7. W. Kunz and H. Rittel, *Issues as Elements of Information Systems*, Working Paper 131, Inst. Urban and Regional Devt., Univ. Calif. at Berkeley, 1970.
8. N. Goldman, "Three Dimensions of Design Development" *Proc. AAAI*, 1983
9. C. Potts, "Software Engineering Research Revisited," IEEE Software, September, 1993, in press.
10. K. Mani Chandy and J. Misra *Parallel Program Design: A Foundation*, Addison-Wesley, 1988.
11. R. Schank and R. Abelson, *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*, Lawrence Erlbaum Associates, 1977.
12. A. Dardenne, S. Fickas and A. Van Lamsweerde, "Goal-directed Requirements Acquisition," *Science of Computer Programming*, to appear.
13. S. Fickas and R. Helm, "Knowledge Representation and Reasoning in the Design of Composite Systems," *IEEE Trans. Software Eng., 18(6)*: 470-482, 1992.
14. J. Lee, "Extending the Potts and Bruns Model for Recording Design Rationale," *Proc. 13th Int. Conf. Software Eng.*, IEEE Comp. Soc. Press, 1991
15. B. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.
16. J. Wood and D. Silver, *Joint Application Design*, Wiley, 1989.
17. C. Potts, ".Supporting Software Design: Integrating Design Methods, and Design Rationale" in T. Moran and J. Carroll (eds.) *Design Rationale: Concepts, Techniques, and Use*, Lawrence Erlbaum Associates, 1994, in press.

ments analysis. We chose a commercial document processor for our case study over a prototype tool that we had developed ourselves [1]. That tool, which was based on Emacs, was not very usable. To follow an inquiry-based approach, we believe that the user needs to have access to an effortless *annotation environment*. We are currently reimplementing and enhancing the functionality of our Inquiry Cycle support tool in HyperCard and MetaCard. This version, *Suiko*, will provide a more transparent annotation environment. We are comparing retrieval or browsing strategies with a 'bird's-eye' (network) view versus a 'turtle' (node-based) view [17] of the requirements documentation and discussions. Suiko also supports agenda management and progress tracking.

(b) Scenario types. We are investigating the representation and value of different types of scenarios. We are also investigating goal-based heuristics that suggest what scenarios to analyze and which of those to elaborate further.

(c) Transition to design. Following work in the OOA/OOD communities [2, 6], we are looking at the transition from scenario-based requirements analysis to object identification and responsibility-driven design.

References

1. C. Potts and K. Takahashi, ''An Active Hypertext Model for System Requirements,"*Proc. 7th Int. Workshop Software Specification and Design*, IEEE Comp. Soc. Press, 1993, to appear.
2. I. Jacobson, *Object-Oriented Software Engineering: A Use-case Driven Approach*, Addison-Wesley, 1992.
3. J. Karat and J.L. Bennett, ''Using Scenarios in Design Meetings—A Case Study Example," in J.Karat (ed.) *Taking Software Design Seriously: Practical Techniques for Human-Computer Interaction Design*, Academic Press, 1991.
4. K. Benner, M. Feather, W.L. Johnson and L. Zorman, ''Utilizing Scenarios in the Software Development Process," *Proc. IFIP WG8.1 Working Conf. Inf. Sys Development Process*, North-Holland, 1993.
5. M. Lubars, C. Potts and C. Richter, ''Developing Initial OOA Models,"*Proc. 15th Int. Conf. Software Eng.*, IEEE Comp. Soc. Press, 1993.
6. K. Rubin and A. Goldberg, ''Object Behavior Analysis," *Comm. ACM, 35(9)*: 48-62, September, 1992.

The reasons for elaborating a requirement are often useful to know later when implementing or maintaining the system. Because discussions are attached to requirements or scenario components and may be consolidated into rationale objects [1], we assume that the rationale for requirements should be fairly easy to retrieve. However, our emphasis has been on keeping track of ephemeral reasoning to improve the requirements analysis process, not to justify reasoning for subsequent implementors. We have not evaluated the subsequent use of recorded requirements discussions.

Effort

Our estimate of effort spent in the case study is in line with commercial practice. A full elaboration of the all 16 scenarios for the meeting scheduler, several iterations of the inquiry cycle, and a formal revision process would take on the order of 500 to 1,000 person-hours.

If the resulting product were developed following what Boehm calls a 'semi-detached' development mode and delivered about 32,000 lines of code, his Basic COCOMO model estimates a project effort of 146 person-months [15]. Since a semi-detached project typically spends seven percent of its total development effort in requirements and planning, about half of which is actually spent doing requirements analysis, the requirements effort would require 4.7 person-months, or about 750 person-hours.

Another sanity-check is to compare our projected effort with that required to perform a JAD session [16]. If the session lasts for one week and involves ten stakeholders full-time, with two person-weeks of preparation and two of follow-up, then the total effort would be 800 person-hours.

Obviously, these estimates are only very rough. However, the fact that the numbers are in rough agreement shows that following an Inquiry Cycle approach does not require inordinate effort.

Future Work

Our ongoing work emphasizes three directions:

(a) Tool support. Although our emphasis is on process, not tools, our experience shows that tool support is an important factor in the successful application of inquiry-based require-

We have only begun to investigate different forms of scenario analysis and their effectiveness in clarifying and improving requirements. We are especially interested in comparing the value of general, thematic descriptions of scenarios with detailed, instantiated scenarios. Intuition suggests that increased detail is more appropriate once one knows more about the system. Paradoxically, however, our preliminary work suggests that fully instantiated scenarios are equally useful early in the requirements analysis process. Perhaps the effort required to construct them forces stakeholders to surface and discuss assumptions that would otherwise be hidden for longer. A team of stakeholders may even go as far as role-playing a concrete scenario (as we did in the case of `Annie Out Of Town`).

The Inquiry Cycle and its Shortcuts

Our practical experience convinces us that the Inquiry Cycle vocabulary is expressively adequate for the types of discussions held during requirements analysis. Intelligent tool support would require a more explicit and formal model of the domain (e.g. meeting scheduling) and a richer ontology of speech acts and transformations (e.g. [14]). However, increased formality would defeat the object of using the Inquiry Cycle as a foundation for directing and systematizing exploratory thought during the requirements phase.

Since our goal is to devise methodological guidelines and principles that help in the analysis of requirements for real systems, we conclude that the Inquiry Cycle exhibits an appropriate level of structure. We have striven to achieve a balance between regimenting an inherently informal and situated activity, and providing no guidance at all.

The Inquiry Cycle facilitates the recording of discussion information in two ways:

(a) It is artifact-based. Unlike most idea-generation and issue-based methods, the Inquiry Cycle is directed at *reviewing existing artifacts*. Therefore, discussion always centers around the stated requirements and missing requirements that emerge from analyzing scenarios

(b) Short-cuts are always possible. For example, a stakeholder may record an assumption about implementation constraints by annotating the appropriate requirement without having to raise a spurious question first.

pant has not responded by the drop-dead date, the scheduler prepares a list of possible meeting times based on the replies it has received."

A non-functional requirement change request, on the other hand, is a change to the requirements *document*. Here, the change is expressed as the addition of a clarification or definition. By adding the definition, the system is further constrained and potential ambiguities are removed. However, the original intent is not changed. An example is: *Add clarification about conflict resolution mechanisms—"All resolution strategies can be applied to weak conflicts. All but asking the participants to revise their preferences apply to strong conflicts."* (In a weak conflict participants' preferences conflict; in a strong conflict their availabilities do.)

An editorial change request is a request to re-word or rewrite some requirement. It is not intended to remove ambiguity, but to correct grammatical or spelling errors or to introduce constistent terminology. An example: *Change "potential attendee" to "potential participant."*

Following the analysis, 31 changes were made to the requirements documentation (28 to the requirements, three to scenarios). About two-thirds of these changes (65 percent) were part of a full inquiry cycle consisting of questions, answers, reasons and a change. Of the short cuts to the inquiry, about half (54percent) of the discussions omitted questions, the changes in these cases resulting from assumptions that were attached directly to the requirements document or scenario. The rest (56 percent) omitted answers, too. The changes in these cases arose directly from an analysis of the requirements. Despite the number of changes, the number of requirements increased by only one, from 38 to 39.

Discussion and Conclusions

Scenarios

There is little doubt that scenarios can be useful for elaborating requirements. Our evidence shows that some questions about requirements are not easily answered except by resorting to scenario analysis and that at least half the improvements to a set of requirements can come from analyzing scenarios, not the requirements documents themselves.

question. Consider the question: *Is an important participant's attendance vital—i.e. can a meeting go ahead without an important participant?* This suggests immediately the follow-on question: *Is an active participant's attendance vital?* These can both be generalized into a second follow-on question: *What are the preconditions for holding a meeting?*

The Nature of Assumptions

Stakeholders often make assumptions—that is, they answer a tacit question about the requirements without articulating the question. For example, an assumption about the meeting scheduler' s communication medium could be construed as an answer to the question: *What is the nature of the communication medium?*

Stakeholders seldom justify their assumptions or consider alternatives (we did not). Assumptions, like answers to explicit questions, may be retracted. Given the insidious nature of unrecognized but false assumptions and their tentative nature, the stakeholder should flag all assumptions carefully and make an extra effort to justify and authorize assumptions, consider alternatives, and reconsider them occasionally.

Assumptions seem commonest when discussing implementation constraints or the system' s environment, or when discussion cannot otherwise continue. Some assumptions belong in both categories. For example, the meeting scheduler requirements imply, but do not state, that the system is mediated by electronic mail. We went further and made the scheduler agent itself an e-mail participant. An equally valid interpretation of the requirements, however, would be to have the scheduler be invoked by a single user (the initiator) who would be responsible for sending messages and interpreting replies.

Requirements Evolution

We find qualitatively different goals for three types of change requests: mutation requests, restriction requests, and editorial requests.

A *mutation request* calls for a change or addition to the requirements themselves. Thus, when such a change is enacted, the system being described changes, for example: *Add new requirement ''The initiator specifies a drop-dead date when calling the meeting. If a partici-*

(e) When questions ask about the timing constraints on some event or events. For example: *If a potential meeting attendee does not respond, at what time should the scheduler go ahead and schedule the meeting?* In the meeting scheduler, the meeting cannot be scheduled until at least some of the potential attendees have made known their schedules, but the scheduling cannot be delayed until after the meeting should be held. Answering this *when* question led us to the concept of the *drop-dead date,* the time when the scheduler makes its best schedule on the information available.

(f) Relationship questions ask how one requirement is related to another. The ***drop-dead date*** just mentioned has implications for the interpretation of other requirements, for example: *Are there constraints on the drop-dead date, given the current date and the initiator's proposed date range for the meeting?*

Although most of the questions raised were about the term or idea that was the object of analysis at the time, we also encountered many instances where analyzing one requirement or scenario suddenly prompted a question about something else. It is not practical to untangle the train of thought in most such cases, and we do not try. The *results* of the train of thought, however, are very useful and should not be lost. We call these serendipitous discussion elements *parenthetical insights*.

Questions that led to parenthetical insights were raised in the following two ways, both of which occurred equally often when analyzing the requirements directly and when analyzing scenarios:

(a) What-if questions. A particularly fruitful heuristic is to ask about cases where an action could go wrong or its preconditions be unsatisfied. Pursuing this type of question often leads to insights about apparently unrelated features of the system, for example: *What would happen if the late-comer responds with preferences after the meeting has already been scheduled, and the preferences conflict with the schedule?*

(b) Follow-on questions stem from other pending questions. An important category of follow-on question is where one question generalizes another. The answer to the new, broader question, may lead to changes in the requirements in places other than the one that led to the

Recording questions definitely helps keep track of open issues. All of the questions were answered, one of them directly by a scenario, the others by answers. Most answers (94 percent) explicitly answered questions, rather than being assumptions or facts attached to requirements.

Most of the discussion consisted of raising possibilities that were not covered by the existing requirements, rather than deliberating among alternatives or arguing about the rationale for decisions. The average question had only 1.3 answers suggested. Only about one-third of the recorded answers represented alternatives that had been rejected. Reasons were given for about half the answers (43 percent of the selected answers, 50 percent of the rejected ones).

Kinds of Questions

Requirements discussions are triggered by several distinct types of question.

(a) What is questions request more information about a requirement. They challenge requirements, not scenarios, and are usually resolved by a definition. An example is: *What does 'keeping participants involved' mean?*(The requirement demands that the system keep meeting participants 'involved' in the scheduling process.)

(b) How-to questions ask how some action is performed. They arise from requirements and scenarios and may be resolved by analyzing a scenario. For example: *How does the organization specify how to resolve conflcts?*

(c) Who questions request confirmation about the agent responsible for performing some action or achieving some goal. They are resolved by analyzing scenarios— specifically, by entries in the agent columns of scripts. Scenarios especially help clarify policy issues and the division of responsibility between the system and users. One such question challenges the requirement that there are three types of participant (ordinary, active and important): *Who determines who is active and important?*

(d) What-kinds-of question request further refinements of some concepts. It is not clear from the meeting scheduler requirements, for example, what types of meeting should be supported.

analyzing the requirements document itself could not be answered except by constructing and analyzing scenarios.

The Inquiry Cycle

The number of nodes and links of the various types for the meeting scheduler are shown in Figure 4.. This shows the number of instances of each node and the number of transitions from node to node. The requirements document is less than two pages long. By assigning numbers to all paragraphs and sub-paragraphs, we can count 38 distinct requirements.

Requirements Discussion

Scenarios were at least as effective as the requirements document itself in prompting questions about the requirements. About half (55 percent) of the questions were raised while analyzing or constructing scenarios.

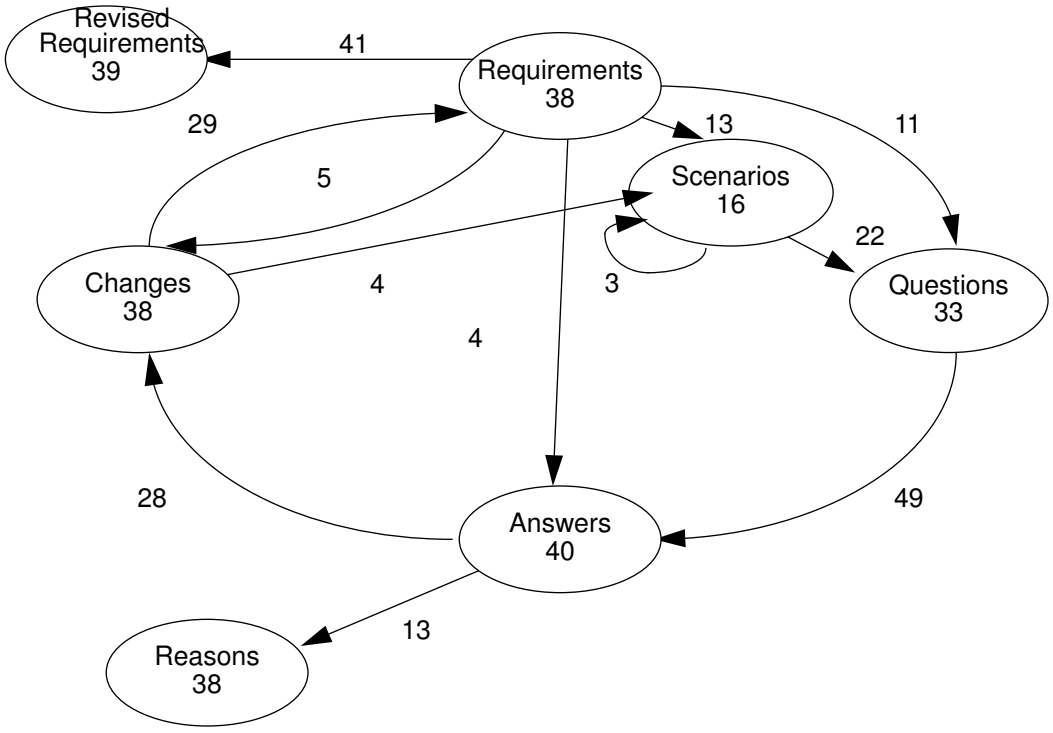

Figure 4: Number of different typers of nodes in the case study with frequencies of transitions between Inquiry Cycle states. In addition to Inquiry Cycle transitions, the diagram shows the frequencies of 'short-cuts'.

tendance of Annie (active), Kenji (important), and Colin (ordinary). The meeting must be held next week, but Kenji and Colin can make it only on days when Annie is out of town. This is a much more concrete scenario than the others.

In our experience, instantiating scripts doubles their size. Single actions in the generic scenarios translate into multiple action instances in the instantiated scenario. (There are 51 actions in the script for `Annie Out Of Town`, compared to the total of 21 actions in the relevant episode definitions.) Compare the initiation episode of Figure 3 with the detailed script in Table 4.

Whether more detail is effective in raising questions and accelerating convergence are questions that we are actively pursuing.

Scenario Use

Analyzing scenarios helps stakeholders raise questions about requirements and answer questions that have already been raised. Over half our changes to the meeting scheduler requirements (58 percent) stemmed from analyzing scenarios rather than reviewing the requirements document itself. About a quarter (28 percent) of the questions that arose while

Table 4: Detailed script for instantiated scenario `Annie Out Of town`

Agent	Action
Esther	Creates new meeting
Esther	Determines that Kenji is an important participant
Esther	Determines that Annie will be presenting
Esther	Determines that Colin is an ordinary participant
Esther	Types meeting description
Esther	Sets date range to be Mon-Fri next week (it's Wednesday p.m. now)
Esther	Determines drop-dead date is Friday noon
Scheduler	Sets timeout to be Fri 9am
Scheduler	Sends boilerplate message to Colin requesting constraints
Scheduler	Sends boilerplate message to Kenji requesting constraints and preferred location
Scheduler	Sends boilerplate message to Annie requesting constraints and equipment requirements

Scenario Detail

Scenarios may be more or less detailed. For example, the episode Scheduling, during which the scheduler determines the time and location for the meeting, must cover three cases. (See Table 3, in which the variants correspond to the episode of Figure 3.) Two are successful cases: the case where there is one feasible meeting time, and the case where there are multiple times. In the unsuccessful case there is no feasible schedule.

Related to detail is the genericity of a scenario. The scenarios presented so far are all generic, because their agents are agent *types*. When agents of the same type interact, or when several agents of one type interact with one of another, it may be better to instantiate them..

Returning to the date conflict scenario, consider a specific case, Annie Out Of Town, in which a particular meeting has been scheduled by an initiator, Esther, that requires the at-

Table 3: Action tables for Scheduling episode

(a) No conflict; several possibilities

Agent	Action
Scheduler	Find meeting times and locations that are in preference set and not in exclusion set
Scheduler	Notify initiator of available times
Initiator	Select time and location

(b) No conflict; one possibility

Agent	Action
Scheduler	Find meeting times and locations that are in preference set and not in exclusion set
Scheduler	Notify initiator of scheduled meeting time and location time

(c) Scheduling conflict

Agent	Action
Scheduler	Find meeting times and locations that are in preference set and not in exclusion set
Scheduler	Notify initiator of conflict

the form of episode names (e.g. `Initiation`, `Responding`). Episodes are themselves constructed out of actions (see Figure 3).

Finally, instead of representing scenarios as temporal sequences of actions or episodes, scenarios could be represented as goal-directed plan executions. Such an approach has been adopted in AI research on story understanding [11] and provides a potential bridge to requirements analysis processes based on goal refinement [12, 13].

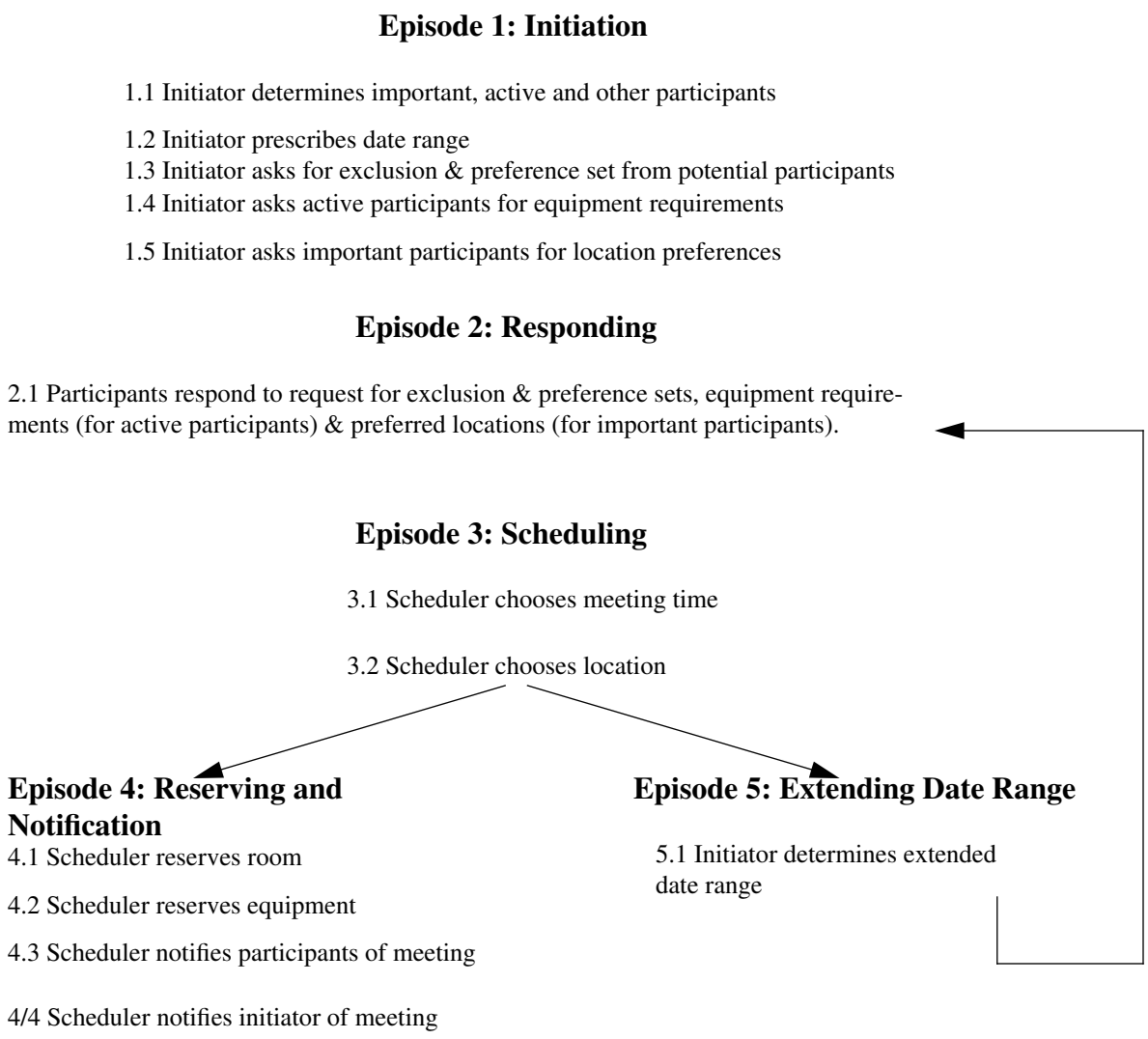

Figure 3: Episodic structure of use case 8 (Date conflict; initiator extends date range) and the action structure of each episode.

Scenarios may be represented as specific cases or as families of cases. When scenarios stand for families of cases, they must distinguish between common and different parts. In `Slow Responder` (Figure 2), there are three sub-cases depending on whether the tardy participant is ordinary, active or important. These cases share common fragments, yet have different outcomes. This sharing of common fragments demonstrates that scenarios may be composed, because the outcomes in cases *a* and *b* depend on the definition of other scenarios.

One may go further in recognizing the episodic structure of scenarios by making episodes the building blocks of scenarios. Episodes are 'phases' of activity, a role that is reflected in

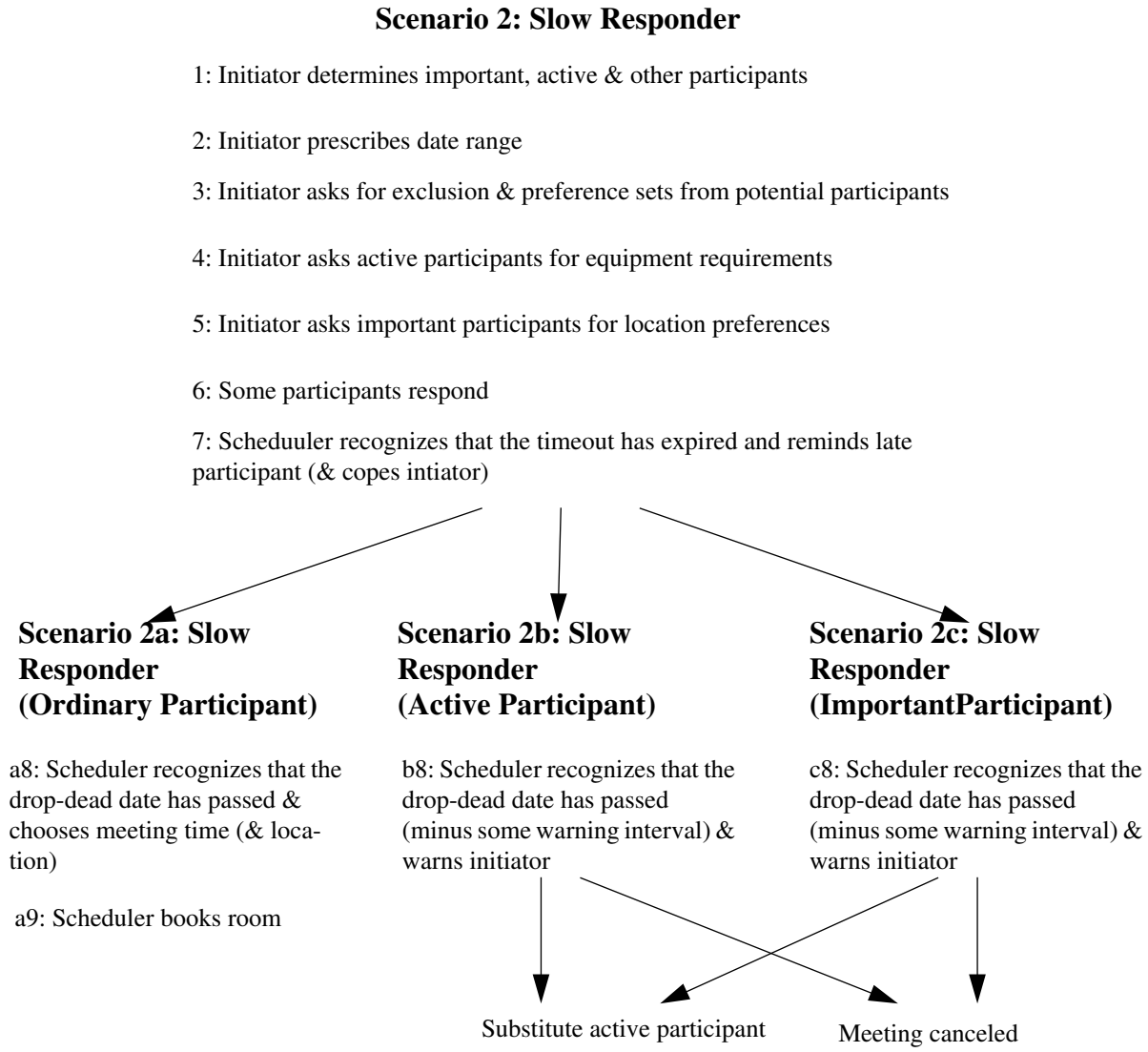

Figure 2: Actions for use case 2 (`Slow Responder`) variants (a) `Ordinary Participant`, (b) `Active Participant`, and (c) `Important Participant`.

This discovery strategy seems quite general: whenever a membership-changing action occurs, stakeholders should consider whether similar changes are covered by the requirements, and, if not, analyze scenarios that explore them.

Scenario Representation

Scenarios may be represented in several ways. Natural language summaries are at the extreme of informality. More formal are tabular representations (see Table 2, which shows the script for scenario 1). Tables, unlike natural language, encode temporal sequence directly.

Table 2: Script (Action Table) for `No Conflicts` scenario

No.	Agent	Action
1	Initiator	Request meeting of a specific type, with meeting info. (e.g. agenda/purpose) and date range
2	Scheduler	Add default (active/important) participants, etc.
3	Initiator	Determine 3 participants
4	Initiator	Identify 1 presenter as active participant
5	Initiator	Identify initiator's boss as important participant
6	Initiator	Send request for preferences
7	Scheduler	Send appropriate e-mail messages to participants (incl. additional requests to boss and presenter)
8	Ordinary participant	Respond with exclusion and preference set
9	Active participant	Respond with exclusion and preference sets and equipment requirements
10	Scheduler	Request required equipment
11	Important participant	Respond with exclusion and preference sets and possibly location preference
12	Scheduler	Schedule meeting based on responses, policies and room availability
13	Scheduler	Send confirmation message to all participants and meeting initiator

Table 1: Scenarios

No.	Use case
1	No conflicts
2	Slow responder
3	Late-coming participant
4	Dropout
5	Substitute active participant
6	Self-appointed active participant
7	Participant changes preferences before meeting is scheduled
8	Date conflict; initiator extends date range
9	Date conflict; participants exclude fewer dates
10	Date conflict; participants withdraw
11	Weak date conflict; participants extend preferred times
12	Room conflict
13	Scheduled meeting bumped by more important meeting
14	Participant tries to double-book
15	Conflict arises after meeting scheduled
16	Meeting canceled

coming participant, it is the initiator's inclusion of the person in the participant set. In both cases, the recovery action is to reschedule.

We found that looking for analogies of this type is an effective way to identify important scenarios. The driving question each time is: 'What can go wrong with this action?'

Another example of scenario discovery is provided by Self-appointed active participant. That scenario is not covered by the requirements, but a similar scenario—Substitute active participant—is. In Substitute active participant, a new active participant substitutes for someone else after scheduling has commenced. This scenario suggests Self-appointed active participant, in which a new active participant is added to the existing ones. The difference between the scenarios is that in the first the new active participant is *substituted*, whereas in the the second, he or she is *added*.

Scenarios

Several issues arise when introducing scenario analysis into the requirements analysis process.

(a) Where do the scenarios come from and how are they identified?

(b) What is the appropriate level of detail and instantiation for scenarios?

(c) How should scenarios be represented?

(d) What types of analysis discussions are facilitated by scenarios?

To address these issues, we first consider specific examples from the meeting scheduler and then return to the issues.

Use Cases and Scenarios in the Meeting Scheduler

We identified 16 scenarios for the meeting scheduler, which are given in Table 1In this paper, we consider three of these in more detail: use cases 1 (No Conflict), 2 (Slow Responder), and 8 (Date conflict; initiator extends date range)..

No Conflict is the simplest case, in which the participants' schedules do not conflict, and meetings do not contend for the same room. Slow Responder is the case in which a potential attendee does not respond in time for the meeting to be scheduled. Because the requirements do not stipulate how long the scheduler should wait, nor what actions it should take to remind the tardy participant or the initiator, we had to explore alternatives in the Date conflict; initiator extends date range scenario. In that scenario, the initiator resolves a scheduling conflict by suggesting other times.

Scenario Identification

The requirements were sufficiently detailed to give rise to most of the scenarios (14 of the 16). The remainder arose while considering other scenarios. For example, Late-coming participant is not covered by the requirements, but arose from analyzing Slow responder, in which a participant ignores the initiator's request for preferences. As in Slow responder, an action that affects an agent is delayed, requiring the system to recover. In the case of Slow responder, the delayed action is the participant's response, whereas in Late-

such a computational approach ignores many of the factors that make a workgroup support system succeed or fail.

The meeting scheduler helps people schedule rooms and equipment for meetings. Each meeting is called by an initiator and may have ordinary, active and important participants. The requirements do not clearly define these terms. We assume that presenters are active participants and may have special equipment requests. We further assume that an important participant's attendance is more important than that of an ordinary participant should a scheduling conflict arise. The initiator proposes some time constraints for the meeting and the potential attendees respond with their available and preferred times. Sometimes the scheduler can schedule the meeting; sometimes there are conflicts.

We assume that the reader has attended meetings, has a busy schedule, and has some experience using personal information managers. Armed with this near-universal knowledge and the above summary, it should be possible to follow the requirements analysis for the case study.

Analysis Process

We jointly performed a requirements analysis for the meeting scheduler. Using the Inquiry Cycle as a framework, we met regularly to review the requirements document and scenarios we constructed. To simplify the data gathering and data analysis, we maintained electronic copies of the requirements, a discussion document (consisting of questions, answers and reasons), a change request list, and a collection of use cases and scenario scripts. We produced these using FrameMaker, taking care to cross-reference related text objects together. Some of the documentation and modification was done during meetings. Mostly, however, we resorted to printed notes, updating the documents between meetings.

Meetings lasted 30-90 minutes and were held two or three times per week for ten weeks. Much of this time was spent in the elaboration of scenarios (part of a related investigation), and discussions about available and planned tool support. Thirty person-hours therefore seems a reasonable estimate for the analysis.

ideas and not just their validation [9], we have conducted a preliminary case study of a small-scale requirements document.

The case study is an exploratory analysis. We investigate the types of questions that are naturally asked about a set of written requirements, how those questions tend to be answered, and the role that scenarios play in this process. The Inquiry Cycle provides a framework for addressing these research questions.

Overview of Paper

In the next section we describe the example system and how we analyzed its requirements. Next, we discuss some issues that arise in scenario analysis, presenting illustrations from the case study. Then we go into further detail about the Inquiry Cycle as it applies to the example. Finally, we present our conclusions, discuss tool requirements, and present directions for future work.

Applying the Inquiry Cycle to a Case Study

The Meeting Scheduler Example

As a case study, we consider the problem of automatically scheduling meetings. Axel van Lamsweerde and his students have written a short requirements document for a meeting scheduler system[1]. We chose it for several reasons:

(a) The research community has adopted it as a benchmark.

(b) The requirements illustrate problems typical of requirements for real systems: they specify policies that may not work well in every organization, there is ample opportunity to dispute different interpretations, and many important issues are left unresolved.

(c) Specialized domain knowledge is not necessary to understand the case study.

(d) The example hides many interesting contextual factors. Although it is possible to treat meeting scheduling as a straightforward optimization problem (see [10] for such a treatment),

1. A. Van Lamsweerde, R. Darimont, Ph. Massonet, *The Meeting Scheduler Problem: Preliminary Definition*. Copies may be obtained from Prof. Van Lamsweerde, Universite Catholique de Louvain, Unite d'Informatique, Place Sainte-Barbe, 2, B-1348 Louvain-la-Neuve, Belgium. (avl@info.ucl.ac.be)

We call this shared information *requirements documentation*. We assume that the requirements documentation may include use cases and scripts for scenarios.

A *requirements discussion* consists of of three kinds of elements: *questions*, *answers*, and *reasons*. Most discussions start because a stakeholder has a question about a requirement. By answering questions and justifying answers, stakeholders develop a clearer understanding of the requirements, and notice ambiguities, missing requirements, and inconsistencies. An answer describes a solution to a question or set of questions. A reason is a justification given for an answer.

The ultimate effect of a requirements discussion is a commitment to freeze a requirement or change it. A *change request* may be traced back to a discussion, which constitutes its rationale, and traces forward to the changed requirement once it has been acted upon.

Our purpose in developing the Inquiry Cycle has not been to propose a rigid process model that stakeholders must follow. Short-cuts are always possible. For example, a requirement may be changed after little or no discussion. An 'answer' may be given to an unstated question, as often happens when stakeholders articulate assumptions that are not explicitly documented in the requirements. Choices may lack rationale, because stakeholders may judge the reasons for some answers to be obvious. A change request may be executed directly without a recorded discussion.

It is the integration of requirements documentation, discussion and evolution that distinguishes the Inquiry Cycle from speech-act models such as IBIS [7] or taxonomies of design transformations [8]. For a more detailed exposition and justification of the Inquiry Cycle model, see [1]. In this paper, we describe the model through the form of an example requirements analysis process.

Purpose of the Case Study

Although direct project experiences and empirical investigations led us to develop the Inquiry Cycle model and emphasize scenario elaboration, we have not yet validated our specific formulations of these ideas against large-scale system development projects. As a step in this direction and an attempt to incorporate practical experiences into the development of the

The Inquiry Cycle

The Inquiry Cycle is a cyclical model of the requirements elaboration process involving three components: *requirements documentation*, *requirements discussion*, and *requirements evolution*.

We assume the participants in the conversation are *stakeholders*—those with a stake in the system outcome. Stakeholders may include such diverse groups as end users, indirect users, other customer representatives, and developers. We do not enshrine these conventional roles or job titles in the model itself. Throughout this paper, we use the neutral term 'stakeholder' where 'analyst' might more normally be used.

Stakeholders share information about the system in the form of requirements documents (if any exist), background information about the problem domain and implementation constraints, and additional information produced during the requirements elaboration process.

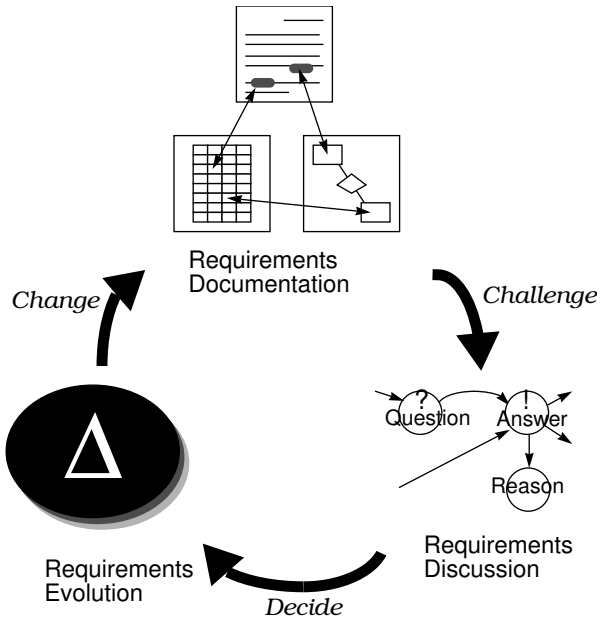

Figure 1: The Inquiry Cycle.Requirements documentation, consisting of requirements, scenarios and other information, is discussed through questioning, answering and justification of choices. Choices may lead to requested changes, which in turn modify the requirements documentation.

Introduction

The Inquiry Cycle is a framework for describing and supporting discussions about system requirements [1]. It divides requirements analysis into three intertwined processes: proposing or writing requirements, challenging or discussing them, and refining or improving them. Although the Inquiry Cycle does not require a specific method, it is especially compatible with the use of scenario analysis.

In this paper, we present an extended example of the Inquiry Cycle in operation, categorize the types of requirements discussion that occur in practice, and suggest some heuristics for analyzing requirements. We also explain how concrete scenarios improve analysis.

Scenarios

When people write requirements or discuss a requirements document they are imagining a system that does not yet exist. To ensure that they are describing the imagined system fully, it is always useful to ask ' what if' questions about specific cases or situations. These cases and the sequences of events that result are known as *scenarios*.

Using scenarios to test detailed requirements is accepted practice. Less well understood is the role that scenarios play in helping to understand informal requirements and to specify them in the first place. Recently, there has been much interest in the systematic use of scenarios in discovering and refining requirements [2, 3, 4, 5].

By 'scenarios' different authors mean different things. Some mean sequences of events [5, 6]; others mean more general ' use cases' [2]. Some use tabular or diagrammatic notations to describe scenarios [6, 5]; others present user interface storyboards [3] or prose descriptions [2]. We define a *scenario* as one or more (usually several) end-to-end transactions involving the required system and its environment. *Use cases* are short, informal descriptions of scenarios (perhaps, just a phrase to describe it). *Scripts* are more detailed definitions in a tabular or diagrammatic form. Our scripts are linear traces of event occurrences

In this study, we investigate the effectiveness of differing levels of detail and how scenarios of different types help in the discussion of specific requirements.

Inquiry-Based Scenario Analysis of System Requirements

Colin Potts[1]
Kenji Takahashi[2]
Annie Anton[1]

GIT-CC-94/14

January 1994

Abstract

The Inquiry Cycle is a formal structure for describing and supporting discussions about system requirements. It divides requirements analysis into three intertwined processes: proposing or writing requirements, challenging or discussing them, and refining or improving them. In this paper, we present an extended example (a meeting scheduler) of the Inquiry Cycle in operation, categorize the types of requirements discussion that occur in practice, and suggest some heuristics for analyzing requirements. We also explain how concrete scenarios improve analysis.

1. Full address: College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, USA. {potts, anton}@cc.gatech.edu
2. Full address: NTT Software Laboratories, NTT Shiragwa Twins Bldg., 1-9-1 Kohnan Minato-ku, Tokyo, Japan. kenji@nttspe.ntt.jp