Surfacing Root Requirements Interactions from Inquiry Cycle Requirements Documents

William N. Robinson & Suzanne Pawlowski

DEPARTMENT OF COMPUTER INFORMATION SYSTEMS
GEORGIA STATE UNIVERSITY

October 1, 1997
GSU CIS Working Paper
CIS-97-5
Surfacing Root Requirements Interactions from Inquiry Cycle Requirements Documents

William N. Robinson                        Suzanne Pawlowski
Department of Computer Information Systems
Georgia State University
Atlanta, GA 30301-4015 USA
+1 404 651 3867
wrobinson@gsu.edu           http://cis.gsu.edu/~wrobinso

Abstract

Systems requirements errors are numerous, persistent, and expensive. To detect such errors during the development of a requirements document, we have defined Root Requirements Analysis. This technique is simple in that it is based on: generalizing requirements to form root requirements, exhaustively comparing the root requirements, and applying simple metrics to the resultant comparison matrix. Root Requirements Analysis is also effective. In the case study described in this article, the technique finds that 36 percent of the case's root requirements interactions result in problems which require further analysis. Moreover, the technique provides a specific, operational procedure to guide the efficient iterative resolution of identified requirements conflicts. The process of Root Requirements Analysis itself is not specific to a particular methodology. It can be applied directly to requirements in a variety of forms, as well as to the documentation of requirements development. We took this later approach in the case study illustrating how Root Requirements Analysis can augment the Inquiry Cycle model of requirements development. Finally, the technique is amenable to support through collaborative CASE tools, as we demonstrate with our DEALSCRIBE prototype.

Index Terms—Requirements engineering, goal/assumption surfacing, stakeholder analysis, conflict management, meta-modeling, requirements Inquiry Cycle Model, scenario analysis
1 Introduction

Requirements analysts need tools to assist them in reasoning about requirements. To some degree, Computer Aided Software Engineering tools have been successful in providing support for modeling and code generation[7][24][32]; however, they have been less successful in supporting requirements analysis[24]. 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[16]. Thus, analysts will benefit from techniques and tools which directly address requirements analysis.

In this article, we describe a requirements analysis technique which can be used to identify certain types of problems that occur in requirements documents, as well as provide a strategy for their removal. The technique, called Root Requirements Analysis, can be used to uncover requirements interactions which can cause problems during the development, operation, or maintenance of systems. Once these requirements conflicts are surfaced, several analyses, based on simple conflict counts and percentages, can be used to guide their efficient correction. A case study has validated the utility of the technique. Additionally, a multi-user World Wide Web tool has been developed to support Root Requirements Analysis.

1.1 A Need to Surface Requirement Interactions

The importance of requirements engineering can be illustrated with some of the more extreme disasters attributed to failures in requirements engineering: (1) complete loss of the Ariane 5 launcher[25], (2) the crash of Lufthansa A320 in Warsaw[21][22], and (2) 62 and 79 percent of safety-related functional faults in the Voyager and Galileo space probes, respectively[26]. Requirements errors are: numerous, ranging from 25 percent to 70 percent of total software errors—in US companies, averaging one per function point[19]; 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[5].

No doubt, there are a variety or reasons that requirements errors go undetected. However, as Peter G. Neumann notes in his book on Computer Related Risks, one reason may be the complex interactions among 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." — Peter G. Neumann[31].

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 have other interfering actions. We refer to such negative interactions between requirements, as a requirements conflict.

1.2 Related Research Addressing Requirements Conflict Surfacing

Three general approaches to surfacing requirements conflicts are:

- **Ontological** In the ontological approach, conflict surfacing is assisted by providing a set of meaningful terms, or ontology, by which one can specify conflict relationships between requirements. Such works include requirements interaction types[8], fuzzy interactions[48], and general requirements relationships[36]. These more basic works can be incorporated into methodological or technological frameworks.

- **Methodological** In the methodological approach, the application of a system development method surfaces conflicts. Many of those which explicitly surface requirements conflicts do so by integrating multiple views of system requirements—for example, CORE[28], ETHICS[29], Soft Systems Method[6], ViewPoints[33], and CORA[37]. As part of the integration step, these methods compare representations, detect conflict, generate resolutions to create a consistent, integrated description. Some system development methods surface conflicts by incorporating a specific technique, such as the Inquiry Cycle’s use of scenario analysis[35].

- **Technological** Some technological approaches support a specific methodology, while others are methodology neutral; however, these approaches have a common goal of providing a specific technique, or automation, which can be used to surface requirements conflicts. Examples include: conflict detection through a collaborative messaging environment[4][17][23], structure-based conflict detection[43], scenario-based conflict surfacing[2][27][35], and conflict classification[11][20].

The technique we describe in this article, called Root Requirements Analysis, is similar to other technological approaches in that it can be automated. One distinction is that it has an defined procedure by which to select a subset of requirements for analysis. This is important, as analysis of all interactions among all requirements can involve significant computation; for \( n \) requirements, there are \( n(n-1)/2 \) binary relationships. WinWin’s approach is similar in that
it has an analysis focusing procedure[4]. By attributing requirements with non-functional quality goals, such as performance, users of the WinWin system can be notified when two or more requirements affect performance. Users can then analyze the attributed requirements to determine if they do indeed conflict. By only comparing requirements with similar quality attributes, analysis is reduced to a subset of the many possible comparisons. Scenario analysis is also similar in that it narrows conflict analysis to requirements which participate in an undesirable scenario. In the case of Root Requirements Analysis, requirements are generalized to form root requirements; it is these root requirements which are then analyzed. Moreover, once the root conflict relationships are determined by an analyst, analysis of the root requirements conflicts can be used to efficiently guide an iterative conflict resolution procedure in order to derive a consistent requirements document[37][40][42].

In this article, we next introduce requirements of a distributed meeting scheduler and use it to illustrate problems of interacting requirements. The subsequent sections present the Root Requirements Analysis technique (§ 3), a case study of applying the technique to the distributed meeting scheduler requirements (§ 4), a tool prototype for supporting the technique (§ 5), and finally, conclusions (§ 6).

2 Requirements for a Distributed Meeting Scheduler

Throughout this article, we draw on examples from analyses of requirements for a distributed meeting scheduler. We choose the meeting scheduler because of: (1) the complex requirements interactions which, depending on how they are addressed, lead to considerable variation in the resulting implementations; (2) the availability of a widely circulated, yet rich, requirements document[45]; and (3) the publication of a prior analysis of the case[35][44]—including our own[37][38][41]. Hence, this case allows us, and others, to compare analyses[13].

The general problem of the meeting scheduler can be summarized by the introduction to the requirements[45]

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 description refine the roles of the meeting scheduler and participants. However, this introduction will be sufficient to understand the examples that follow.

2.1 An Illustrative Problem: Interacting Requirements

Any substantial requirements document will have requirements which depend on other requirements. For example, consider two requirements of the distributed meeting scheduler:

\[ R_8 \] :: "The scheduler shall minimize participant scheduling effort."

\[ R_{13b} \] :: "Participant’s preference information shall be complete and accurate."

If the scheduler continually probes participants for information in order to achieve requirement \( R_{13b} \), then participant effort will be high. Thus, requirement \( R_8 \) depends on requirement \( R_{13b} \). Moreover, they have a potentially negative or conflicting interaction. As part of the requirements development process, a new requirement may be specified such that it clarifies the dependency between \( R_8 \) and \( R_{13b} \) and removes any potential conflicts. For example, consider the new requirement:

\[ R_{new} \] :: "The scheduler shall infer participant meeting preferences"

If we assume that the scheduler’s ability to infer participant schedules is (nearly) as good as a participant’s ability to state their preferences, then this resolves the potential conflict between requirements \( R_8 \) and \( R_{13b} \). Additionally, this new requirement resolves a conflicts between \( R_8 \) and another requirement, \( R_{13d} \):

\[ R_{13d} \] :: "Participant’s shall be given reminders to respond to requests for information."

Additionally, it adds further support to the achievement of two other requirements:

\[ R_{2b} \] :: "The meeting is held on the date preferred by the participants."

\[ R_{2d} \] :: "The meeting is held at a location preferred by the important participants."

However, the new requirement introduces a new conflict, since it conflicts with \( R_{13c} \):

\[ R_{13c} \] :: "The scheduler shall provide participants with the ability to state their preferences."

Thus, (1) requirements depend on each other, and (2) requirements resolutions alter these dependencies. (The latter actually is implied by the former, since requirement resolutions are themselves new requirements.) In general, conflict resolution seeks to reduce the overall number of conflicts as each new resolution is added to the requirements document (i.e., seek monotonically decreasing contention). Additionally, the process seeks to reduce the number of prior resolutions that must be reconsidered as new resolutions are introduced (i.e., seek minimize resolution backtracking). However, achieving these process goals in practice is difficult. Root Requirements Analysis is one technique aimed at...
solving the efficient identification and resolution of requirements conflicts.

3 Root Requirements Analysis

Two objectives of Root Requirements Analysis are: (1) understanding the relationships among the requirements, and (2) ordering requirements by their degree of conflicting relationships. Using this knowledge, one can iteratively and efficiently resolve conflicts in a large requirements document.

The overall procedure of Root Requirements Analysis is:

(1) **Identify** or **surface**, key root requirements that “cover” all others in the requirement document.
   Root requirements (indirectly) lead to all concepts of a requirements document.

(2) **Structure the root requirements**

(3) **Identify key relationships among root requirements**
   For example, rank requirements by their degree of conflict, called *contention*.

(4) **Iteratively resolve the most conflicting requirements**

In the specific application of Root Requirements Analysis described herein, we manually: (1) apply generalization to derive root requirements, (2) apply pairwise root requirement comparison to derive root requirements relationships, and (3) use the requirements relationships to iteratively select the most contentious requirements for resolution. This technique is important in that it provides a systematic method by which key requirements conflicts can be surfaced and then systematically selected for efficient resolution. The following section summarizes each step.

3.1 Identifying Root Requirements

The objective of root requirement identification is to determine key requirements whose interaction analysis leads to the discovery of significant requirements relationships. While one could exhaustively compare every requirement with every other, in practice, such analysis is not feasible for non-trivial requirements documents. Instead, we seek to identify root requirements which represent key concepts from which other requirements are derived through elaboration. While the binary comparison of such root requirements will not uncover every requirement relationship, it will narrow analysis to key requirements to which further analysis can be applied.

The means of identifying root requirements varies with the structure of the initial requirements representation. The overall procedure is as follows: (1) group requirements into sets by the concepts they reference, (2) order requirements by generality, (3) generate or select the most general requirements for each concept, and (4) repeat steps 1-3 until concept generalizations are not meaningful. The resulting requirements are the root requirements. While it is desirable that the root requirements be a minimal set which cover all other requirements through some set of development relationships, such as elaboration, it is not necessary. In our application, root requirement identification was an informal process aimed at identifying key requirements from which key analyses can be derived.

In applying the above identification procedure, some root requirements are directly identifiable in the requirements. For example, Rₖₗₐ₃: "Meetings shall be held as soon as possible". In other cases, some inferencing or generalization was needed to define a root requirement that captured the key concept of a set of requirements. For example, Rₖₗₐ₄: "Scheduling of meetings shall be completed as soon as possible", was inferred from requirements related to meeting scheduling. In each case, the root requirements were subjectively deemed to cover a key concept which concerned a number of requirements: meeting time, and meeting scheduling, respectively.

As defined above, root requirements are generalizations of a requirements document. Even within this requirements subset, requirements can be arranged in a concept generalization hierarchy. For example, often a single abstract requirement may have a number of related *sub-requirements*, which define specialized properties of the concept. For example, consider the following requirements:

Rₕₗₐ₃: "A scheduling decision shall be based on complete and accurate information."
Rₕₗₐ₄: "The scheduler shall obtain and maintain accurate and complete room availability information."
Rₕₗₐ₅: "The scheduler shall obtain and maintain accurate and complete participant preference information."
Rₕₗₐ₆: "The scheduler shall provide participants with the ability to state their preferences."

Requirement Rₕₗₐ₃ covers the concept of the other requirements (Rₕₗₐ₄, Rₕₗₐ₅, Rₕₗₐ₆), as they all describe required properties of information used in scheduling decisions. Thus, requirement Rₕₗₐ₃ is a more central member of the root requirements, as it subsumes the concept of the other requirements. We define the minimal set of root requirements, each of which is the most general of its concept, as the *dominant root requirements*. Again, determining the dominant root

---

1. Note that if requirement generalization is not a selective process, then a single requirement (e.g., Thing) would result. Thus, we apply generalization only when we subjectively deem it conceptually meaningful.
requirements is subjective, as we must determine when requirement concepts of are distinct—as a manual operation. This has not been a problem. However, the dominant root requirements can be so abstract that it is difficult to determine interesting relationships among them. Thus, we subjectively select some specialized sub-requirements of the dominant root requirements, to join to the dominant root requirements to form our root requirements working set.

The above discussion can be summarized as: (1) generalize all requirements, (2) form a set of the most general requirements where each references an independent concept, and (3) consider adding some specializations of each selected requirement.

3.2 Structuring Requirements

The objective of requirement structuring is to determine for each requirement: the stakeholder(s), the basic requirement description, the mode of achievement, the actor for achievement, and any associated qualitative goals. We have found these requirements attributes to be useful in reasoning about requirements interactions. Hence, in preparation for interaction analysis, an analyst translates the textual requirements into requirements with the following structure:

```plaintext
[RequirementName] \text{Requirement}^{Perspective:: ModeResponsibleAgent, \textbf{[(ModeQualitativeGoals)]}}
```

As an example requirement, consider the requirement:

The Scheduler should obtain participant preference information.

Rewritten as a structured requirement, we have:

```plaintext
ParticipantPreferences \text{R}^{All}_{13b}:: \text{Scheduler}, \textbf{Achieve}, \text{ "Description of participant calendar preference information" (Max Accuracy)}
```

It states that the requirement named ParticipantPreferences (requirement number 13b) is defined in all perspectives, and that the Scheduler agent is responsible for achieving the context where an unspecified agent describes participant calendar preferences. The ParticipantPreferences requirement does not indicate how descriptions are achieved; however, the following requirement fills in this detail. It specifies that participants describe their own preferences and that such descriptions should be maximally accurate; moreover, the Scheduler should maximize the ease of use for participants.

```plaintext
ParticipantDescribePreferences \text{R}^{All}_{13c}:: \text{Scheduler}, \textbf{(Max EaseOfUse)}
```

In general, requirements may be named. Each is numbered and associated with a perspective. The other elements are as follows:

- **Description** The basic state or event description of the requirement.
- **Description qualitative goals** Goals concerning the non-functional, or qualitative, nature of the description: for example, maximize preference accuracy in the ParticipantPreferences requirement concerns the accuracy of the described preferences.
- **Actor agent** The actor agent indicates which agent is responsible for achieving the description.
- **Mode responsible agent** The mode responsible agent indicates which agent is responsible for achieving the mode of the requirement.
- **Mode** The mode indicates the intentional nature of the description: either Allow, Achieve, Maintain, Avoid, or Cease.
- **Mode qualitative goals** Goals concerning the non-functional, or qualitative, nature of the achievement mode are specified; for example, maximize ease of use in the ParticipantDescribePreferences requirement concerns how the Scheduler allows participants to describe their preferences.

The above requirements structure focuses discussion on key elements of stakeholder requirements; these include: which perspective owns them, who is responsible for achieving them, and what are the non-functional aspects of the requirement and its mode of achievement. Identifying the responsibility of requirements has been shown to be a key in identifying the system boundary[49], as well as key to composing systems[12][14]. We have included Allow and Cease in the growing ontology of achievement modes[1][37][41][44]. Additionally, we distinguish between the actor achievement and mode achievement. This allows for expressions such as, "The system should allow a participant to describe their preferences"; it indicates that participants achieve descriptions, but the system allows (or provides for) participant achievement. We have found that such direct distinction between actor and modal responsibility facilitates the identification of system boundary and other interactions among requirements responsibility. Similarly, we have provided for the specification of qualitative goals, while keeping the distinction between actor and modal qualitative goals.

3.3 Analyzing Root Requirements

The objective of analyzing root requirements is to determine an ordering of requirements by their degree of expected conflict. The first step is to construct a requirements comparison matrix. This matrix is then used to create require-
ments rankings based on: the number and percentage of (conflicting) relationships, and the subjective probability of conflict occurrence. Each of these analyses is presented next.

An exhaustive root requirements comparison matrix is constructed by labeling both the row and column headings of a matrix with the root requirements. Next, for each unique requirements pair the potential conflict relationship is subjectively assigned; this creates an upper triangular matrix. For the purposes of this analysis, the qualitative descriptors of: Very Conflicting (-), Conflicting (-), Neutral (N) Supporting (+), and Very Supporting (++) were used. For example, the relationship between ScheduleUsingGoodInfo (RJ3 above) and ScheduleMeetingSoon (RJ3 above) was assigned Very Conflicting based on the time necessary to collect the required information.

Table 1 illustrates the types of relationships that were identified in the Root Requirements Analysis of the case describe in section 4. For example, RJ3 and RJ3 are assigned Very Supporting because RJ3, is a subgoal of RJ3. Such relationship rationale is also indicated in the table. While such requirement interrelationships can be defined formally[1][8] and even automatically derived from formal requirements[10][38], we simply relied on an subjective determination during phase.

<table>
<thead>
<tr>
<th>Example Requirements</th>
<th>Relationship</th>
<th>Relationship Rationale</th>
<th>Potential Conflict Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>RJ3 : &quot;A scheduling decision shall be based on complete and accurate information.&quot;</td>
<td>Very Supporting</td>
<td>RJ3 is a subgoal of RJ3</td>
<td>N/A</td>
</tr>
<tr>
<td>RJ3c : &quot;Participants shall be provided the ability to state their preferences.&quot;</td>
<td>Supporting</td>
<td>RJ3c enables RJ3</td>
<td>N/A</td>
</tr>
<tr>
<td>RJ3 and RJ5b : &quot;Scheduling shall maximize efficiency.&quot;</td>
<td>Neutral</td>
<td>Not directly related.</td>
<td>N/A</td>
</tr>
<tr>
<td>RJb : &quot;A scheduled meeting shall consist of at least two participants.&quot;</td>
<td>Conflicting</td>
<td>RJ5b somewhat thwarts RJ3</td>
<td>0.5</td>
</tr>
<tr>
<td>RJ3 and RJ5a : &quot;While scheduling, the time-period for resolving conflicts shall be minimized&quot;</td>
<td>Very Conflicting</td>
<td>RJ5a thwarts RJ3</td>
<td>1</td>
</tr>
<tr>
<td>RJ6 : &quot;The scheduler shall minimize participant scheduling effort.&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Illustration of Root Requirements Matrix Relationships

Table 1 also illustrates the subjective assessment of the probability that the requirements conflict will occur in the running system. For example, based on the given requirements it was estimated that about one-half of all scheduling attempts would either: (a) schedule a meeting based on incomplete or inaccurate information (RJ3), or (b) extend the time-period for resolving scheduling conflicts in order to gain more information (RJ5); thus, RJ3 and RJ5 are indicated with a 0.5 conflict potential in table 1. While the relationships of table 1 are both subjective and approximate, they have provided a good initial characterization requirements relationships. In general, such subjective relationships are commonplace among informal requirement techniques[23][34], as well as some formal techniques[8].

Once the comparison matrix is established, it can be used to create requirements rankings. First, for each requirement, the number of conflicting relationships in which it participates can be determined. Second, contention can be determined. We define the degree requirements contention to be the percentage of all relationships the requirement participates in which are conflicting; thus, if a requirement's contention is 1, then it conflicts with every other requirement in the requirements document. Third, for each requirement, the average potential conflict can be determined by averaging across its conflicting relationships. From each of these simple computations, the requirements can be rank ordered for resolution.

4 A Case Study

To assess the utility of Root Requirements Analysis, we applied the method to an established requirements engineering problem case, that of the distributed meeting scheduler introduced in section 2. The objective of the case-study was to assess two hypotheses: (1) could Root Requirements Analysis be easily incorporated into an existing methodology?, and (2) could Root Requirements Analysis add value by uncovering requirement relationships? Fortunately, we obtained access to analysis documents generated during an the Potts et. al. application of the Inquiry Cycle to the
distributed meeting scheduler problem[35]. By applying Root Requirements Analysis to the Inquiry Cycle discussion documents, we were able to assess both hypotheses.

4.1 The Requirements Inquiry Cycle

The Inquiry Cycle model uses a rich inquiry-conversation metaphor by which discussion about documented requirements lead to the evolution of the requirements document. Discussion is prompted by a question. Answers, representing positions of stakeholders, suggest alternative ways to resolve the question; answers may have rationale that justify the alternative. For example, the question “What are the preconditions for holding a meeting?” generated two answers; one of those answers was selected and the rationale for that selection was captured; a new requirement was added to the requirements document. Questions may be generated through a variety of techniques, the most common of which is scenario analysis. Scenarios describe a system’s behavior in specific situations, providing a description of “one or more end-to-end transactions involving the required system and its environment”[35].

Figure 1 illustrates the result of applying the Inquiry Cycle model to the distributed meeting scheduler. The case produced 33 questions of the original requirements; 40 answers to those questions; 38 changes to the requirements; and 18 reasons for the changes that were made.

4.2 Incorporating Root Requirements Analysis in the Inquiry Cycle

Given the Inquiry Cycle analysis, we considered two ways to apply Root Requirements Analysis. First, the original requirements could be analyzed; such a case-study would result in a direct comparison between the Inquiry Cycle and Root Requirements Analysis. Second, the requirements discussion of the Inquiry Cycle could be analyzed. We choose this second approach, as it provided an illustration of how Root Requirements Analysis could augment another method.

Root requirements were generated from the Inquiry Cycle requirements discussion (questions, answers and reasons) in the following way:

1. The requirements discussion was segmented by concept. For example, a discussion concept of user calendar privacy for meeting availability included one question, one answer, and two proposed requirements changes.

2. Root requirements were generated for each discussion concept. Within some concept discussions, a root requirement was explicitly stated (8). In other cases, it could be directly inferred (14) or inferred through some reasoning (8); see figure 3.

Figure 2 illustrates how the 30 root requirements were generated, 13 of which were dominant root requirements.

4.3 Root Requirements Analysis of the Inquiry Cycle Discussion

Once the root requirements were determined, a requirements comparison matrix was created. Figure 2 summarizes the type and number of requirements relationships which were uncovered as part of the matrix comparison. Given the large number of conflicting and very conflicting relationships, three further analyses were applied. First, a graph of the

---

Figure 1. Results of applying Root Requirements Analysis to the Inquiry Cycle discussion. In the Inquiry Cycle, ovals indicate the number of unique instances of a type, while arcs indicate the flow of analysis within the Inquiry Cycle. The Root Requirements and graph of relationship counts by type was created from Root Requirements Analysis.

GSU Working paper CIS 97-5
type and number of conflicting relationships was created, as illustrated in 2. Second, a requirements contention graph was created, as illustrated in figure 3. Third, an average potential conflict graph was created, as illustrated in figure 4.

Root analysis does provide insight into relationships among requirements. For example, notice that figure 2 shows that the two most conflicting (rightmost) dominant requirements are $R_8$ and $R_{13}$. Requirements $R_1$, $R_2$, $R_3$, $R_8$ and $R_{13}$ are summary requirements; for example, $R_{13}$ includes the relationship counts of $R_{13}$ and its sub-requirements of $R_{13a}$ through $R_{13g}$. The contention graph of figure 3 shows that $R_8$ and $R_3$ are among the most contentious of all root requirements. In this way, these graphs can guide analysis toward the more conflicting requirements.

Further analysis of the actual comparison matrix shows that some of the more contentious requirements interact with each other (e.g., $R_8$ and $R_{13}$), while others do not (e.g., $R_8$ and $R_3$). Moreover, figure 4 shows that $R_8$ and $R_{13}$ have a high average potential conflict; in fact, they have the highest average potential conflict of those that directly interact. Thus, based on conflict count, contention, and potential conflict, $R_8$ and $R_{13}$ are the two best candidates for resolution.

Figure 2. Graph of the numbers of root requirements interactions. The number that each requirement participates with all other requirements for five relationship types (Very Conflicting, Conflicting, Neutral, Supporting, and Very Supporting) are presented in an additive "stacked" graph; ordered by increasingly negative interactions.

Figure 3. Graph of root requirements interactions. The percentage that each requirement participates with all other requirements for five relationship types (Very Conflicting, Conflicting, Neutral, Supporting, and Very Supporting) are presented in an additive "stacked" graph; ordered by increasingly negative interactions.
4.4 Observations

Root Requirements Analysis was easily incorporated into the Inquiry Cycle. First, the requirements discussion was organized and then segmented by concept. Once the discussion elements were organized by concept, identifying or generating root requirements was simple.

Root Requirements Analysis did add value by providing insight into requirements relationships. The systematic comparison ensures that all key binary requirements interactions will be considered. This led to the discovery of 72 conflicting relationships. In comparison, the original Inquiry Cycle discussion explicitly referred to only 9 “conflicts”.

In defense of the Inquiry Cycle, the method is not explicitly aimed at uncovering requirements interactions. Instead, its informal conversational approach is intended to promote an open discussion by stakeholders. Consequently, the discussion moves through topics according to the interests of the stakeholders. Often, implicit assumptions resolve potential conflicts without further discussion. In fact, in the Inquiry Cycle, a conflict is simply the explicit posting of two alternative answers to a question. Thus, while the stakeholders might have understood other conflicts, only 9 were explicitly recorded.

Root Requirements Analysis also provided insight into the most conflicting and interacting requirements. Prior to Root Requirements Analysis, there was no indication that R₈ and R₁₃ were the most contentious directly interacting requirements. The comparison matrix, in combination with the graphs, showed them to be the most directly conflicting.

In the case of the meeting scheduler, resolving the most contentious directly interacting requirements did support the goal of monotonically decreasing contention. We verified this by determining the consequences of resolving high, moderate, and low contention requirements. As illustrated in section 2.1, resolving the highly contentious directly interacting requirements R₈ and R₁₃ not only eliminate their very conflicting interaction, but it also directly eliminated another conflict as well as add support to two other requirements. On the other hand, resolving moderate contention directly interacting requirements did not remove other conflicts, but simply lessened the potential occurrence of other conflicts. Resolving a low contention requirement actually introduced four more conflicts than it resolved. Such empirical evidence supports the common sense notion that highly contentious requirements are the root causes of conflict within a requirements document, and their resolution leverages other improvements in the requirements document.

In general, we found the matrix and its graphs provided a good overview of the requirements interactions. This was useful when determining which requirements conflict to focus on, and which requirements should be considered as part of resolution generation.

5 Supporting Root Requirements Analysis

Little automated support was provided for the above analysis of the distributed meeting scheduler. The requirements and their initial interaction coding were maintained in a word processor. The matrix analysis and graphs were maintained in a spreadsheet program. Such tools are adequate for a small group over a short time-frame. However, these tools are not adequate for more substantial projects.

To address the basic problems of collaborative CASE, information control, sharing, and monitoring should be supported[46]. A requirements analysis tool should minimally support requirements traceability (i.e., through various

![Figure 4. Graph of root requirements average potential conflict. The average estimated probability that each conflict will occur with all other requirements; probability are scaled [0..1]; ordered by decreasing likelihood of occurrence.](image-url)
states: informal, formal, tentative, conflicting, stable, implemented, etc.), concurrent group access, and the various requirements analyses. We have developed a tool with such support to aid Root Requirements Analysis, as well as other techniques which are part of our Conflict Oriented Requirements Analysis (CORA) framework[37].

We have developed a tool, called DEALSCRIBE, which supports conversations about requirements. The initial purpose of building DEALSCRIBE was to determine if Root Requirements Analysis could be made more efficient through tool support. While we did not conduct the case study using DEALSCRIBE, we have experimented with it to confirm its utility. (A case-study using DEALSCRIBE will be conducted next.)

DEALSCRIBE was created by building upon two existing tools: HyperNews and ConceptBase.

- **HyperNews** HyperNews provides a discussion system similar to Usenet News, but it has a World Wide Web interface. In each forum a user can post typed text messages. A message may be posted to the forum, or in response to a particular message. A WWW view of the forum can provide an overview of the discussion, where messages are laid out in an tree format that shows replies to a message indented under it. HyperNews provides: various views of a forum, user notification of new responses, an email interface, security, and administrative functions.

- **ConceptBase** ConceptBase is a deductive database which provides a concurrent multi-user access to O-Telos objects[18]. All classes, meta classes, instances, attributes, rules, constraints, and queries are uniformly represented as objects. ConceptBase itself operates as a server, while clients, such as ConceptBase’s graphical browser communicate via internet protocols. This system has shown to be a powerful tool for systems development, partly because of its ability to simultaneously represent and query instances, classes, and meta-classes[15][30].

In building DEALSCRIBE, we used ConceptBase to provide definitions of message structures and queries, as well as storage of the message instances. We also modified HyperNews to use the ConceptBase message definitions to present a form-based WWW interface. Now, DEALSCRIBE messages can be simple text (as in HyperNews), WWW input forms, or even the result of an analysis function (e.g., a ConceptBase query). To tailor DEALSCRIBE to a particular development model, a designer simply defines a ConceptBase message hierarchy; additionally, a designer may define (active) analysis messages which run ConceptBase, or other programs, to produce results which are posted to DEALSCRIBE as a message.

To illustrate how DEALSCRIBE can support Root Requirements Analysis, consider the following two ConceptBase classes:

```plaintext
Class StructuredRequirement isA Requirement with attribute
  perspective : Perspective;
  modeAgent : Agent;
  modeGoals : QualitativeGoal;
  mode : Mode;
  actorAgent : Agent;
  description : String;
  descriptionGoals : QualitativeGoal;
end

Class RequirementInteraction isA Interaction with attribute
  r1 Requirement;
  r2 Requirement;
  causality : Causality;
  correlation : Correlation;
  probOccurrence : Probability;
  riskImpact : FuzzyNumber;
  contextDependence : FuzzyNumber;
  dependencyCauses : Class
end
```

The class, StructuredRequirement, defines the elements of structured requirement (§ 3.2), but also the elements of the input form which can be filled in by a user of DEALSCRIBE. Similarly, users can post RequirementInteraction messages indicating how requirements interact. Once such messages are posted, the following parameterized ConceptBase query will collect requirements which have a specific interaction type (e.g., InputInteraction = VeryConflicting) for a specific requirement (e.g., InputRequirement):

```plaintext
GenericQueryClass Interactions isA RequirementInteraction with parameter
  InputRequirement : Requirement
  InputInteraction : RequirementInteraction
constraint
  Req_with_interaction : $ ((this r1 InputRequirement) or (this r2 InputRequirement)) and (this correlation InputCorrelation) $
end
```

The results of Interactions are then passed to a Perl function which computes the number of conflicting relationships.

---

2. DEALSCRIBE is a member in the DEALMAKER suite of tools aimed at assisting collaboration through negotiation in requirements analysis[41] and electronic commerce[39].
Conclusions

Figure 5. An DEALSCRIBE screen showing part of the forum discussing the meeting scheduler requirements. The requirement and analysis message types support Root Requirements Analysis.

degree requirements contention, and average potential conflict for the input requirement. The results of such analyses are themselves posted to the discussion (and the ConceptBase database). Thus, DEALSCRIBE can track not only the state of the typed requirements, but also the state of various analyses. A DEALSCRIBE screen of our experimental application of DEALSCRIBE for Root Requirements Analysis is shown in figure 5.

6 Conclusions

Root Requirements Analysis focuses on understanding key interaction among requirements and using that knowledge to guide requirements modification. It does not explicitly address general analyses of requirements, such as completeness or correctness of a requirements document. Instead, the techniques focus on pairwise conflicts among root requirements. Hence, the technique is intended to supplement, rather than replace other requirements analysis techniques.

In the meeting scheduler case, the full analysis of the Inquiry Cycle requirements discussion effort amounted to roughly one person-week. However, the Inquiry Cycle requirements discussion documents were only a partial analysis of the meeting scheduler case. Based on the Potts et. al estimate of a full Inquiry Cycle requirements discussion of 4.7 person-months[35], and our own analysis of the meeting scheduler, we believe an independent Root Requirements Analysis may add a total of 0.5 person-months. Our rough estimate of 1.2 hour per Inquiry Cycle root requirement does not take into account any efficiencies that may be gained through incremental or overlapping analyses between the Inquiry Cycle and Root Requirements Analysis. For example, Root Requirements Analysis may help to focus the Inquiry Cycle discussion and thereby reduce the overall effort. Given the Potts et. al. estimate of between 500 to 1000 hours for a complete requirements development using the Inquiry Cycle, an addition of 80 hours, some of which may
Conclusions

focus the discussion, seems worthwhile. In fact, due to the simplicity and efficiency of the method, it would appear that other semi-structured or informal methods, such as JAD, would benefit from the conflict surfacing and resolution guidance of Root Requirements Analysis. Moreover, based on our limited experiences with DEALSCRIBE, we believe that tool support will substantially decrease this level of effort.

In the meeting scheduler case, we found that 36 percent of all possible root requirements interactions had some conflicting aspect. Finding such problems early on is a fundamental goal of requirements engineering. While such analysis does add some effort to development projects, some projects, such as safety critical projects, will find it worth the effort.

Acknowledgment

We gratefully acknowledge the help and cooperation of Drs. Colin Potts and Annie Antón for providing documentation of their Inquiry Cycle analysis of the meeting scheduler. We also thank Georgia State University and the College of Business for funding portions of this research.

References


GSU Working paper CIS 97-5


