$.
Detection Patterns	Given assertions of the *Achieve-Avoid* pattern: $(P \Rightarrow \Diamond Q) \wedge (R \Rightarrow \neg \Box S) \wedge (Q \Rightarrow S)$ Consider the boundary condition: $\Diamond (P \wedge R)$	Apply divergence patterns that have been proven to generate boundary conditions given certain patterns of assertions.
Detection Heuristics	If there is a *SatisfactionGoal* and a *SafefyGoal* concerning the same object (like Achieve-Avoid above), then consider a divergence between the two goals.	Apply informal divergence heuristics that can suggest boundary conditions given certain types of goals.

Consider the following two goals (from [266]).

Goal Achieve[RequestSatisfied]
FormalDef
∀ u:User, r:Resource
Requesting(u,r) ⇒ ◊ Using(u,r)

Goal Avoid[UnReliableResourceUsed]
FormalDef
∀ u:User, r:Resource
¬ Reliable(r) ⇒ □ ¬ Using(u,r)

The two goals match the Achieve-Avoid pattern of table 12, as follows: P: Requesting(u,r), Q: Using(u,r), R: ¬ Reliable(r), and S: Using(u,r). Applying the Achieve-Avoid pattern correctly generates the boundary condition.

◊ ∃ u:User, r:Resource Requesting(u,r) ∧ ¬ Reliable(r)

Finally, KAOS has a number of heuristic rules that suggest where to look for boundary conditions given certain types of goals. For example, given a SatisfactionGoal and a SafetyGoal involving the same object (e.g., RequestSatisfied and UnReliableResourceUsed involve r:Resource), then consider a boundary condition for the object.

- ***Focus***. Inconsistencies are not explicitly represented as a KAOS object. Thus, focusing on a subset of inconsistencies is outside the scope of the framework.
- ***Resolution***. The KAOS framework defines several means of resolving inconsistencies; table 13 summarizes those techniques. (These are the formalized and specialized counter-parts of the resolution methods found in table 7, of section 4.4.1.) In KAOS, resolution is a manual procedure; however, other projects have automated some of the techniques[266]

  As an example, table 13 describes the Avoiding Boundary Conditions technique. It suggests that a divergence can be preventing by ensure that the boundary condition is not satisfied. Applying

Table 13. KAOS Techniques for Resolution Generation

Method	Illustration	Description
Avoiding boundary conditions	$P \Rightarrow \Box \neg B$	Given that a boundary condition *B* leads to a divergence, always avoid *B*.
Goal restoration	$B \Rightarrow \Diamond_{\leq d} \wedge_{1 \leq n} A_i$	If a boundary condition *B* cannot be avoided, then when it occurs restore the assertions of the divergence sometime thereafter.
Conflict anticipation	Where $C \wedge \Box_{\leq d} P \Leftrightarrow \Diamond_{\leq d} \neg \wedge_{1 \leq n} A_i$ Introduce $C \wedge P \Rightarrow \Diamond_{\leq d} \neg P$	Where a persistent condition P can eventually lead to conflict, anticipate the time period *d* that leads to conflict and introduce a new goal to negate the condition ¬P before the time period elapses.
Goal weakening	Given Divergence[DomainTheory,*B*, $\wedge_{1 \leq n} A_i$] Introduce $R \equiv \wedge_{1 \leq m} A_i \wedge \neg B$, *where* $m \leq n$	Weaken the goal (requirement) so that a boundary condition is not met, thereby a divergence is removed. This can be done by "adding a disjunct, removing a conjunct, or adding a conjunct in the antecedent of an implication."[266]
Resolution patterns	Temporal relaxation: weaken $\Diamond_{\leq d} A$ to $\Diamond_{\leq c} A$ where c > d	Apply resolution patterns that have been proven to generate remove boundary conditions given certain patterns of assertions. For example, given that A must occur before *d* time units, extend the time to *c*, where *c* is greater than *d*.
Alternative goal refinement	Given goal G refined to assertions $\wedge_{1 \leq n} A_i$ that involve a divergence, Consider a new refinement of G such that the new assertions $\wedge_{1 \leq n} A_j$ are not involved in a divergence.	Given a top-most goal G and its refinement hierarchy H in which the leaf goals of H lead to divergence, consider alternative refinements of G that lead to different refinement hierarchies.
Resolution heuristics	If there is a *Competition* divergence among agent instances, Then consider the introduction of a reservation policy.	Apply informal resolution heuristics that can suggest how to remove boundary conditions given certain types of goals.
Object refinement	Where $R_1 \equiv \wedge_{1 \leq n} A_i$, $R_2 \equiv \wedge_{n+1 \leq m} A_j$ and Divergence[DomainTheory,*B*, $\wedge_{1 \leq m} A_k$] Given O in *B*, specialize O to $S_{1..h}$ and Introduce $\vee_{1 \leq p} S \Rightarrow R_1 \wedge \vee_{p+1 \leq h} S \Rightarrow R_2$	Given a domain theory, assertions $\wedge_{1 \leq n} A_i$, and a boundary condition *B* leads to a divergence, specialize an object into disjoint subtypes and restrict (via a conditional) the applicability of the divergent assertions into disjoint conditions (cf., [229])

this technique to the resource boundary condition above yields the new goal.

∀ u:User, r:Resource
Requesting(u,r) ⇒ Reliable(r)

In words, all requested resources must be reliable. The other techniques of table 13 similarly modify the requirements. For example, Goal Restoration allows that a goal will fail; however, at sometime later, the conditions of the goal should be restored.

- ***Selection***. Resolutions are not explicitly represented in the KAOS framework, thus the selection process is outside the scope of the framework.

6.5 Deficiency-Driven Requirements Analysis

Since 1985 researchers at (and from) the University of Oregon have published papers on requirements and specification analysis[72]. The group has relied on a variety of techniques from artifi-

cial intelligence to construct automated assistants that critique requirements as part of a deficiency-driven design process. The original overarching project was named KATE, denoting Knowledge-Based Acquisition of Specifications[72]. Since, then various prototypes have been constructed (e.g., Critic[71], OPIE[4], OZ[215][221], SC[48]), the latest of which is Critter[66].

6.5.1 Products

The KATE project did not seek to define a requirements language. Rather, the project has adapted existing models (e.g., Numerical Petri Nets[66], Qualitative Physics[48], STRIPS predicates[4]) in order to meet the representation and reasoning needs of the automated assistant being constructed.

6.5.2 Process

As part of the KATE project, a variety of computerized support has been specified and constructed. The collective support that those prototypes could provide can be considered in terms of the processes described in section 4, as shown below. But first, the overarching process of KATE prototypes, deficiency-driven design, is summarized.

In *deficiency-driven design*, violations of system requirements or environmental constraints drive the state-based search that generates alternative designs through the applications of design operators. In KATE prototypes, a design is a detailed representation in which the interaction among selected system requirements and environmental constraints can be analyzed through some form of design simulation. The failure of a design to satisfy a system requirement or an environmental constraint is characterized as a deficiency. Deficiencies are remedied by the application of operators. A operator may: 1) add new agents, 2) reassign the responsibility for activities to different agents, 3) alter the communication among agents, or 4) weaken system requirement or environmental constraints. Operator application leads to a new design state, which can then be analyzed for deficiencies. The process stops when all requirements and environmental constraints are satisfied in a design.

- ***Partitioning***. For the most part, KATE prototypes do not address the partitioning of requirements prior to analysis. As part of deficiency-driven design, Critter does form a decomposition of requirements through agent assignment (cf., [59]). Such a decomposition does aid analysis, but it has not been used to reduced the scope of analysis.
- ***Identification***. Requirement interactions, specifically conflicts, are identified by simulating designs and relating deficiencies to requirements. For example, using the abstract planner OPIE, one can find a scenario of agent and environmental actions that lead to the satisfaction of a requirement. Conversely, one can find a failure scenario. (To do the analysis, a requirement is negated and specified as a OPIE goal). The conjunction of requirements can also be analyzed. In Oz, the conjunction of requirements held by different stakeholders was analyzed for stakeholder requirement interaction (using OPIE). Once it has been established that requirements conflict, Oz regressed the requirements through the scenario to identify the specific predicates that led to the conflict. The KATE project also demonstrated how qualitative requirement interactions can be discovered using qualitative physics envisonment (in SC). Finally, a case-based approach to interaction identification was demonstrated in Critic.
- ***Focus***. For the most part, conflicts are not explicitly represented in KATE prototypes. Oz is the exception, with its interference structure. While Oz interferences could be sorted by its attributes (e.g., stakeholders, requirements, relations, objects), it has not been used to automatically focus on a subset of conflicts. (Oz does use multi-criteria analysis to focus the resolution

generation process.)

- ***Resolution***. Oz and Critic provide resolution generation. Critter provides the Brinkmanship heuristic that applies to a specific type of "brink" constraint that is to be maintained.

  $T \Rightarrow o \neg \exists (x, y, z) \bullet P(x, z) \wedge P(y, z) \wedge x \neq y$

  In words, it says that (in all system behaviors) it should be the case that in the state that results from system transition **T** given a predicate (P) there exists no two domain values (x, y) that can be simultaneous found for the range value (z). A brink conflict occurs as the system transitions to a state in which the brink constraint fails. Below, is an instantiation of the brink conflict for a train control system.

  StartTrain $\Rightarrow o \neg \exists$ x, y: Train, z: Location $\bullet$ Location(x, z) $\wedge$ Location(y, z) $\wedge$ x $\neq$ y

  Given the (uncontrolled) StartTrain transition, it is possible to place two trains on the same track "block", possibly leading to a wreck. To resolve this conflict, the transition is replaced with a controlled transition: StartTrain-Controlled. In the controlled transition, if the brink condition can occur, then the controlled transition cannot execute; to ensure this, a design fragment is introduced. Generalizing this resolution technique, we have the following:

  AvoidBrinkTransition (a.k.a., AvoidBoundaryCondition)
  Given T, such that $T \Rightarrow o \neg \exists (x, y, z) \bullet P(x, z) \wedge P(y, z) \wedge x \neq y$
  Replace T with $T' \equiv$ **If** $P(x, z) \wedge P(y, z) \wedge x \neq y$ **Then** T **Else nil**

  In words, if a transition can lead to a constraint failure in the next state, then incorporate the constraint as a condition of the transition.

  Oz automates such conflict resolutions methods (as characterized in section 6.4.2) including object refinement, goal weakening via object generalization, compromise, and alternative goal refinement. Requirements DealMaker[228], a decedent of Oz, applies these same methods and adds avoid boundary condition, which is essentially AvoidBrinkTransition—as characterized in section 6.4.2

  Finally, Critic recognizes certain requirements patterns as bad (i.e., should be absent), while others as good (i.e., should be present). Thus, when it recognizes a design fragment it suggests elements that should be removed or added in order to satisfy the specified requirements, and remove the recognized conflicts.

- ***Selection***. Oz (and DealMaker[218]) provides an interactive search procedure that iteratively shows a multi-criteria evaluation of resolution alternatives and then allows for the application of new resolution methods. Initially, an analyst is presented with conflicts. Next, the analyst directs the application of a resolution generation method. In a multi-criteria graphic display, each alternative is linked under the conflict it resolves. (The display shows the satisfaction of various criteria in bar graphs.) Next, the analyst can choose one or more resolutions to replace a conflict, or he can continue applying generation methods. Each method can be applied to either the original conflict or any of the generated resolutions. The analyst stops search by asking Oz to substitute resolutions for each conflict, and then executing the Oz command CreatePerspective which forms the integrated stakeholder perspective.

6.6 Software Cost Reduction

Since 1978 researchers at (and from) the Naval Research Laboratory have published papers on software requirements and specification[102]. Originally introduced to help specify the A-7E software package, the Software Cost Reduction (SCR) project has been successful in specifying and analyzing large, real-time, embedded systems. In SCR, a system and its environment are formally

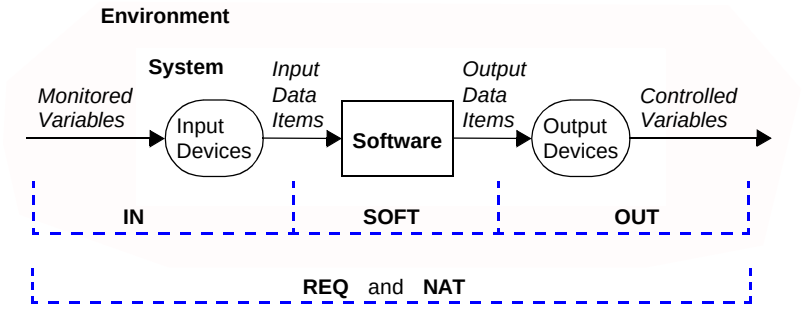


Figure 10.An illustration of SCR's four variable model.

modeled. Then, using the SCR toolset, one can analyze requirements for inconsistencies as well as check that specified properties hold (e.g., safety requirements).

SCR assumes a four variable model, as illustrated in figure 10[97]. In this model, input devices watch *monitored variables* of the environment and produce values, called *input data items*, that are processed by the software. The software then produces *output data items* that are processed by output devices that affect *controlled variables* of the environment. The IN, SOFT, and OUT relations specify the mapping of monitored values to input values, input values to output values, and output values to controlled values, respectively. The REQ relation specifies the relationship of monitored variables to controlled variables. Finally, NAT describes natural laws, or constraints, of the environment.

An SCR requirements specification consists of: 1) definitions for monitored, controlled, and intermediate *term* variables, 2) tables of transformations, T, that describe the software behavior (SOFT) by computing new variable values given changes to variables, and 3) properties of the system (REQ) and its environment (NAT). From such a description, the SCR toolset is able to check consistency within the requirements, as well as ascertain if specified properties (REQ) hold during the systems execution[97].

6.6.1 Products

The SCR language supports the four variable model. Thus, it provides constructs to define variables, transformations, and properties. In SCR,

> a system Σ is represented as a 4-tuple, $\Sigma = (S, S_0, E^m, T)$, where S is the set of states, $S_0 \subseteq S$ is the initial state set, E^m is the set of input events, and T is the transform describing the allowed state transitions. In the initial version of the SCR formal model, the transform T is deterministic, i.e., a function that maps an input event and the current state to a (unique) new state. Each transition from one state to the next state is called a *state transition* or, alternately, a *step*. To compute the next state, the transform T composes smaller functions, called *table functions*, [...] These tables describe the values of the dependent variables—the controlled variables, the mode classes, and the terms. Our formal model requires the information in each table to satisfy certain properties. These properties guarantee that each table describes a total function.—[97]

In SCR, transforms are defined tabular form. Each table defines how an event will change a variables value. An "*event* is a predicate defined on two consecutive system states that indicates a change in system state."[100] It is denoted $@T(c) \equiv \neg c \wedge c'$, where c is a condition that is false and then becomes true in the next state, denoted as c'.

As an example, consider a transition table for the variable InjectionPressure[14]. This table can be used to define when coolant can be injected into a pressurized container according to the water pressure in the container.

Table 14. A SCR Transition Table for InjectionPressure

Old Mode	Event	New Mode
TooLow	@T(WaterPressure ≥ Low)	Permitted
Permitted	@T(WaterPressure ≥ Permit)	High
Permitted	@T(WaterPressure < Low)	TooLow
High	@T(WaterPressure < Permit)	Permitted

The first row of table 14 indicates that the mode of InjectionPressure changes from TooLow to Permitted when WaterPressure ≥ Low becomes true. A set of such tables concerning typed variables provides the basis for an SCR requirements specification.

Behavioral descriptions, such as that of table 14, can be enhanced with property requirements, such as SoundAlarm ≡ @T(WaterPressure ≥ High) ⇒ AlarmSounding. In words, it indicates that AlarmSounding shall be true when WaterPressure ≥ High is true.[20] Such a requirement property does not specify the behavior of the software, rather it specifies a property that should always be true in the software (REQ). Thus, the property should be true in all states of the model defined in the 4-tuple, $\Sigma = (S, S_0, E^m, T)$. The SCR toolset can check the validity of such properties.

6.6.2 Process

As part of the SCR project, the SCR toolset has been constructed. It can be considered in terms of the processes described in section 4, as shown below.

- ***Partitioning***. Given a requirement property (e.g., SoundAlarm), the SCR toolset can use a "program slicing" technique to define only the relevant subset model of variables and transformations. Analysis on the reduced model can be much more efficient[97].
- ***Identification***. The SCR toolset identifies two types of requirement interactions[97]. First, a number of inconsistencies among requirements can be found through static analysis of the requirement definitions. In addition to the more common syntactic and type checking analysis, the SCR toolset can check that transformations are properly defined. This includes: 1) disjoint definitions among table rows for a transformation (i.e., $\wedge_{1 \le row}$ Condition$_{row} \Rightarrow$ false)[21], 2) complete coverage of a transformation by its table rows (i.e., $\vee_{1 \le row}$ Condition$_{row} \Rightarrow$ true)[22], and 3) reachability of all of a variable's modes from its initial mode[23]. The SCR toolset can also identify a second type of interaction that exists between a requirement property (e.g., SoundAlarm) and the software specification. It translates the SCR specification into the Spin model checker. Spin can then exhaustively explore all states in an attempt to show where the requirement property can fail[104]. The resulting Spin trace describes a scenario by which the failure can occur. This trace can be stepped through within the SCR toolset simulator; requirement properties that fail during such simulation are highlighted in the SCR specification.

[20] Notice that SCR semantics allow for the descriptions of properties on states or between two states. However, it does not have the full power of temporal logic, which also includes properties over sets of states (e.g., P ⇒ ◊Q).

[21] For example, if the third row of table 14 was defined as @T(WaterPressure ≤ Low), then rows one and three would be non-disjoint.

[22] For example, if the first row of table 14 was not included, then transition from the mode TooLow would not be defined..

[23] For example, if the third row of table 14 was not included, then transition to the mode TooLow would not be defined.

- ***Focus***. Inconsistencies are not explicitly represented in SCR. Thus, focusing on a subset of inconsistencies is outside the scope of the framework.
- ***Resolution***. Resolutions or resolution generation are outside the scope of SCR.
- ***Selection***. Selection among resolutions is outside the scope of SCR.

6.7 M-Telos

Two consecutive large European (ESPRIT) projects have focused on requirements engineering. From 1992 to 1995, the NATURE project (Novel Approaches to Theories Underlying Requirements Engineering) produced theories and tools for knowledge representation, domain engineering, and process engineering[112][113]. The success of NATURE led to the CREWS project, from 1996 to 1999. The CREWS (Cooperative Requirements Engineering with Scenarios) project developed, evaluated, and demonstrated the applicability of, methods and tools for cooperative scenario-based requirements elicitation and validation[115]. The results of both projects are too numerous to summarize here. Instead, we focus on one project, M-Telos, that defines a technique to manage requirements inconsistencies.

6.7.1 Products

As part of the CREWS project, M-Telos has been defined as a means to support multiple requirements perspectives (a.k.a. modules) and goal-oriented inter-perspective inconsistency management[182]. It is a formal framework and implementation supporting requirements development with a variety of notations, stakeholders views, or other requirements perspectives. As a means to support inter-perspective (and intra-perspective) analysis, the framework supports the following techniques.

- Separation of multiple partial models
- Dynamic definition and customization of notations
- Tolerance of conflicts
- Goal-orientation inter-perspective analysis
- Dynamic definition and customization of analysis goals

The M-Telos framework has been implemented in the ConceptBase, a deductive database that supports meta-modeling[111].

Figure 11 illustrates the M-Telos approach to perspective management. In the approach, requirement notations are instantiated from a central meta meta model. Requirement perspectives (in various notations) are encapsulated in modules. The characterization of requirement inconsistencies among modules is presented in “resolution” modules.

In figure 11, the module depicted in the top of the figure illustrates the centralized meta meta model for a requirements analysis project. For this example, it is important to note that it represents the following analysis goal (as constraints on the meta meta model).

Analysis Goal: Each exchanged MEDIUM must contain DATA

This is an analysis goal derived through stakeholder dialogs. In a business context, it means that documents exchanged should have information (i.e., they should not be content-free). The goal reflects the abstract classes from which the two requirement notation perspectives were developed. For example, the lower right module of figure 11 indicates the notation of requirement statements in that perspective; it contains statements structured as, “A FORM must include ITEMs.” Moreover, a FORM is an instance of MEDIUM and ITEM is an instance of DATA.[24] Similarly,

[24] Actual requirements that refer to instances of forms or items from the problem domain are not shown.

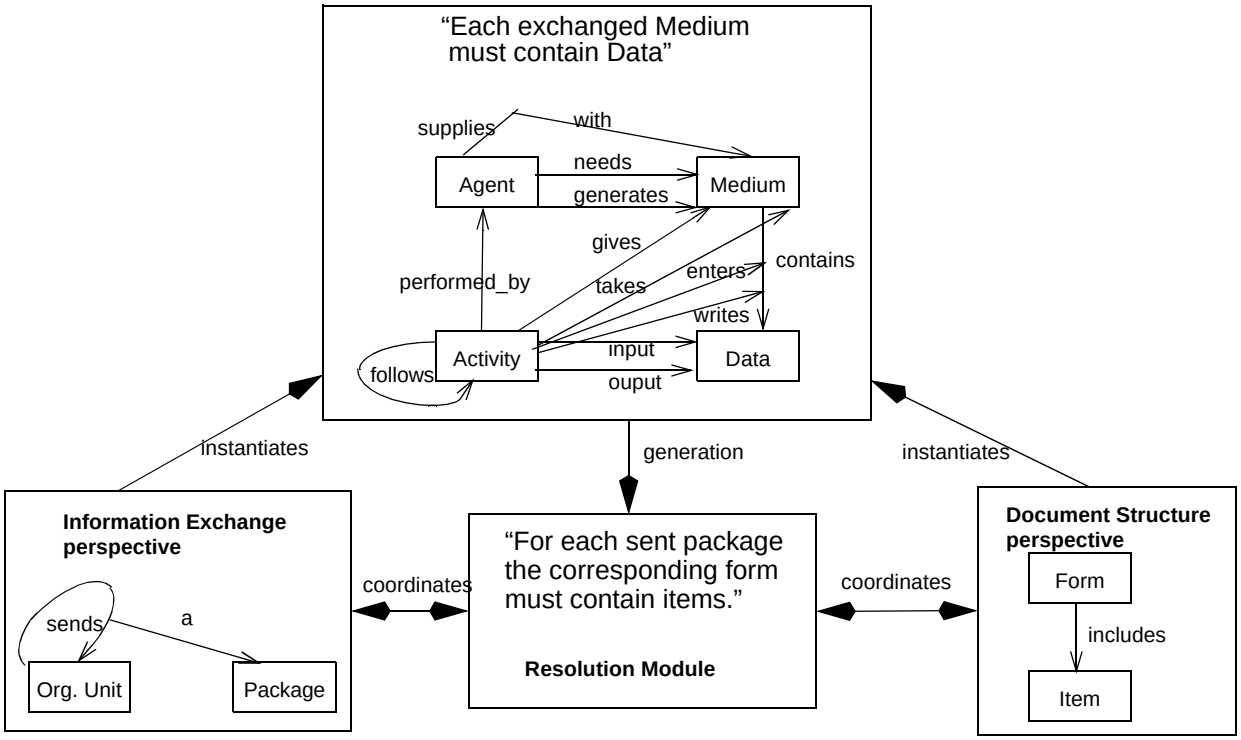


Figure 11.An example illustrating perspective management in M-Telos.

the lower left module of figure 11 instantiates the meta meta model with its requirement notation, "An ORGANIZATIONAL-UNIT sends PACKAGEs to other ORGANIZATIONAL-UNITs."

In figure 11, the resolution module is used to characterize inconsistencies that can occur between requirements in the two requirement modules. From the meta meta model and the notation descriptions in the two requirements modules, M-Telos automatically specializes the above analysis goal to be the following refined analysis goal.

Refined Analysis Goal: "For each sent PACKAGE the corresponding FORM must contain ITEMs."

Using the refined analysis goal, the resolution module is able to monitor the other two modules. Every time a requirement instance is defined using the notation of either module, the resolution module can check for violation of the refined analysis goal.

Inconsistency checking in M-Telos can be conceptualized in terms of a stream of transactions that add or delete requirements. After a new requirement is added (or deleted) an analysis goal may become violated. In M-Telos, this is called *primary inconsistency* with respect to the goal. In the case of adding objects, those M-Telos objects that were added and caused the inconsistency are considered *provisionally inserted objects*; deleting objects is analogous. As the result of a transaction, a goal's status may change from violated to satisfied. If the goal's satisfaction depends on objects have been provisionally inserted (or deleted), then it is called a *secondary inconsistency* with respect to that goal.

At the lowest level, inconsistency management becomes the task of computing goal satisfaction with respect to provisional objects. At a higher level, the user can specify the types of inconsistency that will be allowed: primary inconsistency (Allow/Not Allowed) and secondary inconsistency (Allow/Not Allowed). The combination provides for four levels of goal satisfaction. Grouping the two levels where some inconsistency is allowed gives us: *goal satisfaction*, *qualified*

goal satisfaction, and *goal violation*. Thus, the user of M-Telos can control what types inconsistency will be tolerated in requirement analysis goals, from no inconsistency to goal violations.

6.7.2 Process

Below, the M-Telos support is presented in terms of the processes described in section 4, with some references to related projects.

- ***Partitioning***. The M-Telos framework allows for the partitioning of requirements into any subsets an analyst chooses. The framework itself does not provide any categories; rather, it provides the views into which one can place requirements. However, in related work, a subject's situation parameters (agent, focus, notation, and time) are proposed as means to systematically defined bounded requirement sets[171].
- ***Identification***. Once requirements and analysis goals are represented in M-Telos, the automated system can derive the inconsistencies incrementally or on-demand. In related work, inconsistencies can be determined by comparing requirements against domain-independent analysis goals[253][254], or domain-dependent analysis goals[158], rather than M-Telos's project specific meta meta model and goals.
- ***Focus***. In M-Telos, inconsistencies are associated as goal violations. It has not been demonstrated how the framework could prioritize the further analysis or the resolution process. However, intergoal analysis goals could be developed to specify the ordered application of consistency rules
- ***Resolution***. There is no resolution generation technique in M-Telos. Rather, analysis goals are used to identify inconsistencies, which are then represented in a "resolution" module. An interesting extension may be to show users provisional objects for qualified or violated analysis goals. This is similar to finding the causes of conflict through goal regression in operator-oriented requirements (cf. section 6.4). With such a inconsistency context, users could generate alternative resolutions; for example, avoiding a boundary condition found among the provisional objects (cf. section 6.4).
- **Selection**. Selection among resolutions is outside the scope of M-Telos.

7 CONCLUSIONS

This article has presented an introduction to requirements interaction management. An evolution of supporting concepts and their related literature were presented. Process and product issues were summarized. Using the issues, seven research projects exemplifying requirements interaction management were presented.

As this article has shown, requirements interaction management has become a critical area of requirements engineering. Research in this area is growing. Many current research efforts are focused on automated techniques that support the identification and focus on undesirable requirement interactions. Other lesser explored areas include requirement partitioning, automated resolution generation, resolution selection, and the integration of requirements interaction management into the larger development life-cycle. As these problems are solved, requirements interaction management will become a more integral part of development. As a result, we expect to observe the development of systems with higher stakeholder satisfaction and fewer failures.

ACKNOWLEDGMENTS

This article has been improved through the generous support of the following people who have commented on earlier drafts: Martin Feather ...

REFERENCES

[1] AAAI, Workshop on Models of Conflict Management in Cooperative Problem Solving, AAAI, Seattle, Wa, August 4, 1994.

[2] Adler, M., Davis, A., Weihmayer, R., and Worrest, R., *Conflict-resolution strategies for nonhierarchical distributed agents*, Morgan Kaufmann Publishers Inc.(1989).

[3] Anderson, J. S. and Farley, A. M. Plan Abstraction Based on Operator Generalization. AAAI, *Proceedings of the 1988 AAAI National Conference on Artificial Intelligence*, Morgan Kaufmann, St.Paul, Minnesota, 1988, 100-104.

[4] Anderson, J., Fickas, S., V*iewing Specification Design as a Planning Problem: A Proposed Perspective Shift*, In 5th International Workshop on Software Specification and Design, Pittsburgh, 1989 (Also in *Artificial Intelligence and Software Engineering*, D. Partridge (ed), Ablex, 1991.)

[5] Atzeni, P., and R. Torlone. "A Metamodel Approach For The Management of Multiple Models and The Translation of Schemes," *Information Systems*, Vol. 18, No. 6, 349-362, 1993.

[6] Avison, D.E., Wood-Harper, A.T., Multiview: An Exploration in Information Systems Development, Blackwell Scientific Publications, 1990.

[7] R. Balzer, Tolerating Inconsistency, Proc. of 13th Int. Conf. on Software Engineering (ICSE-13), IEEE CS Press., Austin, Texas, USA, 13-17th May 1991, 158-165.

[8] Barbacci, M., Longstaff, T.H., Klein, M.H., Weinstock, C.B., Quality Attributes, Software Engineering Institute, CMU/SEI-96-TR-036, March, 1997.

[9] Barbacci, M., Klein, M.H., Weinstock, C.B., Principles for Evaluating Quality Attributes of Software Architecture, Software Engineering Institute, CMU/SEI-95-TR-021, December, 1995.

[10] Barki, H., Hartwick, J., Rethinking the Concept of User Involvement, MIS Quarterly, 13 (1), 1989, pp. 53-61.

[11] Batini, C., Lenzerini, M., Navathe, S.B., A Comparative Analysis of Methodologies for Database Schemea Integration, ACM, Computing Surveys, 18 (4), December, 1986 pp. 323- 364.

[12] Bendifallah, S., and Scacchi, W., Work structures and shifts: An empirical analysis of software specification teamwork. IEEE, *Proceedings of the 11th International Conference on Software Engineering*, 1989, pp. 260-270.

[13] Bharadwaj, Ramesh and Heitmeyer, C.L., "Model Checking Complete Requirements Specifications Using Abstraction," NRL Memorandum Report NRL/MR/5540--97-7999, November 10, 1997.

[14] Bharadwaj, Ramesh and Constance L. Heitmeyer. "Model Checking Complete Requirements Specifications Using Abstraction," NRL Memorandum Report NRL/MR/5540--97-7999, November 10, 1997

[15] Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MaCleod, G.J., Merritt, M.J., Characteristics of Software Quality, North-Holland, New York, 1978.

[16] Barry Boehm, Alexander Egyed, Julie Kwan, Dan Port, and Archita Shah , Ray Madachy, Using the WinWin Spiral Model: A Case Study, IEEE, *Computer*, July 1998.

[17] Barry Boehm and Alexander Egyed, WinWin Requirements Negotiation Processes: A Multi-Project Analysis, *Proceedings of the 5th International Conference on Software Processes*, 1998

[18] Boehm, P. Bose, E. Horowitz and M. J. Lee, Software Requirements as Negotiated Win Conditions, *First International Conference on Requirements Engineering,* IEEE, (April 18-22 1994).

[19] Boehm, B., In, H., Identifying Quality-Requirement Conflicts, IEEE, Software, March, 1996, 25-36.

[20] Boehm, B., Software Engineering Economics, Printice-Hall, Englewood Cliffs, N.J., 1981.

[21] Boehm, Barry, W., A Spiral Model of Software Development and Enhancement, in Computer, Vol. 21, No. 5, 1988., pp. 61-72.

[22] Boehm, B.W. and Ross, R. "Theory W Software Project Management: Principles and Examples," IEEE Transactions on Software Engineering, July 1989, pp.902-916

[23] Boehm, Barry W., Seven Basic Principles of Software Engineering, *Journal of Systems & Software*, 3(1) | March 1983, pp. 3-24.

[24] Botten, N., Kusiak, A., Raz, T., Knowledge Bases: Integration, Verification, and Partitioning, European Journal of Operational Research, 42, (2) Sep 25, 1989, pp. 111-128.

[25] F.P., Jr. Brooks, No silver bullet: essence and accidents of software engineering IEEE, Computer, April 1987, 10-19, (20) 4.

[26] Buchheit, M., Jeusfeld, M.A., Nutt, W., Staudt, M., Subsumption between queries to object-oriented databases. Information Systems, 19, 1, March 1994
[27] Buchanan, B.G., Shortliffe, E.H., (Eds.), Rule-Based Expert Systems: The MYCIN experiments of the Stanford heuristic programming project, Addison-Wesley, 1984.
[28] Callahan, J.R., Montgomery, T.L., An Approach to Verification and Validation of a Reliable Multicasting Protocol, International Symposium on Software Testing and Analysis (ISSTA'96), San Diego, CA, 8-10 January, 1996, pp. 187-194.
[29] Checkland, P., Systems Thinking, Systems Practice, John Wiley & Sons, 1981.
[30] Chen H., Lynch, K. J., Automatic construction of networks of concepts characterizing document atabases. IEEE Transactions on Systems, Man and Cybernetics, 22(5):885-902, September/October 1992.
[31] Chen H., Lynch, K. J., Basu, K., and Ng, T., Generating, integrating, and activating thesauri for concept-based document retrieval. IEEE EXPERT, Special Series on Artificial Intelligence in Text-Based Information Systems, 8(2):25-34, April 1993.
[32] Chen, M. and Nunamaker, J., The architecture and design of a collaborative environment for systems definition, *Data Base*, Winter/Spring 1991, 22-28.
[33] Chikofsky, E.J., Rubenstein, B.L., CASE: Reliability Engineering for Information Systems, in *Computer Aided Software Engineering (CASE)*, Ed. Chikofsky,E., IEEE, 1993, pp. 147-153.
[34] Christel, M.G., Wood, D.P., Stevens, S.M., AMORE: The Advanced Multimedia Organizer for Requirements Elicitation, Software Engineering Institute, CMU/SEI-93-TR-12, June 1993.
[35] Chung, L., Nixon, B., Yu, E., Using Non-Functional Requirements to Systematically Support Change, IEEE, Second International Symposium on Requirements Engineering, March 27-29, 1995, pp. 132-139.
[36] Chung, L., Nixon, B., Yu, E., Dealing with Change: An Approach Using Non-Functional Requirements, Springer-Verlag, *Requirements Engineering Journa*l, (1), 1996, pp. 238-260.
[37] Chung, L., Nixon, B., Yu, E., Using Quality Requirements to Systematically Develop Quality Software, Fourth Conference on Software Quality, October 3-5, McLean, VA, 1994.
[38] Conry, S.E., Kuwabara K., Lesser, V.R., Meyer, R.A., and , Multistage negotiation for distributed satisfaction, IEEE, *Transactions on systems, man, and cybernetics*, Vol. 21, No. 6, November/December 1991, 1462-1477.
[39] Conry, S., Meyer, R., and Lesser, V., Multi-stage negotiation in distributed planning, Eds. A.H. Bond, L. Gasser, *Readings in distributed artificial intelligence*, Morgan Kaufmann , San Meteo, California (1988) 367-384.
[40] Cugola, G., Di Nitto, E., Guggetta, A., Ghezzi, C., A Framework for Formalizing Inconsistencies and Deviations in Human-Centered Systems, ACM, *Transactions on Software Engineering and Methodology*, 5 (3), 1996, pp. 191-230.
[41] Curtis, B., Krasner, H., and Iscoe, N., A field study of the software design process for large systems, *Communications of the ACM*, Vol. 31, No. 11 (November 1988), 1268-1287.
[42] Dardenne, A., van Lamsweerde, A., Fickas, S., Goal-Directed Requirements Acquisition, *Science of Computer Programing*, 20 1993, 3-50.
[43] Darimont, R, van Lamsweerde, A., Formal Refinement Patterns for Goal-Driven Requirements Elaboration, ACM SIGSOFT, *Fourth Symposium on the Foundations of Software Engineering*, San Francisco, CA, October 16-18, 1996.
[44] Darimont, R, Delor, E., Massonet, P., van Lamsweerde, A.,GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering, IEEE, *Proceedings of the 20th Interactional Conference on Software Engineering*, Kyoto, April 1998, Vol 2, pp. 58-62.
[45] Davis, A., *Software Requirements: Objects, functions, and states*, Prentice Hall, 1993.
[46] Davis, R., Meta-Rules: Reasoning About Control, North-Holland, *Artificial Intelligence*, 15, 1980, 179-222.
[47] Dill, D.L., Drexler, A.J., Hu, A.J., Yang, C.H., Protocol Verification as a Hardware Design Aid, IEEE International Conference on Computer Design: VLSI in Computers and Processors, IEEE Computer Society, 1992, pp. 522-525
[48] Downing, K., Fickas, S., *A Qualitative Modeling Tool for Specification Criticism*, Conceptual Modelling, Databases, and CASE: An Integrated View of Information Systems Development, Peri Loucopoulos (ed), Ablex, 1991
[49] Drucker, P.F., The Practice of Management, Harper & Row, New York, 1954.
[50] Durfee, E.H., Coordination of distributed problem solvers, Kluwer Academic Publishers, Boston, 1988.
[51] Easterbrook, S., "Domain Modeling with Hierchies of Alternative Viewpoints, IEEE, International Symposium on Requirement Engineering, San Diego, CA, January 4-6, 1993, 65-72.
[52] Easterbrook S., Co-ordinating distributed ViewPoints: the anatomy of a consistency check, Concurrent Engineering: Research & Applications, CERA Institute, (2), 1994.

[53] S. Easterbrook and B. Nuseibeh, Using ViewPoints for Inconsistency Management, *Software Engineering Journal*, 11(1): 31-43, BCS/IEE Press, January 1996.
[54] Egyed, A., Boehm, B., USC, *Analysis of Software Requirements Negotiation Behavior Patterns*, USC-CSE-96-504, 1996.
[55] Alexander Egyed and Barry Boehm, A Comparison Study in Software Requirements Negotiation, Proceedings of the 8th Annual International Symposium on Systems Engineering (INCOSE'98), 1998.
[56] Emmerich, W., Finkelstein, A., Montangero, C. & Stevens, R. "Standards Compliant Software Development" in *Proc. International Conference on Software Engineering Workshop on Living with Inconsistency*, (IEEE CS Press), 1997
[57] Feather, M.S., FLEA : Formal Language for Expressing Assumptions Language Description, June 25, 1997.
[58] Feather, M., Constructing specifications by combining parallel elaborations, *IEEE Transactions on Software Engineering* Vol. 15 (February 1989)
[59] Feather, M., Language Support for the Specification and Development of Composite Systems, *ACM Trans. on Programming Languages and Systems*, 9 (2) , April 1987, pp. 198-234.
[60] Feather, M., Fickas, F., Robinson W., *Design as Elaboration and Compromise*, Workshop Notes from the Sixth National Conference on Artificial Intelligence, *Automating Software Design*, Kestrel Institute, AAAI, St. Paul, MN, August 25, 21-22, 1988.
[61] Feather, M.S., Fickas, S., van Lamsweerde, A., Requirements and Specification Exemplars, *Automated Software Engineering*, Kluwer, 4 (4), 1997.
[62] Feather, M.S., Fickas, S., van Lamsweerde, A., Ponsard, C., Reconciling System Requirements and Runtime Behavior, *Proceedings of the International Workshop on Software Specification and Design (IWSSD 98)*, Isobe, IEEE CS Press, April, 1998.
[63] Festinger, L., Conflict, Decision, and Dissonance, Tavistock Publications, Ltd., London, 1964.
[64] Fickas, S., Automating the Transformational Development of Software, IEE, Transactions on Software Engineering, 11 (11), November, 1985, pp. 1268-1277.
[65] Fickas S., *Supporting the programmer of a rule based language*, Expert Systems, 4(2), May 1987.
[66] Fickas, S., Helm, R., *Knowledge representation and reasoning in the design of composite systems*, IEEE, Transactions on Software Engineering, Special issue on knowledge representation and reasoning, June, 1992.
[67] Fickas, S., Anderson, J., Robinson, W.N., *Formalizing and automating requirements engineering*, CIS-TR-90-03, University of Oregon, April 6, 1990.
[68] Fickas, S., Downing, K., Novick, D., Robinson, B., The Specification, Design, and Implementation of Large Knowledge-Based Systems, IEEE, Artificial Intelligence in the Northwest, Northcon/85, Portland, OR, October 22-24, 1985, 8/2.
[69] Fickas, S., Feather, M.S., Requirements Monitoring in Dynamic Environments, *Proceedings of the 2nd International Symposium on Requirements Engineering*, IEEE Computer Society Press, York, England (March 1995) 140-147.
[70] Fickas, S., Robinson, W., Feather, M., Conflict and compromise in specification design, In *Proceedings of the AAAI-88 Automated Software Development Workshop*, Minneapolis, 1988
[71] Fickas, S., Nagarajan, P., Being suspicious: critiquing problem specifications, Morgan Kaufmann Publishers Inc., Proceedings of the 7th National Conference on Artificial Intelligence, August 1988, pp. 19-24.
[72] Fickas, S., A knowledge-based approach to specification acquisition and construction, CIS-TR-85-13, University of Oregon, November 1985.
[73] Finkelstien, A. (Ed), Viewpoints 96: An International Workshop on Multiple Perspectives in Software Development, *ACM Symposium on Foundations of Software Engineering*, San Francisco, USA, October 14th & 15th, 1996.
[74] Finkelstein, A., Kramer, J., & Nuseibeh, B. (Eds.). (1994b). Software Process Modelling and Technology. Somerset, UK: Research Studies Press Ltd. (Wiley).
[75] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein and M. Goedicke, Viewpoints: A Framework for Multiple Perspectives in System Development, International Journal of Software Engineering and Knowledge Engineering, Special issue on 'Trends and Future Research Directions in SEE', World Scientific Publishing Company Ltd., 2(1): 31-57, March 1992.
[76] Finkelstein, A., Fuks, H., Multi-party specification, *5th International workshop on software specification and design*, (1989) 185-195.
[77] Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., & Nuseibeh, B., “Inconsistency Handling In Multi-Perspective Specifications” IEEE *Transactions on Software Engineering*, 20, 8, (1994), 569-578.
[78] Fuks, H., Negotiation using commitment and dialogue, Imperial College of Science Technology and Medicine, London(February,1991)
[79] Fisher, R., William, U., with Bruce Patton, editor. Getting to Yes: Negotiating Agreement Without Giving In, Penguin Books, New York, 1991.

[80] Fox, M.S., Gruninger, M., (1997), "On Ontologies and Enterprise Modelling", International Conference on Enterprise Integration Modelling Technology 97, Springer-Verlag, to appear.
[81] Fox, M.S., Barbuceanu, M., and Gruninger, M., (1996), "An Organisation Ontology for Enterprise Modelling: Preliminary Concepts for Linking Structure and Behaviour", Computers in Industry, Vol. 29, pp. 123-134.
[82] Francalanci, C., Fuggetta, A., Integrating Conflicting Requirements in Process Modeling: A Survey and Research Directions, Information and Software Technology, Elsevier, 39 (3) 1997 pp. 205-216.
[83] Gasser, L., Huhns, M.N., (Eds), Distributed Artificial Intelligence, Morgan Kaufmann Publishers Inc, San Mateo, 1989.
[84] Gasser, L., Huhns, M.N., (Eds), Distributed Artificial Intelligence Vol II, Morgan Kaufmann Publishers Inc, San Mateo, 1989.
[85] Georgeff, M.P., A theory of action for multiagent planning, Proceedings of 1984 conference of the AAAI, Morgan Kaufmann Publishers, 1984, pp. 121-125.
[86] Gilb, T., Software Metrics, Winthrop Publishers, Cambridge, MA 1977.
[87] Gotel, O. & Finkelstein, A.; "Contribution Structures" in Proc. 2nd International Symposium on Requirements Engineering RE95, (IEEE CS Press) 1995, 100-107
[88] Greenspan, G. Mylopoulos, J., Borgida A., : On formal requirements modeling languages: RML revisited. In Proc. 16th International Conference on Software Engineering, 1994, pp. 135-148
[89] Graf, D.K., Misic, M.M., The Changing Roles of the Systems Analyst, *Information Resources Management Journal*, 7 (2), Spring 1994, pp. 15-23.
[90] Gruninger, M., and Fox, M.S. (1995), "Methodology for the Design and Evaluation of Ontologies", Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95, Montreal.
[91] Guarino, N., The Ontological Level, Eds., R. Casati, B. Smith and G. White, *Philosophy and the Cognitive Sciences*, Vienna: Hölder-Pichler- Tempsky, 1994.
[92] R. Gustas, On Related Pragmatic Categories and Dependencies within Enterprise modelling, Second Scandinavian Research Seminar on Information and Decision Networks, Vaxjo University, Sweden, May 1995. (Also available as NATURE-95-13 from RWTH Aachen, Informatik V - Information Systems.)
[93] U. Hahn, M. Jarke, and T. Rose. Teamwork support in a knowledge-based information systems environment. IEEE Transactions on Software Engineering, 17(5):467-482, May 1991.
[94] Willie Hammer, Product Safety Management and Engineering, Printice-Hall, Inc., Englewood Cliffs, N.J., 1980.
[95] Hauser, J.R. & Clausing, D., "The House of Quality", *Harvard Business Review*, May-June, 1988, pp. 63-73.
[96] Hearst, M., Information integration, IEEE Intelligent Systems, Vol. 13, No. 5, September/October 1998 pp. 12-24.
[97] Heitmeyer C.L., Jeffords, R.D., and Labaw, B.G., "Automated Consistency Checking of Requirements Specifications," ACM *Trans. on Software Eng. and Methodology* 5, 3, July 1996, 231-261.
[98] Heitmeyer, C., Mandrioli, D., Formal Methods for Real-time Computing: An Overview, in *Formal Methods for Real-time Computing*, Heitmeyer, C., Mandrioli, D., Eds., Chichester, UK, J. Wiley, 1996, pp. 1-32.
[99] Heitmeyer, Constance L., Kirby, James Jr., Labaw, Bruce, Archer, Myla, Bharadwaj, Ramesh, "Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications," IEEE, *Transactions on Software Engineering*, vol. 24, no. 11, November 1998.
[100] Heitmeyer, Constance, Kirby, James, and Labaw, Bruce. "Applying the SCR Requirements Method to a Weapons Control Panel: An Experience Report.", Formal Methods in Software Practice '98, Clearwater Beach, FL, March 4-5, 1998
[101] Heninger, K.L., Specifying Software Requirements for Complex Systems: New Techniques and their Application, IEEE, *Transactions on Software Engineering*, 6, pp. 2-13, 1980.
[102] Heninger, K., Parnas, D. L., Shore, J. E., and Kallander, J. W. Software requirements for the A-7E aircraft. Tech. Rep. 3876, Naval Research Lab., Wash., DC, 1978.
[103] Heym, M., and H. Osterle. “Computer-aided Methodology Engineering,” *Information and Software Technology*, Vo. 35, no. 6/7, 345-353, June/July 1993.
[104] Holzmann, G.J., The Model Checker Spin, IEEE, Transactions on Software Engineering, 23, 1997, pp. 279-295.
[105] Høydalsvik,G.M., Sindre, G., On the Purpose of Object-Oriented Analysis. *Proc OOPSLA 93*, 1993, pp. 240-255.
[106] Horwitz, S., Prins, J., and Reps, T., Integrating non-interfering versions of programs. ACM Transactions on Programming Languages and Systems 11, 3 (July 1989), 345-387
[107] A. Hunter and B. Nuseibeh, Managing Inconsistent Specifications: Reasoning, Analysis and Action, ACM Transactions on Software Engineering and Methodology, October 1998

[108] Jacobs, S., Kethers, S., Improving Communication and Decision Making within Quality Function Deployment, *First International Conference on Concurrent Engineering, Research, and Application*, Pittsburgh, USA, August, 1994.
[109] Janis, I., Mann, L., Decision Making : a Psychological Analysis of Conflict, Choice, and Commitment, The Free Press, New York, 1979.
[110] Jantsch, Technological Forecasting in Perspective, Organization for Economic Co-operation and Development, Paris, 1967.
[111] Jarke, M., Gallersdorfer, R., Jeusfeld, M.A., Staudt, M., Eherer, S., ConceptBase - a Deductive Object Manager for Meta Data Management, Journal of Intelligent Information Systems, 4 (2), March 1995, 167-192.
[112] M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, Y. Vassiliou, Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis, First Intl. Symp. on Requirements Engineering, San Diego, USA, 1993, pp. 19-31.
[113] M. Jarke, K. Pohl, R. Dömges, S. Jacobs, H.W. Nissen, Requirements Information Management: The NATURE Approach, Ingenerie des Systemes d'Informations (Special Issue on Requirements Engineering), Vol.2, No. 6, 1994.
[114] Jarke, M., Pohl, K., Requirements engineering in 2001: (Virtually) managing a changing reality, *Software Engineering*, November 1994, pp. 257-266.
[115] Jarke, M., CREWS: Towards Systematic Usage of Scenarios, Use Cases and Scenes, Matthias Jarke, WI (Wirtschaftsinformatik) 99, Saarbrücken, 3.-5. März 1999, Springer Aktuell, (Also, available as CREWS-99-02 from RWTH Aachen, Informatik V - Information Systems.)
[116] Jelassi, M.T., Foroughi, A., *Negotiation Support Systems: An Overview of Design Issues and Existing Software,* North-Holland, Decision Support Systems, (5), 1989, 167-181.
[117] Jeusfeld, M.A., Johnen, U.A. : An executable meta model for re-engineering of database schemas. Proc. 13th International Conference on Conceptual Modeling (ER'94), Manchester, UK, Dec. 1994
[118] P. Johannesson, M. Hasan Jamil, Semantic Interoperability - Context, Issues, and Research Directions, Second Intl. Conf. on Cooperating Information Systems, Toronto, Canada, May 1994.
[119] Jones, C. Patterns of Software Systems Failure and Success, International Thomson Computer Press, 1996.
[120] Jones, C., Software Challenges, *Computer*, Vol. 28, No. 10, October 1995.
[121] Jourdan, J., Dent, D., McDermott, J., Mitchell, T., and Zabowski, D., Interfaces that learn: a learning apprentice for calendar management," CMU-CS-91-135, Carnegie Mellon University (May 7, 1991).
[122] Kannapan, S.M., Marskek, K.M., “An Approach to Parametric Machine Design and Negotiation in Concurrent Engineering”, Concurrent Engineering: Automation, Tools, and Techniques, Edited by A. Kusiak, John Wiley & Sons, 1993, 509-534.
[123] Kant, E., Barstow, D., The refinement paradigm: The interaction of coding and efficiency knowledge in program synthesis, IEEE, Transactions on Software Engineering, SE-7 (5) September 1981.
[124] Karlsson, J., Ryan, K., A Cost-Value Approach for Prioritizing Requirements, IEEE, *Software*, September, 1997.
[125] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J., The Architecture Tradeoff Analysis Method ICSE-98, 1998.
[126] Keeney, R., Raiffa, H., *Decisions with multiple objectives*, John Wiley and Sons, New York(1976).
[127] Kersten, G.E., Szpakowicz, S., Negotiation in Distributed Artificial Intelligence: Drawing from Human Experience, IEEE, *Proceedings of the 27th Annual Hawaii International Conference on Systems Sciences*, 1994, 258-270.
[128] Kim, E., Lee, J., An Exploratory Contingency Model of User Participation and MIS Use, Information and Management, 11 (2), 1986, pp. 87-97.
[129] Kimbrough, S., Lee, R.M., On Formal Aspects of Electronic (or Digital) Commerce: Examples of Research Issues and Challenges, IEEE, *Proceedings of the 29th Annual Hawaii International Conference on Systems Sciences*, 1996, 319-328.
[130] Klein, M., Supporting Conflict Resolution in Cooperative Design Systems, IEEE, *Transactions on Systems, Man, and Cybernetics*, 21 (6), November 1991, 1379-1390.
[131] Krasner, Herb, Curtis, Bill, and Iscoe, Neil, Communication breakdowns and boundary spanning activities on large programming projects, from *Empirical Studies of Programmers*: Second Workshop (edited by Gary M. Olson, Sylvia Sheppard, and Elliot Soloway), Ablex Publishing Corporation, Norwood, NJ, 47-64. (Conference held in Washing, D.C. on 12/7-8, 1987)
[132] Krasner, H., Requirements dynamics in large software projects, A perspective on new directionsin the software engineering process, *Proc. IFIP*, Elsevier, New York, pp. 211-216.
[133] Kraus, S., Wilkenfeld, J., The function of time in cooperative negotiations, AAAI, Proceedings of Ninth National Conference on Aritficial Intelligence, July 14-19, Vol 1, pp. 179-184.

[134] Kraus, S., Lehmann, D., Designing and Building a Negotiating Automated Agent, *Computational Intelligence*, 11(1):132-171, 1995
[135] Kusiak, A., (ed), *Concurrent engineering: automation, tools, and techniques*, John Wiley & Sons, 1993.
[136] Kwa, J., Tolerant planning and negotiation ingenerating coordinated movement plans in an automated factory," *Proceedings of the first international conference on industrial and engineering applications of artificial intelligence*, (1988)
[137] Kumar, K., and R. J. Welke. "Methodology Engineering: A Proposal for Situation-Specific Methodology Construction," in *Challenges and Strategies for Research in systems Development* (eds. W. W. Cotterman and J. A. Senn), John Wiley and Sons Ltd., 257-269, 1992
[138] Ladkin, P., In *The Risks Digest*, Neumann, P.G. (Ed.), ACM, 15.13, October, 1995.
[139] Ladkin, P., In *The Risks Digest*, Neumann, P.G. (Ed.), ACM, 15.30, December, 1995.
[140] Lander, S., Lesser, V., A framework for the integration of cooperative knowlege-based systems, *Workshop on integrated architectures for manufacturing*, IJCAI , Detroit, Michigan (August 24, 1989)
[141] Lander, S.E., Lesser, V.R., Understanding the Role of Negotiation in Distributed Search Among Heterogeneous Agents, IJCAI, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, France, August, 1993, 438-444.
[142] Leite, Julio Cesar Sampaio do Prado, Freeman, P., : Requirements Validation Through Viewpoint Resolution. IEEE, Transactions on Software Engineering, 1253-1269.
[143] Lempp, P., Rudolf, L., What Productivity Increases to Expect from a CASE Environment: Results of a User Survey, in *Computer Aided Software Engineering (CASE)*, Ed. Chikofsky, IEEE, 1993, pp. 147-153.
[144] Lenat, D.B., Brown, J.S., Why AM and Eurisko Appear to Work, *Proceedings of the Third National Conference on Artificial Intelligence*, August 22-26, 1983, Washington, D.C., AAAI, pp. 236-240.
[145] Leventhal, N., Using Groupware to Automate Joint Application Development, Journal of Systems Management, 45 (5), September/October 1995, pp. 16-22.
[146] Leveson, N. G., Safeware: System Safety and Computers, Addison-Wesley Pub. Co. Inc., 1995.
[147] Leveson, N. G., Heimdahl, M.P.E., Hildreth, H., Reese, J.D., Requirement Specification for Process Control Systems, IEEE, *Transactions on Softare Engineering*, 20, 1984.
[148] Lim, L., Benbasat, I., A Theoretical Perspective of Negotiation Support Systems, Journal of Management of Information Systems, Winter, 9 (3) 1992-1993, 27-44.
[149] Lions, J.L., *ARIANE 5: Flight 501 Failure*, Report by the Inquiry Board, European Space Agency, Paris, July 19, 1996.
[150] Liou, Y.I., Chen, M., Using Group Support Systems and Joint Application Development for Requirements Specification, *Journal of Management Information Systems*, 10(3), Winter 1993-1994, pp. 25-41.
[151] F. X. Liu and J. Yen, An Analytic Framework for Specifying and Analyzing Imprecise Requirements, in Proceedings of 18th International Conference on Software Engineering (ICSE-18) , Berlin, Germany, pp 60-69, March 25-30, 1996.
[152] London, P.E., Feather, M.S., Implementing Specification Freedoms, North-Holland, Science of Computer Programming, 2, 1982, 91-131.
[153] Lubars, M., Potts, C., Richter, C., A review of th state of practice in requirements modeling, *First International Symposium on Requirements Engineering*, IEEE, January 4-6 1993.
[154] Lutz, R.R., Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems, *First International Symposium on Requirements Engineering*, IEEE, January 4-6 1993, pp. 126-133.
[155] Lyytinen, K., Hirschheim, R., Information systems failures—a survey and classification of the emperical literature, Oxford Surveys in Information Technology, Vol. 4, Oxford University Press, 1987, pp. 257-309.
[156] MacCrimmon, K. R. & Taylor, R. N., Decision making and problem solving. In M.D. Dunnette (Ed.), *Handbook of Industrial and Organizational Psychology* Chicago, Il: Rand McNally College Publishing Company, 1976, pp. 1397-1453.
[157] Maiden, N. Minocha, S. Ryan, M., Hutchings, K., Manning, K., A Co-operative Scenario-based Approach to the Acquisition and Validation of Systems Requirements, in Proceedings of Human Error and Systems Development, Glasgow University, Scotland, March 19-22, 1997.
[158] N.A.M. Maiden, A.G. Sutcliffe Requirements Critiquing Using Domain Abstractions, *Intl. Conf. on Requirements Engineering, Colorado-Springs*, USA, April 1994
[159] N.A.M. Maiden, CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements, Kluwer, Journal of Automated Software Engineering, 1998.
[160] Magal, S., Snead, K., The Role of Causal Attributions in Explaining the Link Between User Participation and Information System Success, Information Resources Management Journal, 6 (3), Summer 1993, pp. 19-29.
[161] Manna, Z., Prueli, A., The Temporal Logic of Reactive and Concurrent Systems, Springer-Verlag 1992.
[162] Manna, Z., Prueli, A., Tools and rules for the practicing verifier, Department of Computer Science, Stanford University, Technical Report, CS-TR-90-1321. 1990.

[163] Markus, L., Keil, M., If We Build It, They Will Come: Designing Information Systems That People Want to Use, *Sloan Management Review*, Summer, 1994, pp. 11-25.
[164] Matwin, S., Szpakowicz, S., Koperczak, Z., Kersten, G.E., Michalowski, W., *Negoplan: An Expert System Shell for Negotiation Support*, IEEE, Expert, 4(4) 50-62.
[165] Mazer, M., A knowledge-theoretic account of negotiated commitement," CSRI-237, Univerity of Toronto (November 1989).
[166] Mazza, C., Fairclough, J., Melton, B., De Pablo, D., Scheffer, A., Stevens, R., Software Engineering Standards, Prentice Hall, 1994.
[167] McKeen, J., Guimaraes, T., Successful Strategies for User Participation in Systems Development, JMIS, 14 (2), Fall 1997, pp. 133-150.
[168] McMillan, K.L., Symbolic Model Checking - an Approach to the State Explosion Problem, TR, Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1992.
[169] Mi, P, Scacchi, W, Process Integration for CASE Environments, IEEE *Software*, Vol. 9(2), 45-53, (March 1992). Reprinted in Computer-Aided Software Engineering (Second Edition), E. Chikovsky (ed.), IEEE Computer Society (1993).
[170] Miller, J., Palaniswami, D., Sheth, A., Kochut, K., Singh, H., WebWork: METEOR's Web-based Workflow Management System, Technical Report #UGA-CS-TR-97-002, Department of Computer Science, University of Georgia, March 1997.
[171] R. Motschnig-Pitrig, H.W. Nissen, M. Jarke, Proc. of the 9th Intl. Conf. On Software Engineering and Knowledge Engineering (SEKE ’97), Madrid, Spain, June 17-20, 1997.
[172] Mostow, J., Voigt, K., Explicit integration of goals in heuristic algorithm design, IJCAI87 (January 1987).
[173] Mumford, E., Wier, M., *Computer systems in work design the ETHICS method*, London, Associated Business Press, 1979.
[174] Mullery, G., CORE - a Method for controlled requirements expression, in *Proceedings of the Fourth International Conference on Software Engineering*, IEEE, CS Press, 1979, pp. 126-135.
[175] Mumford, E., Wier, M., *Computer systems in work design the ETHICS method*, London, Associated Business Press, 1979.
[176] Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M. (1990). Telos: a language for representing knowledge about information systems. ACM Trans. Information Systems 8, 4.
[177] Mylopoulos, J., Borgida, A., Yu, E., Representing Software Engineering Knowledge, *Automated Software Engineering*, Kluwer, 4, 1997, pp. 291-317.
[178] Mylopoulos, J., Chung, L., Nixon, B., Representing and Using Non-Functional Requirements: A Process-Oriented Approach, IEEE, Transactions on Software Engineering, 18 (6), June 1992, pp. 483-497.
[179] Mylopoulos, J., Chung, L., Yu, E., From Object-Oriented to Goal-Oriented Requirements Analysis, ACM, Communications of the ACM, 42(1), January, 1999, pp. 31-37.
[180] Naiman, C. F., and A. M. Ouksel. “A Classification of Semantic Conflicts in Heterogeneous Database Systems,” *Journal of Organizational Computing*, 5(2), 167-193, 1995
[181] Nissen, H., Jeusfeld, A., Jarke, M., Zemanek, G., Huber, H., Technology to Manage Multiple Requirements Perspectives, IEEE, Software, March 1996, pp. 37-48.
[182] Hans W. Nissen, Matthias Jarke, Repository support for multi-perspective requirements engineering. Lyytinen/ Welke (eds.): Special Issue on Meta Modeling and Method Engineering, Information Systems 24, 2 (1999).
[183] Neches, R., Swartout, W., Moore J.D., Enhanced maintenance and explanation of expert systems through explicit models of their development, IEEE, Transactions on Software Engineering, SE-11 (11), November 1985, pp. 1337-1351.
[184] Neumann, P.G., *Computer Related Risks*, Addison-Wesley, 1995.
[185] Norman, R.J., Nunamaker, J.F., Jr., CASE Productivity Perceptions of Software Engineering Professionals, *Communications of the ACM*, 32 (9), September 1989, pp. 1102-1108.
[186] B. Nuseibeh, To Be And Not To Be: On Managing Inconsistency in Software Development, Proceedings of 8th International Workshop on Software Specification and Design (IWSSD-8), pp164-169, Scloss Velen, Germany, 22-23 March 1996, IEEE CS Press
[187] Nuseibeh, B., Kramer, J., Finkelstein, A., A Framework for Expressing the Relationship between Multiple Views in Requirements Specification, IEEE, Transactions on Software Engineering, October, 1994, 760-773.
[188] B. Nuseibeh, Ariane 5: Who Dunnit?, IEEE Software, May 1997.
[189] A. Hunter and B. Nuseibeh, Managing Inconsistent Specifications: Reasoning, Analysis and Action, (to appear in) ACM Transactions on Software Engineering and Methodology, 1998
[190] B. Nuseibeh and A. Finkelstein, ViewPoints: A Vehicle for Method and Tool Integration, Proceedings of the 5th International Workshop on Computer-Aided Software Engineering (CASE '92), 50-60, Montreal, Canada, 6-10th July 1992, IEEE CS Press.

[191] Oliver, J.R., On Artificial Agents for Negotiation in Electronic Commerce, IEEE, *Proceedings of the 29th Annual Hawaii International Conference on Systems Sciences*, 1996, 337-346.
[192] Olsen, G.R., Mark Cutkosky, Jay M. Tenenbaum, & Thomas R. Gruber. Collaborative Engineering based on Knowledge Sharing Agreements Procedings of the 1994 ASME Database Symposium, September 11-14, 1994, Minneapolis, MN.
[193] Osborne, M., Rubinstein. A., A Course in Game Theory, The MIT Press, Cambridge, MA, 1994.
[194] Osterweil, L., Sutton, S., Using Software Technology to Define Workflow Processes, *NSF Workshop on Workflow & Process Automation*, Athens, GA 1996.
[195] Osterweil, L.J., Software Processes are Software Too, *Proceedings of the Ninth International Conference of Software Engineering*, Monterey CA, March 1987, pp. 2-13.
[196] Charles Perrow, Normal Accidents: Living with HIgh-Risk Technology, Basic Books, Inc., New York, 1984.
[197] Klaus Pohl, Requirements Engineering: An Overview, *Encyclopedia of Computer Science and Technology*, A. Kent, J. Williams (editors), Volume 36, Supplement 21, Marcel Dekker, Inc., New York, 1997 (Also available as CREWS-96-02 from RWTH Aachen, Informatik V - Information Systems.)
[198] Potts C., Bruns G. "Recording the Reasons for Design decisions", *Proceedings of the 20th International Confernece on Software Engineering*, 1988 pp 418-427
[199] C. Potts, “Using Schematic Scenarios to Understand User Needs”, *Proc. DIS 95 - ACM Symposium on Designing interactive Systems: Processes, Practices and Techniques*, University of Michigan, August 1995.
[200] Potts, C., Using Schematic Scenarios to Understand User needs, ACM, *Proceedings of the Symposium on Designing Interactive Systems: Processes, Practices, and Techniques* (DIS’95), University of Michigan, August, 1995.
[201] Potts, C., Takahashi, K., Anton, A., Inquiry-Based Require ments, Analysis, IEEE, *Software*, pp. 21-32.
[202] Potts, C., Requiements Completeness, Enterprise Goals and Scenarios, Georgia Institute of Technology, College of Computing, August, 1994.
[203] Pruitt, D., *Negotiation Behavior*, Academic Press Inc.(1981).
[204] Raiffa, H., *The art and science of negotiation*, Harvard University Press(1982).
[205] Raiffa, H., *Decision Analysis*, Addison-Wesley, Reading, Mass., 1968
[206] Ram, S., Ramesh, V., A Blackboard-Based Cooperative System for Schema Integration, IEEE Expert/Intelligent Systems & Their Applications Vol. 10, No. 3, June 1995
[207] Ramesh, B., Dhar, V., Supporting systems development by capturing deliberations during requirements engineerng, IEEE, Transactions on Software Engineering, 1992, pp. 498-510.
[208] Ramesh, B., Jarke, M., Towards Reference Models for Requirements Traceability, IEEE, Transactions on Softwre Engineering, to appear.
[209] Brian.Randell, "London Ambulance Service Inquiry Report (long)," Forum On Risks to the Public in Computers and Related Systems, http://catless.ncl.ac.uk/risks, Vol. 14, Issue 48, March 24, 1993.
[210] Rangaswamy, A., Eliashberg, J., Burke, R.R., Wind, J., Developing Marketing Expert Systems: An Application to International Negotiations, Journal of Marketing, 53, October, 1989, 24-39.
[211] Rasmusen, E., A Model of Negotiation, Not Bargaining, Indiana University, Indiana University Working Paper in Economics No. 94-007, May 9, 1995.
[212] Robbins, S., *Organizational behavior: concepts, controversies, and applications*, Prentice Hall, NJ(1983).
[213] Robinson, W., Towards formalization of specification design, Masters thesis, University of Oregon (June 1987).
[214] Robinson, W.N., Automating the Parallel Elaboration Of Specifications: Preliminary Findings, CIS-TR-89-02, University of Oregon, February 1989.
[215] Robinson, W., Automating Negotiated Design Integration: Formal Representations and Algorithms for Collaborative Design, Doctoral Dissertation, University of Oregon, March 1993. (Also available as CIS-TR-93-10 from the University of Oregon.)
[216] Robinson, W.N., Automated Assistance for Conflict Resolution in Multiple Perspective Systems Analysis and Operation, ACM *Workshop on Viewpoints in Software Development*, In Association with the ACM Symposium on Foundations of Software Engineering, San Francisco, USA, October 14th & 15th, 1996.
[217] Robinson, W.N., A Decision Theoretic Perspective of Multiagent Requirements Negotiation, In *Automating software design: interactive design*, Workshop Notes from the Ninth National Conference on Artificial Intelligence, AAAI, July 15, 1991, 154-161. (Also available as RS-91-287 from ISI/USC.)
[218] W.N. Robinson, Electronic Brokering for Assisted Contracting of Software Applets, IEEE, *Proceedings of the 30th Annual Hawaii International Conference on Systems Sciences*, January 7-10 1997, To appear.
[219] Robinson, W.N., Goal-Oriented Workflow Analysis and Infrastructure, National Science Foundation, *Workshop on Workflow & Process Automation Workshop,* Athens, GA, 1996. (Also, available as Georgia State University, Working Paper CIS-96-07, May 1996.)

[220] Robinson, W.N., Integrating Multiple Specifications Using Domain Goals, *5th International Workshop on Software Specification and Design*, (1989) 219-226
[221] Robinson, W.N., Interactive Decision Support for Requirements Negotiation, *Concurrent Engineering: Research & Applications*, Special Issue on Conflict Management in Concurrent Engineering, The Institute of Concurrent Engineering, 1994 (2) 237-252.
[222] Robinson, W.N., Negotiation Behavior During Requirement Specification, *Proceedings of the 12th International Conference on Software Engineering*, IEEE Computer Society Press, Nice, France (March 26-30 1990) 268-276
[223] Robinson, W.N., Preference and Function Modeling in Requirements Mediation, In *Model-based reasoning*, Workshop Notes from the Ninth National Conference on Artificial Intelligence, AAAI, July 14, 1991.
[224] Robinson, W.N., Fickas, S. Supporting Multi-Perspective Requirements Engineering, *First International Conference on Requirements Engineering,* IEEE, (April 18-22 1994) 206-215.
[225] W.N. Robinson, S. Pawlowski, Surfacing Root Requirements Interactions from Inquiry Cycle Requirements, *International Conference on Requirements Engineering,* IEEE, (April 1998) To appear.
[226] W.N. Robinson, S. Pawlowski, Managing Requirements Inconsistency with Development Goal Monitors, IEEE, *Transaction on Software Engineering*, in submission.
[227] Robinson, W.N., Fickas, S., *Negotiation freedoms for requirements engineering*, CIS-TR-90-04, University of Oregon, April 6, 1990.
[228] Robinson, W.N., Volkov, S., *Conflict-Oriented Requirements Restructuring*, GSU CIS Working Paper 96-15, Georgia State University, Atlanta, GA, September, 1996. (In submission to IEEE Transactions on Software Engineering.)
[229] Robinson, W.N., Volkov, S., A Meta-Model for Restructuring Stakeholder Requirements, *Proceedings of the 19th International Conference on Software Engineering*, IEEE Computer Society Press, Boston, USA (May 17-24 1997), pp 140-149.
[230] Robinson, W.N., Volkov, S., Supporting the Negotiation Life-Cycle, ACM, *Communications of the ACM*, to appear.
[231] Robey, D., Farrow, D.L., Franz, C.R., Group Process and Conflict in Systems Development, The Institute of Management Sciences, *Management Science*, 35(10), October, 1989, pp. 1172-1191.
[232] Rosenschein, J., Genesereth, M., Deals among rational agents, *Proceedings of the 1985 IJCAI*, (August 1985) 91-99.
[233] Rosenschein, J., Ginsberg, M., Genesereth, M., Cooperation without communication, *Proceedings of AAAI-86*, Morgan Kaufmann Publishers, Inc. (1986) 51-57.
[234] Rosenschein, J. Breese, J., Communication-free interactions among rational agents: a probabilistic approach," in: Eds. L. Gasser, M.N. Huhns, *Distributed artificial intelligence*, Morgan Kaufmann Publishers Inc. (1989)
[235] Rosenschein, J., Zlotkin, G., Rules of Encounter, The MIT Press, 1994.
[236] Ross, D.T.,Structured Analysis (SA): A language for communicating ideas, IEEE, *Transactions on Software Engineering,* 3 (1) January 1977, 16-34.
[237] Rumbaugh, J., Designing Bugs and Dueling Methodologies. *Journal of Object-Oriented Programming*, Jan 1992.
[238] Sandholm, T. and Lesser, V., Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework. *First International Conference on Multiagent Systems* (ICMAS-95), San Francisco, 1995, 328-335.
[239] Sathi, A., Fox, M., Constraint-directed negotiation of resource reallocations, Eds. L. Gasser, M.N. Huhns, *Distributed Artificial Intelligence*, Morgan Kaufmann Publishers Inc., San Mateo (1989) 163-193.
[240] Sathi, A., Morton, T., Roth, S., Callisto: an intelligent project management system, *AI Magazine*, (Winter 1986) 34-52.
[241] Schuler, D. Namioka, A., Participatory design, Lawrence Erlbaum Assoc., Hillsdale, New Jersey, 1993.
[242] Sen, S., Durfee, E., A formal study of distributed meeting scheduling: preliminary results, Department of Electircal Engineering and Computer Science, University of Michigan, 1991
[243] Shakun, M.F., Airline Buyout: Evolutionary Systems Design and Problem Restructuring in Group Decision and Negotiation, The Institute of Management Sciences, *Management Science*, 37(10), (October 1991), 1291-1303.
[244] Schelling, T.C., *The Strategy of Conflict*, Cambridge, Massachusetts: Havard Univ. Press, 1960.
[245] Shepard, R., On subjectively optimum selections among multi-attribute alternatives," Eds. W. Edwards, A. Tversky, *Decision making*, (1967) 257-283.
[246] Shaw, M., Gaines, B., Comparing Conceptual Structures: Consensus, Conflict, Correspondence and Contrast, Knowledge Acquisition 1(4), 341-363, 1989

[247] Shaw, M., Gaines, B., A methdology for recognizing consensus, correspondence, conflict and contrast in a knowledge acquisition system, *Workshop on knowledge acquisition for knowledge based systems*, Banff (November 7-11, 1988)
[248] Sheth A., (Ed.), *Workshop on Workflow & Process Automation*, National Science Founation, Athens, GA 1996.
[249] Smith, D.R., Kotik, B., Westfold, S.J., Research on Knowledge-Based Software Environments at Kestrel Institute, IEEE, Transactions on Software Engineering, 11 (11), November, 1985, 1278-1295.
[250] Smith, R. G., "The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver", IEEE, Transactions on Computers, C-29, Vol. 12, December 1980, 1104-1113.
[251] F. Schneider, S. M. Easterbrook, J. R. Callahan and G. J. Holzmann, "Validating Requirements for Fault Tolerant Systems using Model Checking," Third IEEE Conference on Requirements Engineering, Colorado Springs, CO, April 6 - 10, 1998.
[252] Spacappietra, S., and C. Parent. "View Integration: A Step Forward in Solving Structural Conflicts," IEEE *Transactions on Knowledge and Data Engineering*, 6(2), 258-274, April 1994
[253] Spanoudakis G., Constantopoulos P., Analogical Reuse of Requirements Specifications: A Computational Model, Applied Artificial Intelligence: An International Journal, Vol. 10, No. 4, pp.281-306,1996
[254] Spanoudakis, G. & Finkelstein, A. "Reconciling Requirements: a method for managing interference, inconsistency and conflict"; Annals of Software Engineering, (1997)
[255] Storey, V.C., Goldstein, R.C., Chiang, R.H.L., Dey, D., and Sundaresan, S., "Database Design with Common Sense Business Reasoning and Learning." *ACM Transactions on Database Systems*, forthcomming.
[256] Stringer-Calvert, D., Rushby, J., A Less Elementary Tutorial for the PVS Specification and Verification System, CSL Technical Report CSL-95-10, Stanford University, 1995.
[257] A.G. Sutcliffe, N.A.M. Maiden, S. Minocha, D. Manuel, Supporting Scenario-based Requirements Engineering, IEEE, Transaction of Software Engineering: Special Issue on Scenario Management, 1998.
[258] Swartout, W., Balzer, R., On the inevitable intertwining of specification and implementation, *CACM* Vol. 25 (1982) 438-440.
[259] Sycara, K., Decker, K., Pannu, A., Williamson, M., and Zeng, D., Distributed Intelligent Agents, IEEE Expert/ Intelligent Systems & Their Applications Vol. 11, No. 6, December 1996, pp. 36-46.
[260] Sycara, K., Problem Restructuring in Negotiation, The Institute of Management Sciences, *Management Science*, 37(10), (October 1991), 1248-1267.
[261] Sycara, K., Resolving goal conflicts via negotiation, *Proceedings of the AAAI-88*, (1988) 245-250.
[262] A. van Lamsweerde, A. Dardenne, B. Delcourt and F. Dubisy, The KAOS Project: Knowledge Acquisition in Automated Specification of Software. Proceedings AAAI Spring Symposium Series, Stanford University, American Association for Artificial Intelligence, March 1991, pp. 59-62.
[263] A. van Lamsweerde, E. Letier, Integrating Obstacles in Goal-Driven Requirements Engineering, *Proceedings ICSE'98 - 20th International Conference on Software Engineering*, IEEE-ACM (Kyoto, April 98)
[264] van Lamsweerde, A., Darimont, R., Massonet, P., Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt, IEEE, Second International Symposium on Requirements Engineering, March 27-29, 1995, pp. 194-203.
[265] van Lamsweerde, Darimont, R., Massonet, P., The Meeting Scheduler System—Preliminary Definition, Internal Report, University of Louvain, 1993.
[266] van Lamsweerde, Darimont, R., Letier, E., Managing Conflicts in Goal-Driven Requirements Engineering, IEEE, Transactions on Software Engineering, 24 (11) November 1998, pp. 908-926.
[267] A. van Lamsweerde, L. Willemet, Inferring Declarative Requirements Specifications from Operational Scenarios, IEEE *Transactions on Software Engineering*, Special Issue on Scenario Management, IEEE (December 1998)
[268] Velthuijsen, H., Distributed artificial intelligence for runtime feature-interaction resolution, IEEE, *Computer*, Vol. 26, No. 8, August 1993, 48-55.
[269] Vessey, I., Sravanapudi, A.P., CASE tools as collaborative support technologies, *Communications of the ACM,* 38(1) Jan 1995, pp. 83-95.
[270] von Martial, F., Coordinating plans of autonomous agents, Springer-Verlag, 1992
[271] Waldinger, W., Achieving Several Goals Simultaneously, in Machine Intelligence, Vol. 8, E. Elcock, D. Michie (Eds.), Ellis Horwood, 1977.
[272] Walz, D.B., Elam, J.J., and Curtis, Bill, Inside a software design team: Knowledge acquisition, sharing, and integration, Communications of the ACM, Vol. 36, No. 10 (October 1993), 63-77.
[273] Walz, D.B, Elam, J.J., Krasner, H., A methodology for studying software design teams: An investigation of conflict behaviors in the requirements definition phase, from Empirical Studies of Programmers: Second Workshop (edited by Gary M. Olson, Sylvia Sheppard, and Elliot Soloway), Ablex Publishing Corporation, Norwood, NJ, 83-99. (Conference held in Washington, D.C. on 12/7-8, 1987)

[274] Weinberg, B.M., Schulman, E.L., Goals and Performance in Computer Programming, Human Factors, 16 (1), 1974, pp. 70-77.
[275] Werkman, K., Knowledge-based model of using shareable perspectives, *Proceedings tenth international conference on distributed artificial intelligence*, (October 1990) 1-23.
[276] Wilensky R., Planning and understanding, Addison-Wesley, 1983.
[277] Wing, J. A Study of 12 Specifications of the Library Problem. IEEE *Software*, July 1988, pp. 66-76.
[278] Winograd, T., Flores, F., *Understanding Computers and Cognition*, Addison-Wesley, 1987.
[279] Yakemovic, K., Conklin, J., Experience with the gIBIS model in a corporate setting, *Proceedings of CSCW 90*, Los Angeles, 1990.
[280] Yang, W., Horwitz, S., and Reps, T., A program integration algorithm that accommodates semantics-preserving transformations. ACM Transactions on Software Engineering and Methodology 1, 3 (July 1992), 310-354.
[281] Yen, J., Lee, H.G., Bui, T., Intelligent Clearinghouse: Electronic Marketplace with Computer-Mediated Negotiation Supports, IEEE, *Proceedings of the 29th Annual Hawaii International Conference on Systems Sciences*, 1996, 219-227.
[282] J. Yen and W. Tiao, A Systematic Tradeoff Analysis for Conflicting Imprecise Requirements, in *Proceedings of the Third IEEE International Symposium on Requirements Engineering* (RE'97), January 5-8, 1997.
[283] Yu, E.S.K., Mylopoulos, J., An Actor Dependency Model of Organizational Work - With Application to Business Process Reengineering, Proceedings of Conference on Organizational Computing Systems (COOCS'93), Milpitas, CA, November 1-4, 1993.
[284] Zave, P., Jackson, M., Four Dark Corners of Requirements Engineering, *Transactions on Software Engineering and Methodology*, ACM, 6 (1) January 1997, pp. 1-30.
[285] Zave, P., *FAQ Sheet on Feature Interaction*, http://akpublic.research.att.com/info/pamela/faq.html AT&T, 1999.
[286] Zeleny, M., *Multiple criteria decision making*, McGraw-Hill(1982).