ABSTRACT

ALSPAUGH, THOMAS ATKINS Scenario Networks and Formalization for Scenario Management. (Under the direction of Assistant Professor Annie I. Antón.)

Scenarios are widely used to specify the behavior of software due to their informality and accessibility. However, their informality makes them difficult to analyze and manage. We address these difficulties with two complementary approaches, one syntactic and one semantic, that add a small amount of structure to scenarios to allow automated analyses and support. The syntactic approach represents a scenario as a set of attribute-value pairs, some of which may also be viewed as events, each of which is an actor-action pair, that are arranged in a sequence. This representation supports the use of episodes (shared subsequences of events) to show dependency relationships between scenarios and to help maintain those relationships as the scenarios evolve. The representation also supports automated measures of similarity between scenarios, to find duplicates or near-duplicates, searching in a collection of scenarios, and assess requirements coverage and completeness of the collection. The representation can be analyzed for consistency of various attributes within individual scenarios. The semantic approach integrates the scenarios that describe a system into a network that expresses which scenarios can follow each other. The network expresses the context expected by the events of each scenario and the temporal relationships between the scenarios. This information is either implicit or incomplete for an ordinary collection of scenarios. Construction of a scenario network provides process guidance for assessing and improving completeness and consistency of the scenario collection. A scenario network represents equivalence relationships between scenarios, and these relationships can be used to organize and classify the scenarios and to maintain the temporal relationships between scenarios as the scenarios evolve. A scenario network can be analyzed to evaluate completeness of the scenario collection and several kinds of consistency between scenarios in the collection. Together the syntactic and semantic approaches form an effective approach for addressing the scenario management problem, which has not been effectively addresses heretofore.

Scenario Networks and Formalization for Scenario Management

by

Thomas A. Alspaugh

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Doctor of Philosophy

Department of Computer Science

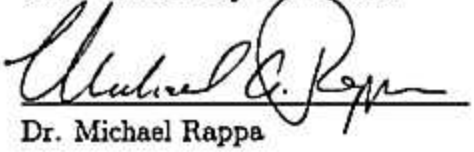
Raleigh, NC

2002

Approved By:

Dr. Annie I. Antón
Chair of Advisory Committee

Dr. W. Rance Cleaveland II



Dr. Michael Rappa

Dr. Mladen Vouk

Dedicated to my parents

Peggy Johnston Alspaugh and Thomas Atkins Alspaugh, Sr.,

and to my daughter Julia Claire Alspaugh.

Biography

Thomas Atkins Alspaugh was born in Greensboro, North Carolina on February 3, 1956, to Peggy Johnston Alspaugh and Thomas Atkins Alspaugh, Sr. He received the Bachelor of Arts in Physics from UNC-Greensboro in 1977 and the Master of Science in Computer Science from the University of North Carolina in 1980. He worked on the A-7 Project, later known as the Software Cost Reduction Project, at the Naval Research Laboratory. His first publication was as a co-author on the 1992 revision of *Software Requirements for the A-7E Aircraft*, a ground-breaking document in the fields of Requirements Engineering and Software Engineering. Dr. Alspaugh received the Doctor of Philosophy in Computer Science in 2002 from North Carolina State University. In that same year he accepted a tenure-track position as Assistant Professor in the Department of Information and Computer Science at the University of California – Irvine. He has one daughter, Julia, born in 1988.

Acknowledgements

This dissertation would not have been possible without the many who have helped, guided, and supported me, especially: my parents; my daughter Julia; my committee members Ana (Annie) I. Antón, W. Rance Cleaveland II, Michael Rappa, and Mladen Vouk; my uncle David W. Johnston who has been an example to me as a Ph.D., professor, and person; colleagues Bradford W. Mott and Laura Bode; and my colleague and dear friend Tiffany Barnes, companion along the same path.

This work was partially supported by NSF grants CCR-9875329 and CAREER Grant #530195.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> At the start I see my subject in a sort of haze. I know perfectly well that what I shall see in it later is there all the time, but it only becomes apparent after a while. *Pierre Auguste Renoir*

The primary focus of this work is the effective use of scenarios to specify and produce software. A scenario describes a sequence of events, each consisting of an actor (human or otherwise) who performs the event and the action that is performed [AABM99]. As discussed in this work, scenarios in themselves do not constitute a satisfactory software specification, and the collections of scenarios needed to describe actual software systems are difficult to manage. In this dissertation, we demonstrate that the addition of a small amount of internal and external structure to scenarios makes scenarios considerably more useful for specifying software and provides an approach for addressing the scenario management problems that arise in large collections of scenarios. Our approach is useful as a means of validating individual scenarios and collections of scenarios. This validation is not performed against an external, independent specification – which is frequently not available in practice – but with respect to internal relationships among the component parts of a scenario and with respect to relationships between scenarios describing the same system. These relationships have not been effectively utilized in the past, and some of them appear not to have been noticed at all before now. In particular, we consider the use of scenarios in requirements engineering, where they are used for expressing the purpose, requirements, and specification of the software. Requirements engineering is of considerable practical

importance due to the potentially high cost of repairing software problems resulting from errors in requirements and the far-ranging effect of those errors on the resulting software. Such errors have been called the most crippling to the resulting software and the most difficult to correct later.

1.1 Context

The context of this work is the area of software engineering, which is concerned with the production of a piece of software for a given purpose, meeting specific requirements and specifications, developed within a given time, using a given collection of resources. Specifically, this work is in the area of requirements engineering, which is concerned with the purpose, requirements, and specification of software. In requirements engineering we study the expression, validation, operationalization, and analysis of software requirements and specifications.

Requirements engineering is an area of great practical significance for software development. The most arresting and direct evidence of this is Boehm's data for the cost of repairing errors made in various phases of a software development process [Boe81]. That data shows the cost of repairing a requirements error rises by a factor of as high as 200 if the error is not detected until later phases of the development or until the software is distributed and in use, compared to its cost if repaired during the requirements process. Brooks characterizes errors in requirements as the most crippling to the resulting product and the most difficult to correct later [Bro87]. It is, therefore, during the requirements specification activities that there is the largest opportunity for gains in software quality, improvements in cost of development, and increased stability of the development plan and schedule. This opportunity has provided a motivation for focusing our efforts on improvements in software specification in the early stages of development, as we describe in the remainder of this dissertation.

1.2 Scenarios in software engineering

A scenario is fundamentally a story about a software system. The use of scenarios in software development has become increasingly common [WPJH98]. Scenarios are useful for describing the behavior required for a system; eliciting, validating, and illustrating requirements; expressing what part of a system is visible from a particular viewpoint; specifying test cases a system must pass; and other purposes. Scenarios are typically expressed in prose and thus are accessible to any reader. Because they are often expressed in the vocabulary of system stakeholders, they can be readily understood by these stakeholders as well as by analysts and developers. Each scenario narrates a sequence of events, and this focus helps the scenario author to choose the facts the scenario should incorporate, and aids the scenario reader in relating the facts the scenario presents.

Some example scenarios in various formats are presented below. Figure 1.1 is a scenario in prose form, containing a great deal of information of various kinds, some of it irrelevant. Figure 1.2 is a scenario in a tabular form, with several different kinds of information organized as attributes. Figure 1.3 is a scenario in the form of an event trace diagram, presenting only the sequence and names of events and the actors that generate and perceive the events.

Scenarios are widely used as an adjunct to more formal requirements and frequently used in place of them [ACD+01]. Evidence of this is found in the study of European software development projects by Weidenhaupt *et al.* [WPJH98], in many object-oriented software development methodologies [Boo93, JCJÖ92, RJB99, SM92], and in anecdotal evidence and our personal experience. Our research addresses this use of scenarios. We do not consider the use of scenarios for elicitation, viewpoints, or other purposes other than specifying the required behavior of a system.

Scenarios have certain definite advantages as a means of describing software behavior. Their narrative structure takes advantage of natural human skills in telling and understanding stories, and their informality makes them more accessible and appealing to those system stakeholders whose technical and mathematical backgrounds are not strong. The requirements and specification of a software system need to be discussed with, checked

Pointy-haired manager (PHM) decides to hold a meeting as soon as possible to discuss with Dilbert and Wally the assignment of cubicles. He calls Dilbert and tells him that he needs to meet tomorrow, if possible. Dilbert checks his calendar and sees that he is free all afternoon after 1:30pm. However, he needs to run some errands at lunchtime and he doesn't want to keep his boss waiting, so he tells him that he is free before 10am and after 2pm. PHM suggests that they meet at 2pm. He next calls Wally, who is not in. He leaves an e-mail message, which Wally responds to two hours later. PHM is out. PHM calls Wally first thing in the morning to schedule the meeting. Wally is free all morning, but has a series of meetings in the afternoon until 4pm. It's too late to schedule the meeting for this morning. PHM has a round of golf at 6pm, but figures having to leave promptly will keep the meeting short. He schedules the meeting at 4pm and asks Wally to tell Dilbert that the time has changed. Since Wally shares a cubicle with Dilbert, this is not difficult. Dilbert is still free at 4pm and the meeting is scheduled for then. PHM goes to the meeting room round the corner from his room, scores out the appointment for 4pm already scheduled there and enters his name with the word "important" next to it.

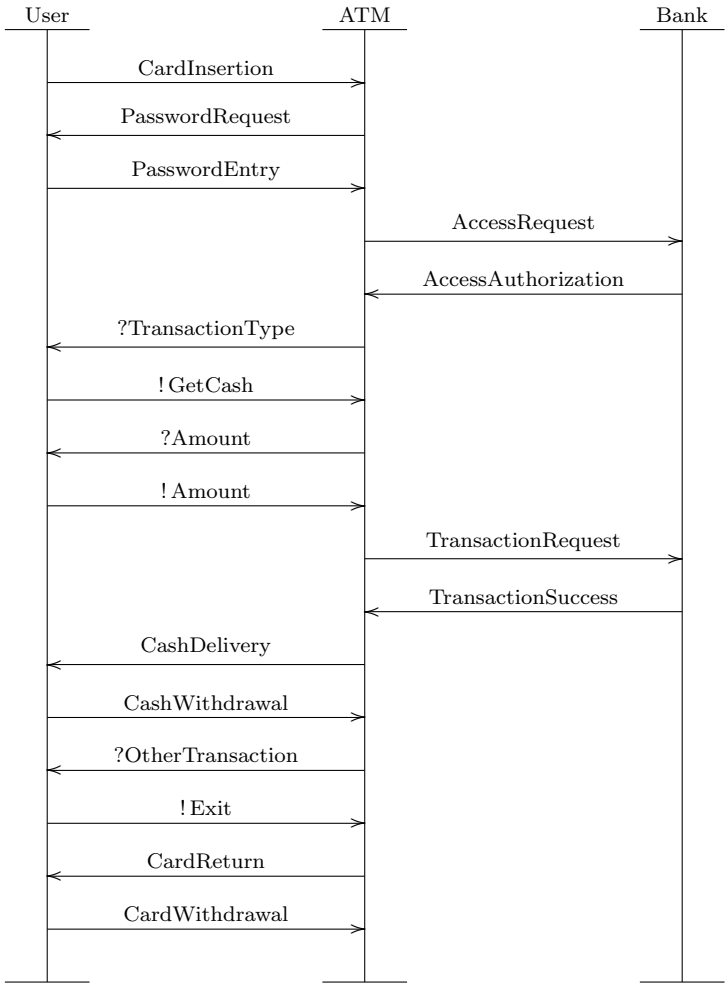
from [Pot01]

Figure 1.1: Meeting Scheduler prose scenario "Automation scenario #0 (manual case)"

Plan Meeting
Preconditions The meeting initiator can collect all necessary info from the attendees.
Normal Flow
1) The meeting initiator creates a new Meeting Schedule based on a date range in which the meeting can take place and a Meeting Type (personal/professional).
2) The meeting initiator collects from the attendees
    a) exclusion set: dates on which the attendee cannot attend the meeting
    b) preference set: dates on the attendee prefers the meeting
    c) equipment requirements
    d) location preference
3) Based on this information, use case Compute Meeting Proposal is started.
Alternate Flow
2') An Attendee might have no preferences
2") If an Attendee does not provide his preferences within the time required, he is assumed to have none.
3') If a weak or strong date conflict occurs:
    The system allows use case Modify Preferences
Postconditions An optimal proposal is made by the system.
explanation The meeting initiator collects meeting info from the attendees. A meeting proposal is made.
type User Need

from [Gor02]

Figure 1.2: Meeting Scheduler scenario "Plan Meeting"



from [LW98]

Figure 1.3: ATM scenario in the form of an event trace diagram

by, and agreed upon by the system stakeholders. Thus there is a definite advantage to expressing scenarios in a form that is directly accessible to these stakeholders, rather than (for example) in a more formal notation that must be translated or interpreted for stakeholders.

The use of scenarios for requirements or as a behavioral specification does involve a number of specific challenges, however. These challenges are listed in Table 1.1. Some of these are inherent in the form of scenarios, and others arise from the lack of an effective way to integrate the scenarios in a collection. For these reasons, a collection of scenarios does not form a good specification. These challenges will be addressed in Chapters 3 and 4.

1.3 Scenario management

Additional problems arise when a large number of scenarios must be dealt with. Collections of 50 or more scenarios are commonly needed to describe real software systems [AA01, ACD+01, WPJH98]. When the number of scenarios exceeds what one person can keep in his or her head at once, a number of challenging problems become prominent, which we list in Table 1.2 and collectively term the *scenario management problem.*

Scenario management is a long-standing problem that the requirements engineering community has been aware of in a general sense for approximately ten years, although in 1998 Jarke *et al.* stressed that the problem was "hardly addressed by research" [JBC98b]. A Dagstuhl seminar on scenario management was held in 1998 [JBC98a], from which resulted two parallel special issues of journals, one of the IEEE Transactions on Software Engineering devoted to scenario management [TSE98], and another of the Requirements Engineering Journal devoted to interdisciplinary use of scenarios [REJ98]. It is notable that, of these two special issues, only a part of one article actually addresses scenario management as it is presently understood in requirements engineering [JBC98b]. Since that time, the requirements engineering interest in scenario management has become more focused on the management of collections of scenarios and the specific problems that arise as the size of collections increases [AABM99]. It is safe to say that the scenario management problem has still not been effectively addressed.

The remainder of this section discusses the individual components of the scenario

1. *Informality* – Taking advantages of scenarios' informality without suffering unintended ambiguity and undesired implications.

2. *Integration* – Using a collection of questionably-related scenarios as a single specification, despite the partial nature of individual scenarios.

3. *Context* – Interpreting individual scenarios in the absence of sufficient context.

4. *Relationships* – Determining and expressing the necessary temporal, causal, dependency, and other relationships between scenarios.

5. *Scope* – Understanding sequences of events that are each in some scenario, but are not in the same scenario.

6. *Gaps and overlaps* – The challenge of recognizing gaps that no scenario specifies and determining the missing behavior; and of recognizing overlaps that are specified by several scenarios and reconciling the disagreements.

7. *Operationality* – Using a collection of scenarios for walkthroughs and other modelling of the system's behavior, despite insufficient connection between the scenarios.

Table 1.1: Challenges in using scenarios for specification

1. *Completeness* – To determine whether a collection of scenarios is complete, and if not, to identify in what specific ways it is incomplete.
2. *Consistency* – To determine if the scenarios of a collection are consistent with each other.
3. *Duplication* – To find any scenarios that are duplicates or near-duplicates.
4. *Organization* – To organize the scenarios in the collection in a useful way.
5. *Classification* – To classify the scenarios into useful groups.
6. *Searching* – To search the collection for a scenario describing a particular behavior, or to determine that no such scenario is present.
7. *Relationships* – To determine and maintain relationships between scenarios.
8. *Process guidance* – To decide where to devote attention next.
9. *Evolution* – To deal with each of the challenges listed above in the face of evolution.

Table 1.2: Scenario management's component problems

management problem (Table 1.2). Some of these subproblems have been identified by other researchers, and we have identified the remainder during the course of this research.

Completeness is identified as a scenario management problem by Weidenhaupt *et al.* [WPJH98]. They phrase this problem pragmatically in terms of deciding which kind of scenario to develop when and deciding when to stop. We express it more abstractly in terms of defining what is meant by completeness in a particular context, determining completeness of a scenario collection, and identifying particular ways a collection is incomplete in order to guide the process of improving the collection. We say a specification for a system is ***complete*** if the system can be implemented based on the specification without the need for additional significant decisions that increase the scope or level of detail of the specification. Completeness is relative to some standard of significance. We note that completeness is an intrinsically moving target in practice, because the decisions of what is and is not essential behavior for a software system are usually a result of producing a requirements specification, not an input to that process. This fact is independent of whether scenarios are used to express the specification.

Determining completeness of a scenario collection is difficult for a number of reasons. Perhaps the most fundamental difficulty is that scenarios are inherently partial descriptions of a system's behavior, so that many scenarios are needed to describe a system, and important properties of the system's behavior are implicit and embodied across many scenarios [LW98]. Other contributing difficulties have been identified during the course of this research. The lack of a clear boundary for the behavior described by a collection of scenarios can make it difficult to decide whether or not a particular behavior is specified, allowed, or forbidden. In practice this frequently occurs between scenarios, where behavior that one analyst may consider as obvious and implied is often not at all obvious to another. Determining if a particular scenario describes all of a specific behavior is problematic because of the ambiguity and accidental implication inherent in prose narrative scenarios. In the simplest case, determining if a specific behavior is not described in any of n scenarios requires reading each of them carefully. Determining if there is missing behavior in a gap between two scenarios requires examining all $(n^2 - n)$ ordered pairs of scenarios. Finally, there are no well-defined criteria that express what it means for a collection of scenarios

to be complete. These reasons contribute to making it extremely difficult to determine completeness for a scenario collection and to identify specific ways in which an incomplete collection is not complete.

Consistency between the scenarios in a collection is a widely shared concern. Paraphrasing van Lamsweerde *et al.*, we say a collection of scenarios is inconsistent if there is no way to implement those scenarios all together [LDL98]. The scenarios can fail to be consistent in a number of ways; the following list is not exhaustive.

- A name is given no designation (*used but not defined*). For example, scenario "Submit Quote" uses a scenario named "Choose Approver", but there is no scenario "Choose Approver" in the collection.
- One name designates two concepts (*designation clash*). For example, "order" is used to mean "an order from a customer to the Company" and also "an order from the Company to a supplier."
- Two names designate the same concept (*terminology clash*). For example, "Planner" and "Plant Planner" both refer to the same role.
- Two scenarios describe conflicting behaviors for the same situation (*nondeterminism*). For example, scenario "Submit Quote" assigns a (possibly different) approver to each item in a quote, while scenario "Approve Quote" has a single approver handle the entire quote.
- A scenario has a precondition that is not fulfilled by any other scenario(s) in the collection (*unsatisfiability*). For example, the precondition of scenario "Get Address Information" requires that the system be display the Address Screen, but no scenario in the collection displays that screen.

We note that deciding if two scenarios in narrative prose are consistent requires reading both of them carefully and comparing them in detail, and that the number of pairs of scenarios in a collection of n scenarios is $O(n^2)$.

Duplication within a scenario collection was identified by us as a scenario management problem [AABM99]. Duplication occurs if two scenarios are textually the same or if

they describe the same behavior, in which case only one scenario is needed and the presence of two involves the risk that one of the two will be changed but the other won't, resulting in an inconsistency. Near-duplicates are worse than duplicates, in that they already involve an inconsistency that will be difficult to spot since the two scenarios are nearly the same. Identifying duplicates or near-duplicates can be characterized as $O(n^2)$.

Organization of a collection of scenarios refers to systematically arranging all the scenarios so that each scenario has a place relative to the others. An organization scheme can make it easier to find particular scenarios, to see which scenarios are related, and to tell if scenarios are missing. An organization scheme can also form the basis for systematically naming the organized elements, by enumerating them linearly (1, 2, 3, ...) or hierarchically (1.1, 1.2, 1.2.1, ...) or in some other fashion that follows the structure of the organization. Organizing a collection of scenarios is characterized as "hard" by Jarke *et al.* [JBC98b]. They attribute this to the fluid character of scenarios. We believe it is more due to the lack of any natural or helpful complete ordering on scenarios and the difficulty of determining relationships between scenarios that might lead to a useful partial ordering.

Classification of scenarios in a collection differs from organization in that a classification divides scenarios into groups, while an organization provides places for individual scenarios. Classification of scenarios into groups that are equivalent in some sense (equivalence classes) was first proposed by us [AA01]. We believe that partitioning into equivalence classes was not considered earlier because it is so difficult to determine whether two scenarios are equivalent in any useful sense. The task of partitioning an entire collection of prose scenarios into equivalence classes would be a daunting task. As an example of related work with equivalence relationships, Breitman and Leite discuss an equivalence relationship on scenarios as part of their framework for scenario evolution, but do not use the relationship to produce equivalence classes; in that framework, two scenarios are equivalent if they share the same episodes and actors [BdPL98]. The benefits of having equivalence classes of scenarios are very substantial for scenario management and for other tasks involving scenarios, since they allow all of the scenarios in a class to be treated as interchangeable so that a task involving one of the scenarios in a class need not be repeated for the others.

Searching a collection for a particular scenario, or to determine that no such sce-

nario is present, is a fundamental task that is repeated over and over in working with a collection of scenarios. Several aspects of this problem make it difficult: the problem of adequately characterizing the scenario that is sought; the problem of comparing this characterization against a candidate scenario; and the scale issue of having to make this comparison against (potentially) every scenario in a collection.

Relationships between individual scenarios are difficult to determine for some of the same reasons that consistency and classification are difficult. Some examples of scenario relationships are "is equivalent to", "depends on", "overlaps with", and "is consistent with". We note that it is $O(n^2)$ to determine which scenarios in a collection of size n are related by a particular pairwise relationship.

Process guidance is cited as a major problem in Weidenhaupt *et al.*'s survey of the use of scenarios in software development, specifically in choosing what scenario to create next and in deciding when to stop developing scenarios; both these are issues of completeness (see above) [WPJH98]. We believe that process guidance is also needed for improving consistency and for eliminating and preventing duplication.

Scenario evolution, the gradual change of the scenarios describing a system during the system's development, is cited as a scenario management problem by Jarke *et al.* [JBC98b]. They specifically refer to maintaining relationships in the face of evolution and note that no good methodology for this was known at that date. We argue that the evolution of a collection of scenarios complicates any approach to addressing the other problems of scenario management.

1.4 Motivation

Our research approach was influenced by a number of motivations. The first was the realization that a collection of scenarios does not form a good specification. Using a collection of scenarios as a specification is somewhat like attempting to assemble a jigsaw puzzle whose pieces all have rounded edges; it is impossible to tell which pieces are adjacent, whether all the necessary pieces are present, or how the pieces go together to form a complete picture. Our intuition was that it would be possible and beneficial to add a small but useful

degree of structure to individual scenarios and to collections of scenarios, by analogy to the addition of a small but useful degree of structure added to the requirements document for the A-7 [AFB+92]. A guiding principle throughout this research effort has been that one person can only think about a limited amount of information at once, and that the attention of an insightful, skilled person is the most important resource for any software project, and the resource whose limits shape the development process most stringently [Dij72]. Object-oriented types and their relationships, and especially the concept of type as a specification of behavior [Ame87], were a touchstone for the development of the concepts and structures described in this dissertation. Finally, shimmering in the distance but always just out of reach, the grail of an operational specification based on scenarios gave constant direction to this work.

1.5 Research classification of this work

Shaw provides several ways of characterizing software engineering research, in terms of what she describes as research settings, research products, and validation techniques [Sha01]. Table 1.3, Table 1.4, and Table 1.5 summarize these characterizations.

The settings of this research, in terms of Shaw's characterizations (Table 1.3), are feasibility, characterization, and method/means. Some of the corresponding questions are:

- Can scenarios be formalized to a small degree that will still produce useful results for scenario management?
- What is the simplest degree of temporal integration of scenarios that will still result in an operational specification based on scenarios?
- How can we efficiently determine relationships among a large collection of scenarios?
- What is a better way of expressing the context of a scenario?
- How can we automate the process of identifying inconsistency and incompleteness within individual scenarios, and across collections of scenarios?

Research settings	Sample questions
Feasibility	Is there an X, and what is it? Is it possible to accomplish X at all?
Characterization	What are the important characteristics of X? What is X like? What exactly do we mean by X? What are the varieties of X, and how are they related?
Method/Means	How can we accomplish X? What is a better way to accomplish X? How can we automate doing X?
Generalization	Is X always true of Y? Given Y, what will X be?
Selection	How do I decide between X and Y?

Table 1.3: Shaw's characterizations of software engineering research settings

Research product	Research approach or method
Qualitative or descriptive model	Organize and report interesting observations Create and defend generalizations from real examples Structure a problem area Formulate the right questions Do a careful analysis of a system or its development
Technique	Invent new ways to do some task Develop a technique to choose among alternatives
System	Embody the result in a system Use the system development as both a source of insight and a carrier of results
Empirical predictive model	Develop predictive models from observed data
Analytic model	Develop structural (quantitative or symbolic) models that permit formal analysis

Table 1.4: Shaw's characterizations of software engineering research products

The research approaches used in this research, in terms of Shaw's characterizations (Table 1.4), are

- to create and defend generalizations from real examples, drawn from the Meeting Scheduler and the Enhanced Messaging System;
- to structure a problem area, by establishing specific relationships that exist within and among scenarios;
- to formulate the right questions to ask in order to guide the reduction of inconsistency and increase of completeness of a collection of scenarios;
- to conduct a careful analysis of the requirements engineering phase of the development of several systems;
- to invent a new way of examining scenario consistency, suitable for automated tool support;
- to embody research results in two automated tools (SM^{a}RT and SN^{e}AT); and
- to develop a structural (symbolic) model permitting formal analysis of scenario-based specifications.

The corresponding research products are descriptive models, techniques, systems, and analytic models.

The validation techniques used in validating this research, in terms of Shaw's characterizations (Table 1.5), are

- persuasion, on the grounds of techniques and examples presented in this dissertation;
- implementation, of specifications using these techniques;
- implementation, of software tools implementing our relationships and techniques;
- evaluation, with respect to the criteria of usefulness for each of the challenges in using scenarios for specification (Table 1.1) and the components of the scenario management problem (Table 1.2); and

- analysis, of analytic formal models of specific systems.

1.6 Overview of this work

This work addresses the challenges in using scenarios for requirements specification and the component problems of scenario management listed in Tables 1.1 and 1.2 respectively. We adopt a two-pronged approach, and term its two complementary parts the *syntactic* and *semantic* approaches.

The syntactic approach (Chapter 3) focuses on the means by which scenarios are expressed and specifically on adding a small amount of structure to their form of expression. It uses a group of mutually-supporting techniques that we collectively term *integrated syntactic analysis (ISA)*. The semantic approach (Chapter 4) focuses on what scenarios signify, especially in relation to each other, and on providing a structure that accomodates all the scenarios describing a system. We call this structure a *scenario network* , and we make use of scenario networks with techniques we collectively term *scenario network construction and analysis (SNCA)*. The syntactic approach may be characterized as *internal*, since it is concerned with the internal structure of scenarios; the semantic approach may be characterized as *external*, concentrating on the external context of scenarios.

In terms of the challenges for specification using scenarios and the component problems of scenario management, our work uses the syntactic approach as a basis for reducing unintended ambiguity, identifying and retaining desired dependency relationships between scenarios, and searching for a particular scenario and identifying duplicate and near-duplicate scenarios. Our work uses the semantic approach to address all the challenges and problems in Tables 1.1 and 1.2 except unintended ambiguity and searching for a particular scenario. The complementary syntactic and semantic approaches form a basis for attacking the problems of scenario management and the challenges of using scenarios as requirements specifications.

The remainder of this dissertation is organized as follows. Chapter 2 discussses the most relevant related work and how our work builds upon it. Chapter 3 presents the syntactic approach, which is based upon a formalization of scenarios and uses glossaries of

Technique	Grounds
Persuasion	A technique, design, or example
Implementation	Of a system or technique
Evaluation	With respect to a descriptive model With respect to a qualitative model With respect to an empirical quantative model
Analysis	Of an analytic formal model Of an empirical predictive model
Experience	Expressed in a qualitative or descriptive model Expressed as decision criteria Expressed in an empirical predictive model

Table 1.5: Shaw's characterizations of software engineering research validation

terms, automated discovery and support of shared episodes, and similarity measures that this formalization allows. The software tool SMaRT (Scenario Management and Requirements Tool) implements this approach. Chapter 4 presents the semantic approach, which uses scenario networks to integrate collections of scenarios in order to express the context of each scenario. The construction and analysis of scenario networks provides means for validating the component scenarios, directing analysts' attention to problem areas, identifying relationships between scenarios, and dealing with scenario evolution. The software tool SNeAT (Scenario Network Analysis Tool) implements this approach. Chapter 5 discusses the case studies done to evaluate and support these two approaches. Chapter 6 summarizes this work and its application to the problems and challenges of the use of scenarios for specification and the management of scenarios in requirements engineering. Appendices present the software tools SMaRT and SNeAT and a summary of the data from each of the case studies.

Chapter 2

Related work

Only in silence the word,
only in darkness the light,
only in dying life:
bright the hawk's flight
on the empty sky.
Ursula K. LeGuin

Our work builds upon research in a number of areas in requirements engineering. Each section of this chapter provides an introduction to one of those areas, summarizes the most pertinent related work, and briefly discusses how our research builds upon that work.

2.1 Scenario management

We use the term *scenario management* to refer to the management and administration of a collection scenarios. Scenario management addresses issues that arise as the number of scenarios for a system increases (see Table 1.2, page 9), such as determining whether a collection of scenarios is complete; determining whether the scenarios are consistent, and managing the inconsistencies between them; detecting and eliminating duplicate and near-duplicate scenarios; organizing and classifying the scenarios for a system; finding the relationships among scenarios, and tracing dependencies among scenarios and between scenarios and other artifacts; process guidance; and scenario evolution. Scenario management is needed for any system with more scenarios than one person can keep track of in his

or her head.

These issues have received attention but have not been addressed effectively. In their survey of the use of scenarios in industry, Weidenhaupt *et al.* note that the creation, documentation, and validation of scenarios is a substantial effort in itself [WPJH98]. Jarke *et al.* specifically note that little research addresses the problems that arise in managing a large set of scenarios [JBC98b]. The term "scenario management" was used with a broader meaning in requirements engineering before about 1999, with a sense that included the use of scenarios to manage the unfolding of events, or to examine the results of alternate courses of action. This broader use reflects the use of the terms in other areas, such as that of business forecasting [GFS98]. The Dagstuhl Workshop on Scenario Management took this broad view [JBC98a], as did the resulting special issue of the *IEEE Transactions on Software Engineering* [TSE98] devoted to scenario management, and the related special issue of the *Requirements Engineering Journal* [REJ98] devoted to interdisciplinary uses of scenarios. The articles in these two special issues address the use of scenarios for various purposes in various situations, rather than the management of scenarios themselves (with the exception of Jarke *et al.* [JBC98b], which contains a section on "scenario management in the large"). However, since that time the use of the term "scenario managment" in requirements engineering has shifted to more specifically describe the management of collections of scenarios. Alspaugh *et al.* defined the term with that meaning and proposed an approach to scenario management based on glossaries, episodes (scenario fragments that appear in several scenarios), and measures of similarity between scenarios [AABM99]. Some of this work is an early version of the work presented in Chapter 4. The approach uses a purely syntactic view of scenarios to support analysts as they work to make their scenarios consistent; trace and maintain dependencies among scenarios; look for scenarios that address a particular behavior; and determine completeness of a group of scenarios [AABM99].

Scenario formalization (see Chapter 4) addresses the problems of finding duplicate or near-duplicate scenarios, searching for a particular scenario, determining and maintaining dependency relationships among scenarios that manifest themselves as episodes (shared subsequences of events), and the effect of scenario evolution on these problems. Scenario networks (see Chapter 4) address all of the problems of scenario management except that

of searching for a particular scenario. This approach provides criteria and process guidance for addressing the completeness and consistency of a collection of scenarios. It provides a means of detecting duplication between scenarios that goes beyond what our scenario formalization provides, by considering duplication from the point of view of the effect of the behavior described by the scenarios. The networks constitute a structure for organizing a collecion of scenarios. The structural relationships arising from a scenario network offer a useful approximation of important relationships among scenarios, provide a basis for classifying scenarios into equivalence classes and for ordering these classes in a behavioral subtyping relationship, and support the maintenance of desired relationships as scenarios evolve.

2.2 Scenario process guidance

Scenario process guidance refers to methods and heuristics that guide the process of creating and refining scenarios. Such process guidance helps analysts identify areas of system behavior that no existing scenario addresses; locate scenarios that conflict with each other; and identify scenarios that either do not sufficiently specify important behavior, or that over-constrain by specifying unnecessary or irrelevant details. Process guidance is important in scenario management (discussed in the preceding section) but merits its own section here due to the significance and amount of work in this area.

Weidenhaupt *et al.* observed that most developers viewed the creation of scenarios as a craft rather than an engineering activity, and that effective theories and heuristics for guiding the creation and refinement of scenarios are needed [WPJH98]. Several researchers have proposed heuristics and theories to support them. Potts *et al.* discuss the refinement of scenarios through the technique of scenario walkthroughs in their Inquiry Cycle [PTA94]. Sutcliffe, Maiden, *et al.* attack the problem of missing scenarios with a method supported by their CREWS-SAVRE tool. In this method, new scenarios are automatically generated for consideration by an analyst, using a library of standard models and alternative sequences of use case events [MMMR98, SMMM98]. Rolland and Ben Achour present a process that begins with a context and initial scenario for a use case, and then guides the capture and

completion of new scenarios and their integration into the use case [RBA98]. The work on viewpoints discussed in Section 2.7 provides process guidance for improving consistency among scenarios.

As discussed in Chapter 4, process guidance is an important benefit of the use of scenario networks, especially for dealing with incompleteness and inconsistency and scenario evolution. The process of constructing a scenario network also guides the discovery and resolution of inconsistencies between scenarios, and of gaps and missing scenarios. We note that the kind of consistency enforced between scenarios by scenario networks is different than that provided by viewpoints: scenario networks compel consistency at the boundaries of scenarios that do not overlap, while viewpoints compel consistency between overlapping scenarios in the areas in which they overlap.

2.3 Preconditions and postconditions for scenarios

Pre- and postconditions for scenarios express what each scenario requires and achieves. A scenario's precondition expresses what the scenario expects to be true when it begins, and a scenario's postcondition expresses what the scenario guarantees to be true when it concludes (if its precondition was met).

Pre- and postconditions are widely used in many contexts. A number of researchers have attached pre- and postconditions (or, equivalently, initial and final states) specifically to scenarios. Rolland and Ben Achour use initial states of agents and final states of episodes to guide the writing of use cases involving these agent and episodes [RBA98]. In their work, an episode is defined to be of "a flow of actions" which can involve sequence, concurrency, iteration, and alternation, together with the possible final states that the flow of actions can reach. Rolland *et al.* attach initial and final states to scenarios in their *L'Ecritoire* tool in support of a heuristic to guide the search for additional goals [RSBA98].

Scenario networks use pre- and postconditions for scenarios to restrict the sequences of scenarios expressed by a scenario network.

2.4 Similarity measures

A *similarity measure* is a function that compares two entities and produces a numerical result indicating the degree to which the entities are similar. It is considered most convenient if the result is normal, so that complete dissimilarity produces a result of 0 and complete similarity produces a result of 1. Similarity measures that are of interest in requirements engineering may be divided into syntactic measures, that compare the representations of the two entities, and semantic measures, which compare some sort of semantic representations instead.

Semantic similarity measures have been used extensively for requirements reuse; for example, *semantic similarity* has been used with conceptual graphs [RM93], analogical reasoning on a pre-existing case base [ML97], and on generic domain models [MS96]. A semantic measure is appropriate for these applications because they reapply the semantic structure of pre-existing requirements to new contexts.

Several researchers have worked in the area of syntactic similarity measures for requirements analysis. Natt och Dag *et al.* examine the syntactic representation of requirements in the form of prose, and use a statistical analysis of the preprocessed text to determine similarity of requirements [NoDRC+01]. Like us, they also use their similarity measure as an indicator of less accessible features, in their case relationships between requirements. Park *et al.* use similarity measures that examine syntactic similarity between prose requirements (in Korean), again using statistical analysis of preprocessed text [PKKS00].

There are a number of approaches for calculating syntactic similarity. Tversky originated techniques based on the consideration of entities as collections of attribute values [Tve77]. The scenario similarity measure presented by Alspaugh *et al.* and in this dissertation are based upon Tversky's work [AABM99]. String similarity measures that take into account the sequence of elements being compared, and that are applicable to strings of different lengths (as scenario event sequences may be), are most effectively based on metrics such as the Levenshtein distance that is based on the minimum number of edits that will convert one string to the other [Kru83]. Calculation of minimum numbers of edits

makes use of dynamic programming algorithms; for the relatively short event sequences that are generally found in collections of scenarios, the complexity of such algorithms is not an issue. Syntactic similarity of prose can be calculated by various statistical approaches [NoDRC+01, PKKS00].

We use syntactic similarity measures because we work with representations that have been formalized so that syntactic similarity is straightforward to interpret and produces results that are useful approximations of the judgment of a careful reader. Because we represent scenarios as collections of attribute values, we concentrate on a group of similarity measures based upon the work of Tversky [Tve77]. These measures are not especially novel, but the use of slightly formal scenario representations that make Tversky measures appropriate is. Similarity measures that use edit distances to compare similarity of event sequences are a natural extension of the work presented here. Statistical approaches are not relevant for our work since we do not compare scenarios in prose form.

2.5 Scenario relationships

A *scenario relationship* expresses equivalence, ordering, causality, inheritance, or some other connection between two scenarios. Examples of scenario relationships between any scenarios S_A and S_B are given in Table 2.1.

Jacobson *et al.* discuss the "uses" and "extends" relationships in their original work on use cases [JCJÖ92], and in later work on the Unified Modeling Language (UML) [RJB99]. Simons and others note that the Jacobson and the UML definitions and redefinitions of

S_A *uses* S_B
S_A *extends* S_B
S_A *includes* S_B
S_A *overlaps with* S_B
S_A is *equivalent to* S_B
S_A is a *subset of* S_B
S_A *depends on* S_B

Table 2.1: Scenario relationships in related work

"uses" and "extends" (and now "includes") have been inconsistent, poorly understood, and problematic, because they induce arbitrary *goto*-like jumps in the flow of control, "extends" is used in the literature and in practice for purposes for which it is not adequate, and neither relationship is sufficient to address long-range dependencies between use cases [Sim99]. Breitman and Leite define and use relationships between scenarios (overlap, equivalence, and subset) to classify and guide the evolution of scenarios [BdPL98]. Their definition of scenario equivalence is "when two scenarios share the same episodes, involve the same actors, but handle different restrictions and exceptions.". Alspaugh *et al.* discuss the importance of dependency relationships between scenarios, and their preservation as the scenarios evolve [AABM99].

Scenario networks provide a basis for several new scenario relationships which we introduce in Chapter 4. These relationships are used in the construction and refinement of the scenario network and its constituent scenarios.

2.6 Glossaries and consistency

A *glossary* is a list of terms with a definition of each term. Glossaries are needed wherever specialized terms are used, or wherever terms are used with meanings more specific than in their ordinary usage. Their use in technical writing is widespread, indeed almost universal.

In the area of requirements engineering, glossaries are widely used and their benefits are generally recognized. We cite a few researchers here in illustration. Heitmeyer *et al.* discuss the use of glossaries (among other techniques) in the consistency checking of requirements specifications [HLK95]. Weidenhaupt *et al.* state that glossaries add a common understanding of terms used in scenarios [WPJH98].

Our work makes use of glossaries as the foundation upon which our structured representation of scenarios is built. We use glossaries in the typical way, giving definitions of terms used to express scenarios. Our approach is distinctive in that we also use glossaries to define the possible values of each kind of scenario attribute, and we then use these sets of possible values to determine which values are equal (values that refer to the same glossary

entry are equal and values that do not are unequal; no other possibilities are considered).

2.7 Viewpoints

A *viewpoint* is a partial specification of a system from a particular perspective [EN95, NKF94]. This is usually the perspective of a single actor who interacts with the system. Viewpoints are used to express the separate perspectives of the various actors of a system, and provide a basis for identifying and eventually reconciling inconsistencies between these perspectives.

Finkelstein, Nuseibeh, Easterbrook, and other researchers discuss viewpoints in requirements engineering and a formalization of them termed ViewPoints [EN95, NKF94]. For a complex system, ViewPoints provide a framework for separating the concerns of the various viewpoints. The specifications from the various viewpoints may be expressed using different specification languages and methods supported by different tools. Each viewpoint is defined in terms of the editing and consistency checking actions appropriate to it, and these actions and the inconsistencies associated with the viewpoint provide process quidance for eliciting and elaborating the specification [NKF94]. If the specification is expressed in terms of a state transition system, then each viewpoint may have its own subset of states and of transitions between them. A particular system state or transition between states may be visible from one viewpoint but not from another. Reconciling the inconsistencies between the specifications from each of the viewpoints produces an integrated specification consisting of all the system's states and transitions [EN95].

Scenario networks are similar to viewpoints in that they produce a unified system specification out of a number of smaller specifications. Viewpoints are most effective where these smaller specifications overlap at least to some extent, and viewpoints integrate the specifications by unifying common elements shared among specifications. The overlaps between specifications are the locus where viewpoint integration takes place. In contrast, the component scenarios of a scenario network are required to not overlap; in a scenario network, overlaps between scenarios are eliminated so that each part of the system's behavior is expressed in exactly one place. A scenario network integrates its specifications by connecting

the exit point of each scenario with the entry points of all scenarios that can follow it. The connections between scenario exits and entries are the locus where integration of a scenario network takes place.

2.8 Scenario integration

We use the term *scenario integration* to refer to any process that unifies a group of scenarios into a single larger entity. Scenario integration is not frequently practiced; scenarios are generally used as individual partial specifications. This practice exacerbates the problems of missing scenarios (and completeness in general), inconsistency between scenarios, and lack of conceptual unity of the system being described.

Dano *et al.* integrate scenarios based on the temporal relationships between them, and construct corresponding Petri nets, as part of their work on formalizing domain-expert use cases and then producing object type state diagrams [DBB97]. A *use case map* (UCM) integrates use cases to provide a whole-system specification [BC96, Buh98]. The use cases are expressed as causal sequences of responsibilities and denoted graphically as a graph with responsibilities attached. The original emphasis was on binding responsibilities to system components and elicitation of requirements and design information. Feature interaction is examined visually by inspection of the UCM notation. More recently, UCMs have been used to express whole-system behavior, and formalized by (manual) translation into the specification language LOTOS [ALBG99]. Sendall's *operation schemas* are system operations, corresponding to Jacobson's transactions that make up use cases, augmented by pre- and postconditions (and other information) [SS00]. The conditions express when each operation can occur and its effect. The operation schemas for a system are thus implicitly integrated into a single specification. The use cases that contain the system operations become emergent phenomena of the operation schemas for the system. A *use case* itself can be considered to integrate some of the scenarios for a system, since it can describe both a principal sequence of actions and also possible variants for alternate orderings, exceptional cases, or error handling [OMG99]. Only scenarios that are variants of each other are contained in a single use case, however, so its scope is limited. Viewpoints provide a means of

integrating overlapping scenarios, as discussed in Section 2.7 [EN95, NKF94].

Scenario integration is one of the primary results and benefits of creating a scenario network. Scenario networks are similar to use case maps in that both provide a graphic notation that indicates how scenarios (use cases) can occur temporally. Scenario networks provide techniques for assessing and improving completeness of the collection of scenarios and consistency among the scenarios.

2.9 Concurrency between scenarios

Concurrency in requirements specification has been addressed by, for example, the CoRE method [FFK94], the modelling approach of Coleman *et al.* [CEG+90], and the Specification and Description Language (SDL) [ITU99a] and Message Sequence Charts (MSC) [ITU99a, ITU99b]. Surprisingly little research has focused on *concurrency between scenarios*, however. A scenario is a sequence of actions and this sequence of actions tends to obscure the fact that in general a scenario occurs concurrently with other activity in the system. Desharnais *et al.* use a state-based formalization of scenarios to explicitly represent concurrency between scenarios [DFKM98]. The graphic notation of UCMs (Section 2.8) expresses the concurrency that is desired between use cases, and has been used to detect feature interactions [BC96]. There is a large body of established work on concurrency and concurrent systems [Mil89, CPS93, CES86], but it has been little applied to concurrency between scenarios.

As discussed in Chapter 4, scenario networks begin to address concurrency between scenarios and this continues to be an area of focus for future work.

2.10 Goals and scenarios

Antón defines a ***goal*** in requirements engineering as a target for achievement, or a high-level objective [Ant97]. Goals are used as a higher-level expression of the requirements of a system. They are needed to express why the particular requirements of a system were chosen, and provide the motives and rationales that lie behind the requirements.

Goals have been used as a means of organizing requirements and scenarios. Antón uses a hierarchy among the goals for a system to structure and organize the requirements and scenarios for that system [Ant97]. Maiden *et al.* have proposed using goals and goal obstacles to organize collections of scenarios [MMMR98]. Dardenne *et al.* use goals as a basis for eliciting and elaborating requirements [DLF93].

In our formalization of scenarios, goals are treated as an attribute that scenarios can possess. However, we also make use of a scenario's goal as a high-level summary of the semantic result of the events described by the scenario.

2.11 Scenario evolution

Evolution refers to the process of gradual change of requirements and scenarios. This change begins during the course of requirements elicitation and expression and continues throughout the entire software development process. Evolution is a component of the scenario management problem, and inevitably exacerbates all the other problems of scenario management.

Breitman and Leite's scenario evolution framework gives three relationships between scenarios (overlap, equivalence, and subset) and maps them onto seven operations that express the evolution of the scenarios [BdPL98]. These operations are merging, encapsulation, and consolidation (on two scenarios), and splitting, inclusion, modification, and exclusion (on one scenario). Viewpoints (see Section 2.7) provide a framework in which to manage inconsistencies in the face of evolution [EN95]. The Inquiry Cycle provides a model which supports the process of requirements evolution and directs the analysts' response to that evolution [PTA94].

Our work addresses scenario evolution by providing a means of determining several useful scenario relationships and maintaining them during the evolution of the related scenarios. In Chapter 4 we define several novel relationships that may be automatically determined using the scenario structures we present. We also present a practical way of determining, highlighting, and maintaining certain dependency relationships between scenarios.

2.12 Specification formalisms for modelling

A *specification formalism* is a basis for constructing a formal model whose behavior mimics important aspects of the system it specifies, but which is abstract, compact, and amenable to analysis. Whereas requirements describe a system by presenting its properties, a model describes a system by behaving like or simulating it. There are a wide variety of specification formalisms, each with its own strengths and weaknesses and areas of greatest applicability. We discuss the formalisms that are most relevant to scenario networks: Petri nets, high-level message sequence charts, and extended finite state machines.

A *Petri net* is a directed graph with two kinds of nodes, *places* and *transitions*, and with arcs that run either from places to transitions or from transitions to places [Pet77]. If an arc runs from a node c to another node d (whether a place or a transition) then c is said to be an input to d, and d an output from c. The state of a Petri net is indicated by the presence of tokens in one or more places, and a token moves from one place to another when a transition between the places fires. A transition can fire when all of its input places are occupied by tokens, and when a transition fires all of its output places receive tokens. Thus, concurrency is inherent in Petri nets and they are particularly useful in modelling concurrent systems. Petri nets are supported by almost four decades of research and a number of software tools [Pet77].

High-level Message Sequence Charts (HMSCs) are a formalism for describing systems in terms of Message Sequence Charts (MSCs) [ITU99b]. A Message Sequence Chart is a graphical representation of sequences of messages transmitted between instances (systems, components, processes, etc.) [ITU99b, RGG96]. A basic MSC is roughly comparable in function to a scenario, with messages or actions of an MSC corresponding to events of a scenario. MSCs may be far more complex than scenarios, however, with timers and quantified times, conditions for restricting message sequences, alternation, iteration, concurrency, references to other MSCs, and various other features for specifying paritally or totally ordered sequences of messages. A High-level Message Sequence Chart connects individual MSCs, not using the MSC notation as its name suggests but an unrelated notation that indicates sequential composition, alternation, iteration, concurrency between two

HMSCs, and recursive composition in which a node of an HMSC can itself be an HMSC [ITU99b, MR97]. MSCs and HMSCs are commonly used in describing telecommunications systems.

An *extended finite state machine* (EFSM) is a finite state machine (FSM) whose states have been augmented by variables [Wie98]. The global state of the machine then consists of its explicit state (one or more of the nodes of its diagram) plus its extended state (the values of the variables). The variables may be external to the FSM or local to it, in which case their scope may be all the states and transitions or some subset of them. The values of the variables may be changed by the explicit state, transitions between extended states, or possibly from outside the EFSM; the values of the variables may be used as guards for each transition. EFSMs are widely used to describe many sorts of systems, and occur in several variants, including Statecharts [Har87, HN96] and state diagrams in the Specification and Description Language (SDL) [ITU99a, Bræ96]. Statecharts extend the basic idea of EFSMs with several sorts of composition, including concurrency and clustering of states to ameliorate state explosion. They are widely used (for example in UML) and software tools such as StateMate are available to support them. SDL was developed especially for telecommunications and embedded systems (as were MSCs and HMSCs), and is most commonly employed in those domains.

A scenario network is an extended finite state machine that has been adapted to express temporal and causal relationships between scenarios. Its nodes are scenarios, and its transitions represent possible paths from one scenario to another. Paths that initiate a new instance of concurrency are marked to distinguish them. Unlike the transitions of an FSM, the transitions of a scenario network are not labelled with inputs, because the event that triggers a transition is part of the scenario the transition leads to. Each scenario is guarded by a precondition expressed in the primitive terms of the network (corresponding to the extended state of an EFSM), and has a postcondition expressing the scenario's effect on the primitive terms. Scenario networks express sequences of actions, as do Statecharts, MSCs, and HMSCs, and like Statecharts and HMSCs the graphic notation for scenario networks is based on that of FSMs and transition systems in general. We limit scenario networks to a simpler structure than that of Statecharts, MSCs, and HMSCs in

order to concentrate on relationships between scenarios, and to focus on the requirements engineering and process challenges that are made clear and can be mitigated even with this simple structure. Scenario networks differ from Petri nets in a number of ways, notably in that scenario networks do not possess separate transition and place nodes, and the form of a scenario network is directly traceable to significant aspects of the behavior of the system it describes.

2.13 Summary

This chapter has presented the related work and described how our work on scenario challenges and scenario management builds upon it. The next two chapters discuss our work directly. Chapter 3 presents the syntactic or internal approach to scenario formalization. We show means for formalizing scenarios to a relatively small degree, and some ways of capitalizing on this formality to address challenges in the use of scenarios. These strategies are compatible and mutually self-supporting, so that their use is relatively effective for the effort expended. Chapter 4 presents the semantic or external approach to scenario formalization. We define the concept of scenario networks, and discuss how scenario networks may be used to attack the challenges of using scenarios and the problems of scenario management. We note how scenario networks build on some of the techniques from the syntactic approach to scenario formalization. We also show how the use of scenario networks in practice can make use of requirements work that should be done in any case, so that the incremental cost of using scenario networks is low, especially compared to the benefits that result.

Chapter 3

Scenario formalization

> There are three possible parts to a date, of which at least two must be offered: entertainment, food, and affection. It is customary to begin a series of dates with a great deal of entertainment, a moderate amount of food, and the merest suggestion of affection. As the amount of affection increases, the entertainment can be reduced proportionately. When the affection *is* the entertainment, we no longer call it dating. Under no circumstances can the food be omitted.
> *Judith Martin ("Miss Manners")*

The informal narrative character of scenarios has definite advantages for requirements engineering, and at the same time introduces a number of difficulties (see Table 1.1, page 8). One way to summarize the difficulties addressed in this section is to say that answering interesting questions about a collection of prose scenarios requires careful reading, focused thought, insight, and skill. When the number of scenarios to consider for a single system is large (50 or more), these difficulties are exacerbated and additional problems are introduced (see Table 1.2, page 9). A large collection of scenarios is too much for one person to keep in mind at the same time [Dij72]. Some of the problems of scenario management are amenable to an exhaustive approach that covers every scenario or scenario pair. Two examples are searching for a scenario describing a particular behavior and determining if the scenarios are (pairwise) consistent. But these are daunting tasks for a large scenario collection, and the tedium of considering all of them increases the likelihood that something significant will be overlooked. In this chapter, we consider several related approaches based on the underlying syntax of scenarios:

- structuring scenarios as event sequences plus attribute values;
- structuring events as an actor-action pair;
- using glossaries to define attributes and their values;
- using glossaries of words and phrases that are used with system-specific meanings;
- using episodes to express dependency between scenarios; and
- using syntactic similarity to measure similarity between scenarios, for searching, uncovering duplication, and estimating requirements coverage.

These approaches mutually reinforce each other and are amenable to automated support. We refer to this collection of approaches as Integrated Syntactic Analysis (ISA).

Throughout this chapter we employ several scenarios for the Meeting Scheduler problem as examples. The Meeting Scheduler is a standard problem in requirements engineering [AABM99]. The example Meeting Scheduler scenarios are shown in Figures 3.1 and 3.2; the first is "Schedule Meeting (no conflicts)" and the second is 'Schedule Meeting with Slow Responder (ordinary participant)." Each scenario includes a name, the purpose the scenario was developed for, author, the goal that is achieved by the scenario's events, and the sequence of events that achieves that goal.

The remainder of this chapter is organized as follows. We define terms that we will use in our discussion of scenario formalization. We present an unobtrusive formalization of scenarios, and assess some reasons why this degree of formality can be considered minimal. Glossaries are fundamental to our formalization and to its use, and we discuss their use and some of the issues that arise. We consider how the formalization may be used in episode management and for similarity measures. One possible use of similarity measures is in assessing requirements coverage of a group of scenarios, and we discuss some ways in which coverage is indicated by similarity. The chapter provides a brief discussion of SM^{a}RT, our software tool that supports ISA. Finally we show some ways in which these techniques address some of the challenges of using scenarios for specifying behavior and some of the problems of scenario management.

Name:	Schedule Meeting (no conflicts)	
Purpose:	Requirements elaboration	
Developer:	Colin Potts	
Goals:	*ACHIEVE* meeting scheduled	
Events:		
No.	**Actor**	**Action**
1.	*Initiator*	Request meeting of a specific type
2.	Scheduler	Add *default participants*
3.	*Initiator*	Determine participants
4.	*Initiator*	Identify active participants
5.	*Initiator*	Identify *initiator*'s boss as *important participant*
6.	*Initiator*	Send request for preference sets
7.	Scheduler	Send appropriate mail messages to participants
8.	Ordinary participant	Respond with *exclusion sets* and *preference sets*
9.	*Active participant*	Respond with *exclusion sets* and *preference sets* and equipment requirements
10.	Scheduler	Request required equipment
11.	*Important participant*	Respond with *exclusion sets* and *preference sets* and location preference
12.	Scheduler	Schedule meeting on the basis of responses, *policies*, and room availability
13.	Scheduler	Send confirmation message to all participants and *initiator*

(From [AABM99])

Figure 3.1: Meeting Scheduler scenario S_1 "Schedule Meeting (no conflicts)"

Name:	Schedule Meeting with Slow Responder (ordinary participant)	
Purpose:	Requirements elaboration	
Developer:	Annie Antón	
Goals:	*ACHIEVE* meeting scheduled	
Events:		
No.	**Actor**	**Action**
1.	*Initiator*	Request meeting of a specific type
2.	Scheduler	Add *default participants*
3.	*Initiator*	Determine participants
4.	*Initiator*	Identify *active participants*
5.	*Initiator*	Identify *initiator*'s boss as *important participant*
6.	*Initiator*	Send request for *preference sets*
7.	Scheduler	Send appropriate mail messages to participants
8.	*Active participant*	Respond with *exclusion sets* and *preference sets* and equipment requirements
9.	Scheduler	Request required equipment
10.	*Important participant*	Respond with *exclusion sets* and *preference sets* and location preference
11.	Scheduler	Recognizes that timeout has expired and reminds late participant
12.	Scheduler	Recognizes that *drop-dead date* has passed
13.	Scheduler	Schedule meeting on the basis of responses, *policies*, and room availability
14.	Scheduler	Send confirmation message to all participants and *initiator*

(From [AABM99])

Figure 3.2: Meeting Scheduler scenario S_2 "Schedule Meeting with Slow Responder (ordinary participant)"

3.1 Terminology

- A ***scenario*** is a sequence of events, plus possibly some associated attributes such as goals, requirements, viewpoint, author, and pre- and postconditions [AABM99]. Common scenario representations can be abstracted to this form [AP98a].
- An ***event*** consists of an actor-action pair. An actor may be a specific person, component, or system, or may be an unbound role or parameter that can be filled by any of several specific actors.
- A ***subsequence*** is a sequence of one or more events that forms all or part of a scenario's complete event sequence.
- An ***episode*** is a named subsequence that is intended to be shared among several scenarios.

3.2 Scenario structure

Structure refers to the way the constituent parts are put together to make a whole, and the mutual relations of those parts in determining the character of the whole. ***Formality*** refers to the use of outwardly visible structure to express an essential inward meaning; and in such a way that operations carried out on the visible structure produce results that preserve the correspondence with inward meaning. We use a formal structure for scenarios so we can answer questions about scenarios by examining their structure.

To answer any interesting question about prose scenarios requires careful reading, hard thought, insight, and skill. The text form of prose scenarios gives wide scope to the form of their expression, so much so that examination of the form itself is not informative. A formal structure for scenarios, in contrast to prose, allows interesting questions to be answered by an algorithm. Our goal here is to enjoy the primary benefit of formalization: to obtain useful results from working with the form in which scenarios are expressed, without having to consider what the scenario means. An analog is algebra, in which powerful results are obtained by manipulating the symbols in which formulas are expressed without needing

to know that v represents velocity, for example. Many more such formal systems could be named (such as symbolic logic, differential calculus, set theory, and the lambda calculus).

One aspect of scenarios that is immediately visible is that they are composed of several distinct types of information (events, goals, conditions, etc.) in the form of attributes and attribute values, and thus scenarios should be considered as entities composed of parts. Entities for which this is the case are the subject of mereology, the field of philosophy concerned with formal theories of "part" and "whole" and related concepts [Sim87]. The basic definitions and distinctions used by researchers to classify this area of study are useful to us in laying out the groundwork for our scenario structure. We assert that a scenario has an identity distinct from its parts, in several senses. It is possible for two scenarios to consist of exactly the same parts, and yet not be considered the same scenario; thus, scenarios are not strictly *extensional*. A scenario can evolve over time by changing its parts (for example by getting a new goal or changing to a different precondition) and still retain its identity and be the same scenario it was originally; thus, scenarios are *temporally variable*. Even for a scenario that does not evolve over time, if its parts did change, this would have no effect on the identity of the scenario; thus, scenarios are *modally variable*. We make use of several relationships among scenarios, including the relationship of *identity*some of whose properties have been implied above; two scenarios can be identical only if they are in fact the same scenario. We will consider various kinds and degrees of *equivalence* between scenarios, where two scenarios are equivalent with respect to some specific operator if applying the operator to each of the two scenarios produces the same result. We follow the common practice of defining *equality* as the "smallest" equivalence, so that equal scenarios are equivalent under all operations.

In order to choose an appropriate formalism for scenarios, we consider what specific operations and relationships are fundamental to scenarios; what operations and relationships are desirable on scenarios; and what sorts of scenario structure can be added naturally, inconspicuously, and advantageously. The most fundamental formal question we can ask about scenarios is whether two of them are *indistinguishable*, in the sense that their representations are indistinguishable. For prose scenarios, we would ask whether they are textually identical; that is, whether the character sequences that express two scenarios are

the same. If two entities are indistinguishable in this sense, they are equal and equivalent in every possible sense except possibly that of being different copies, or having separate identities despite being otherwise indistinguishable. It is relatively easy to determine if two scenarios are textually indistinguishable, and we can say with confidence that two scenarios that are equivalent in this sense are also equivalent in every other sense, from the relatively trivial ("are they equally easy to understand?") to the much more challenging ("do they describe behavior that has the same effect?").

Many pairs of scenarios that are textually distinct are in fact equal or equivalent in important senses. One can think of many trivial changes to the prose in which a scenario is expressed that have no effect on the behavior it describes: changing the letters to uppercase, adding extra whitespace, changing the sequence in which the goals of the scenario are listed, or in which the clauses of the precondition appear, and so forth. Any of these changes produces another scenario that is distinguishable from the original, but otherwise equivalent in all important senses. A formalization of scenarios that was not subject to such petty changes, or with which some classes of changes could be shown to result in equivalent scenarios, would be a valuable one. A formalization for which equality of representation implies equivalence in important senses would be highly desirable, because then answering a simple question (equality of form) also gives the answer to questions that are more exciting. Even a formalization for which equivalence in important senses could be related to equality of representation would be a vast improvement over prose.

There are several ways to approach the issue of distinct but equivalent scenarios. One is to devise rewrite rules and operators that direct the transformation of a scenario from one representation to another one that is equivalent. Algebra, first-order logic, and the lambda calculus are examples of formal systems for which such rules and operators are used. In algebra, for example, addition is commutative so that any term involving addition may be rewritten with the subterms interchanged, producing an equivalent term; in first-order logic, conjunction is idempotent so that any term conjoined with itself may be rewritten with the conjunction replaced by the term; and in the lambda calculus, the α-conversion operator, in which bound variables in a term are renamed consistently, defines an equivalence (α-equivalence) among terms that can be converted to each other by α-

conversion. Each of these systems is formally defined and is written in a formal language (see Table 3.1 for examples). For such rules and operators to be practical, the forms that they act upon must be much more restricted than that of prose scenarios, with formally defined semantics. This approach may not be attractive for scenarios due to the high level of abstraction that is required.

A second way to deal with the issue of equivalent but differently-expressed scenarios is by allowing fewer ways to express scenarios. Then equivalent scenarios are more likely to be expressed the same way, using the same components combined in the same structure. We can do this by restricting what components are permitted, where the boundaries between components can occur, and what structures can combine the components into scenarios. This chapter presents a formalization of scenarios based on this approach.

The structure we have chosen for scenarios is as *a set of attribute-value pairs,* where the attributes are event, goal, requirement, viewpoint, author, precondition, postcondition, and anything else that is useful for a particular scenario. For most purposes we consider scenario events to be ordered in a sequence; but for some purposes (such as calculating similarity) we ignore the sequence. The event sequence is recursively organized, consisting of simple events each consisting of an actor-action pair, and compound events with more complex structure. A compound event can be: an iteration of an event sequence; an alternation between two or more event sequences; or an episode, a named event sequence. This definition of event sequence gives our scenarios considerably more expressive power than is usual, and still allows linear sequences of simple events as is common in other definitions of scenarios. The detailed structure of our formalization is presented in Table 3.2. Examples of real scenarios in this form are presented in the case studies chapter in Table 5.3, page 98. We believe our event sequence representation is an improvement over that generally

Algebra:	$\sum_{k=0}^{n} \frac{n!}{k!(n-k)!} a^{n-k} b^k$
First-order logic:	$[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$
λ-calculus:	$\lambda G.(\lambda g.G(gg))(\lambda g.G(gg))$

Table 3.1: Examples of expressions in several formal languages

scenario	::=	*event-sequence av-list*
av-list	::=	*empty*
	|	*av-list attribute-value*
attribute-value	::=	(*attribute*, *value*)
attribute	::=	*name*
event-sequence	::=	*empty*
	|	*event-sequence event*
	|	*event-sequence iteration*
	|	*event-sequence alternation*
	|	*event-sequence episode*
event	::=	(*actor*, *action*)
iteration	::=	(`iteration`, *event-sequence*)
alternation	::=	(`alternation`, *event-sequences*)
event-sequences	::=	(`sequence` *event-sequence*)
	|	*event-sequences* (sequence *event-sequence*)
episode	::=	(`episode`, *name*)

Terminal symbols:
*attribute*s are named sets of values; examples of attribute names are "goal," "requirement," "viewpoint," "author," "precondition," and "postcondition".
*episode*s are named *event-sequence*s.
*name*s and *value*s are strings.

Table 3.2: Our scenario formalization

provided by scenarios (a single linear sequence) and use cases (a main linear sequence with variant linear sequences, augmented by the unnecessarily complex and ineffective "uses," "includes," and "extends" of other use cases [Sim99].

This structure is useful because it allows us to compare two scenarios by comparing their structural form and the components that comprise it (see next section on Glossaries). If the forms are the same, and the structurally-corresponding components are the same, then the scenarios are equal and thus equivalent (because equality is the strongest equivalence). The use of this structure and of glossaries directs the expression of scenarios so that equivalent scenarios are much more likely to be expressed equally. This allows an exact but exacting comparison, requiring skill, insight, and hard thought, with an approximate but easy one that produces no false positives and few false negatives.

We present the operations and relationships that we want for scenarios in terms of questions we ask about scenarios. The most fundamental question is: *are two scenarios equal?* This is of particular interest since, as described above, a single scenario may be expressed in many distinct ways. A natural and useful extension of equality is similarity: *how close to equal are two scenarios?* Recognizing that scenarios describe a sequence of events transpiring over time, we are also interested in equality between parts of scenarios: *do two scenarios contain subsequences of events that are equal or similar?* Finally we extend our consideration to scenarios that are not necessarily equal but may be considered interchangeable in specific situations or under specific operators: *are two scenarios equivalent?* A particularly interesting equivalence is the relationship of operational equivalence: *are the effects of the events described by two scenarios the same?* We believe that this relationship is an example of one that cannot practically be determined from the events, and our formalization of scenarios explicitly does not attempt to support it. Instead, we compare scenario attributes such as goals or postconditions that can be considered as summaries of the effects of a scenario's events [Ant97, RGK99].

A scenario's attribute values, actors, and actions are all defined in glossaries, as discussed in the next section.

3.3 Glossaries

Glossaries are used to specify the set of values each attribute can have, and to define each value. Each attribute is given its own glossary of values (to distinguish values for different attributes that may coincidentally share the same name). This use of glossaries simplifies the work of determining if two attribute values are the same: if the values are given by the same glossary entry, they are the same, and otherwise they are distinct. At the same time, the glossaries provide a connection back to the prose form of the scenarios, so that the formal representation of a scenario may be translated back into a recognizable prose representation of the same scenario.

Example attribute glossaries for Meeting Scheduler scenarios S_1 and S_2 are shown in Table 3.3. Every attribute value that appears in the scenarios is present in the corresponding attribute glossary. Note that events are pairs of values, one actor and one action. There is no scenario name glossary, because we classify scenario names as the expression of a scenario's identity, rather than as values of a "Scenario Name" attribute.

We use ordinary glossaries as well, that is, glossaries giving definitions of terms. One possible term glossary for the two Meeting Scheduler scenarios appears in Table 3.4.

It is clear that the use of glossaries also has some positive effects that are less direct. A glossary encourages an author to move the work of clearly expressing a concept out of the main text, and into the glossary. This is a sort of prose reuse: the term need only be made clear once, in one location in the glossary, rather than at each use of the definition. A second sort is reuse of terms, where the same term is reused as appropriate, instead of one or more new terms with the same or nearly the same definition. Both kinds of reuse result in efficiencies for the writer and for the reader. This *term coagulation* results in a specification that uses a smaller number of more-distinct concepts, rather than a larger set of concepts that are less clearly distinguished from each other. There is perhaps some minimum number of terms needed in a particular case, but in practice it is much more likely that there will be an unnecessarily large number of poorly-distinguished terms.

Glossaries have benefits for specification and communication in general, and these apply to formalized scenarios as well. Glossaries encourage having definitions for terms,

"Purpose" glossary: requirements elaboration
"Developer" glossary: Annie Antón Colin Potts
"Goal" glossary: ACHIEVE meeting scheduled
"Actor" glossary: Active participant Important participant Initiator Ordinary participant Scheduler
"Action" glossary: Add default participants Determine participants Identify active participants Identify initiator's boss as important participant Recognizes that drop-dead date has passed Recognizes that timeout has expired and reminds late participant Request meeting of a specific type Request required equipment Respond with exclusion and preference sets and equipment requirements Respond with exclusion and preference sets and location preference Respond with exclusion and preference sets Schedule meeting on the basis of responses, policies, and room availability Send appropriate mail messages to participants Send confirmation message to all participants and meeting initiator Send request for preferences
"Event" glossary: (Active participant, Respond with exclusion and preference sets and equipment requirements) (Important participant, Respond with exclusion and preference sets and location preference) (Initiator, Determine participants) (Initiator, Identify active participants) (Initiator, Identify initiator's boss as important participant) (Initiator, Request meeting of a specific type) (Initiator, Send request for preferences) (Ordinary participant, Respond with exclusion and preference sets) (Scheduler, Add default participants) (Scheduler, Recognizes that drop-dead date has passed) (Scheduler, Recognizes that timeout has expired and reminds late participant) (Scheduler, Request required equipment) (Scheduler, "Schedule meeting on the basis of responses, policies, and room availability") (Scheduler, Send appropriate mail messages to participants) (Scheduler, Send confirmation message to all participants and meeting initiator)

Table 3.3: Meeting Scheduler attribute glossaries

Term	*Definition*
ACHIEVE	Verb for goals that express ensuring a condition that had not been true.
active participant	A participant who can demand that the meeting room have specific equipment.
default participant	Each meeting type may have default participants who are always invited.
drop-dead date	The latest date to which the scheduling of the meeting can be delayed.
exclusion set	The set of times that a specific participant cannot attend the meeting.
important participant	A participant whose preferred location is considered when scheduling the meeting.
initiator	The participant who requested a specific meeting.
policies	Unspecified rules and heuristics that are followed for all meetings.
preference set	The set of times that a specific particpant would prefer to attend the meeting.
required equipment	The equipment specified by the active participants.

Table 3.4: Meeting Scheduler term glossary

and using terms that have definitions. They enable a reader to read confidently, with the knowledge that he or she can look up any unfamiliar term. Glossaries reduce the semantic demands on a human reader, by expressing information with a limited number of well-defined term. This benefit is even more marked if a specification is to be read by a program.

3.4 Episode management

Episodes, sequences of events intentionally shared among two or more scenarios, play a significant role in scenario management. An episode indicates an intentional relationship between the scenarios that share it. Frequently an episode also indicates a relationship between the scenarios sharing the episode and scenarios that can occur earlier and prepare what is needed by the events in the episode, or that can occur later and depend on the effects of those events.

The identification of episodes in a collection of 50 or more scenarios is tedious and error-prone if done manually. The scenario formalization described in Section 3.2 supports the automated identification of possible episodes by making it easier to identify equal events. An event in our formalization consists of an actor and an action. The number of actors for a system is generally much smaller than the number of events, and these actors are relatively well-defined. The number of actions is larger, and in some cases there are nearly as many actions as events. There may be several slightly different glossary entries for the same action, simply because there are so many actions that thay may be hard to manage. This leads us toward a strategy giving priority to actors when determining possible episodes. Currently the intervention of an analyst is necessary to locate equal events, and thus episodes.

We envision several ways to support the identification of equal actions, using the support of a tool. The Scenario Management and Requirements Tool (SMaRT), currently under development at North Carolina State University, provide such support for collections of scenarios. In each of the cases listed below, the tool may present pairs of potentially equal actions, possibly in conjunction with the corresponding events. An analyst using the tool determines whether the actions are equal, and the tool records the decision for future use.

1. *The tool presents only the pairs of actions that the analyst asks for.* The analyst chooses which pairs to examine. For a thorough comparison, many pairs must be examined.

2. *The tool presents potentially equal pairs of actions based on similarity of their glossary entries.* The tool examines the textual definitions of actions in the glossary and presents pairs of actions whose definitions are textually similar. The analyst may have to examine many pairs, but fewer than in the first case.

3. *The tool presents only the potentially equal actions that, if equal, would make two events equal.* The analyst only need examine the pairs that might result in identification of an episode. Fewer pairs need be examined, and many actions might not need to be examined at all.

4. *The tool presents only potentially equal actions that, if equal, would make two subsequences of n or more events equal.* In addition to the conditions imposed in the three previous cases, this case further restricts the presented pairs of actions to only those that appear in two shared subsequences of length $n \geq 2$ that are potentially equal. n may be adjusted to reduce the number of presented pairs as desired.

Once episodes are identified and recorded, the tool supports the maintenance of the scenario relationships they represent in several ways:

1. Highlighting of episodes when a scenario is presented.

2. Links from each scenario to others sharing the same episode.

3. Two ways of presenting a scenario that shares an episode: as an independent sequence of events, or as a variant on another scenario sharing that episode.

4. Warnings and alternate presentations and choices when an event that is part of an episode is edited; change all scenarios containing the episode? break the relationship into two groups, and only change the scenarios in one group?

Table 3.5 shows the potential episodes in Meeting Scheduler scenarios S_1 and S_2. The event subsequences common to them are listed, together with each event's number in S_1 and S_2 respectively.

The value of this support increases markedly as the number of scenarios increases and the number of analysts working on them increases, due to the increased difficulty of identifying, tracking, and maintaining these relationships in that situation.

3.5 Similarity

In a collection of scenarios, we are interested in the similarity between pairs of scenarios for several reasons. Similarity can indicate possible duplication among scenarios. It can help the search for a particular scenario. When one scenario needs attention, similar scenarios may also need attention. We discuss these reasons below, and then consider

Event in S_1	Event in S_2	Actor	Action
1.	1.	*Initiator*	Request meeting of a specific type
2.	2.	Scheduler	Add *default participants*
3.	3.	*Initiator*	Determine participants
4.	4.	*Initiator*	Identify active participants
5.	5.	*Initiator*	Identify *initiator*'s boss as *important participant*
6.	6.	*Initiator*	Send request for preference sets
7.	7.	Scheduler	Send appropriate mail messages to participants
8.	–	Ordinary participant	Respond with *exclusion sets* and *preference sets*
9.	8.	*Active participant*	Respond with *exclusion sets* and *preference sets* and equipment requirements
10.	9.	Scheduler	Request required equipment
11.	10.	*Important participant*	Respond with *exclusion sets* and *preference sets* and location preference
–	11.	Scheduler	Recognizes that timeout has expired and reminds late participant
–	12.	Scheduler	Recognizes that *drop-dead date* has passed
12.	13.	Scheduler	Schedule meeting on the basis of responses, *policies*, and room availability
13.	14.	Scheduler	Send confirmation message to all participants and *initiator*

Table 3.5: Three possible Meeting Scheduler episodes in S_1 and S_2

one similarity measure for scenarios that has been formalized as previously described in Section 3.2.

Duplication can occur when an analyst writing a scenario does not realize that the collection already contains a scenario describing the same behavior; or when during scenario evolution two scenarios change so that although they initially described two distinct behaviors, they converge to describe a single behavior. In a large collection, such duplication can go unnoticed if it must be detected manually. An automated similarity measure can focus the attention of an analyst on those scenarios that are most likely to be duplicates, so that only a few pairs of scenarios need be compared to eliminate duplication. Exact duplication of a scenario in a collection introduces the risk that as the scenario collection evolves, one of the duplicates will be updated and the other will not, so that the collection becomes ambiguous and internally inconsistent; near-duplication of a scenario in a collection introduces ambiguity and inconsistency directly.

A scenario *similarity measure* is a function that produces a number expressing the degree of similarity between two scenarios. Although no research has been published specifically on similarity measures between scenarios, there has been research on measuring similarity between various kinds of entities (see Chapter 2.4) [BGMM99, NoDRC+01]. Here, we discuss a similarity measure that is specifically tailored for scenarios formalized as described in Section 3.2. This similarity measure was described by Alspaugh *et al.* [AABM99]. To measure similarity, we consider each scenario as a set of attribute values. We assign the attributes embedded in episodes and events to the scenario in which they appear, so that all attributes of a scenario are examined.

A similarity measure, in combination with the formalization for scenarios discussed above, can help in searching for particular scenarios by providing an approach by which the scenarios in a collection can be examined mechanically to find a particular one. This approach is: list attributes that the desired scenario would have; use a program to compare these attributes to the attributes of the scenarios in the collection; manually examine the scenarios that are most similar in terms of these attributes, to find whether the desired scenario is present and if so which one it is.

A similarity measure also provides one means of directing an analyst's attention

to the scenarios he or she should consider. In many cases, when an analyst is considering or editing a particular scenario, the same consideration or changes are likely to be needed for other scenarios related to it. The scenarios which are most similar to the one in question are likely to contain at least some of the scenarios that need attention. We discuss other ways of identifying such scenarios in Chapter 4.4.

Our similarity measure considers only the syntactic representation of scenarios. It examines whether two attribute values are identical, but not any other relation between them (such as ordering or subsumption) that might arise from a semantic structure. This syntactic focus has several advantages: it is easier to understand, the results of the measure map in an intuitive way to the scenarios, and the extra labor of constructing a semantic structure is not needed.

This similarity measure has several important properties. First, its values are normalized to lie between 0 and 1, with 0 indicating complete dissimilarity and 1 indicating equality. Second, it is customizable for different applications, including scenario search mechanisms and grouping strategies. Third, it can be computed using a computer algorithm, so that similarity can be measured without costly and time-consuming human evaluations. Fourth, it uses attribute glossaries, described above, to simplify scenario comparisons.

For scenario comparison, we select scenario attributes with common values stored in a glossary. Then, "equivalent" attributes are those with the same name in the glossary. In a more complicated strategy, an algorithm could classify equivalent attribute values according to an external semantic structure. Here, we limit the attributes considered to those that require the use of glossaries.

The similarity measure compares scenarios as sets (i.e. unordered lists without duplication) of attribute values, and examines these sets for overlap. Other classes of similarity measures account for sequence of elements such as events [Kru83]; our measure does not, in order to produce a measure that is more easily understood. Thus, for two scenarios S_1 and S_2, each has an associated list of attribute values. Consider the lists of attributes for two scenarios S_1 and S_2:

$$
\begin{aligned}
S_1 &= \{Actor_3, Actor_5, Actor_6, Goal_2, Purpose_2, \\
&\quad Viewpoint_1, ConcretenessLevel_0\}; \\
S_2 &= \{Actor_3, Actor_4, Actor_5, Actor_7, Goal_2, \\
&\quad Purpose_1, Viewpoint_1, ConcretenessLevel_0\}.
\end{aligned}
$$

We define the *similarity measure* $S(S_1, S_2)$, the similarity between scenarios S_1 and S_2, as the number of common attribute values in each attribute list, divided by the sum of the sizes of each attribute list, (see [Tve77] for other ratio models):

$$S(S_1, S_2) = \frac{2 \cdot |S_1 \cap S_2|}{(|S_1| + |S_2|)}$$

The factor of two normalizes the result so that identical scenarios have a similarity of 1. Therefore, the similarity between scenarios S_1 and S_2 is $S(S_1, S_2) = 10/15 = 0.667$. The similarity measure can be seen as a percentage of overlap, where the minimal similarity value of 0 indicates no similarity, and the maximal value of 1 indicates complete overlap in the attribute lists.

The similarity of Meeting Schedule scenarios S_1 and S_2 (Figures 3.1 and 3.2) is demonstrated in Table 3.6. We list all of the attribute-value pairs in the two scenarios, except for the developer's name which we consider unhelpful for similarity; the name is considered to represent the identity of the scenario and is not an attribute. There are a total of 22 attribute-value pairs in the two scenarios taken together. Each scenario contains 20 of them, and there are 16 attribute-value pairs that occur in both. The resulting similarity is

$$\frac{2 \cdot 16}{20 + 20} = 0.8$$

indicating that the scenarios are relatively similar.

In this simple scheme, each attribute value can be considered to have a "weight" of 1. A family of similarity measures arises when we allow different weighting schemes, where each attribute or attribute value is assigned a weight between 0 and 1. Then, when the similarity measure is taken, each attribute value in the measure is multiplied by its weight.

Present in S_1	Present in S_2	Atribute	Value
1	1	Action	Add *default participants*
1	1	Action	Determine participants
1	1	Action	Identify *initiator*'s boss as *important participant*
1	1	Action	Identify active participants
0	1	Action	Recognizes that *drop-dead date* has passed
0	1	Action	Recognizes that timeout has expired and reminds late participant
1	1	Action	Request meeting of a specific type
1	1	Action	Request required equipment
1	1	Action	Respond with *exclusion sets* and *preference sets* and equipment requirements
1	1	Action	Respond with *exclusion sets* and *preference sets* and location preference
1	0	Action	Respond with *exclusion sets* and *preference sets*
1	1	Action	Schedule meeting on the basis of responses, *policies*, and room availability
1	1	Action	Send appropriate mail messages to participants
1	1	Action	Send confirmation message to all participants and *initiator*
1	1	Action	Send request for preference sets
1	1	Actor	*Active participant*
1	0	Actor	Ordinary participant
1	1	Actor	*Important participant*
1	1	Actor	*Initiator*
1	1	Actor	Scheduler
1	1	Goal	*ACHIEVE* meeting scheduled
1	1	Purpose	Requirements elaboration
20	20	Number of attribute:value pairs in each scenario	
16		Number common to both scenarios	
0.8		Similarity $\frac{2 \cdot 16}{20+20}$	

Table 3.6: Similarity between Meeting Scheduler scenarios S_1 and S_2

One weighting function might, for example, assign a weight of 1 to each actor, and a weight of 0 to all other attributes. With this weighting, scenarios S_1 and S_2 would have a similarity measure of 2/7, about 0.289; by this measure, the two scenarios are much less similar. In this way, the weighted similarity measure emphasizes similarity in particular attributes (by assigning them high weights) and ignores difference in others (by assigning them weights of zero). This can be used for grouping similar scenarios based on particular attribute values, or for scenario searches. The weighted similarity measure can be particularly important for episode searching and matching.

Mathematically, we can write the weighted extension of S as SW, the weighted similarity measure between two scenarios, where a denotes an attribute, and $wt(a)$ denotes the weight assigned to attribute a. Note that, to avoid division by zero, we define the similarity between two scenarios to be 0 if all their attribute values have zero weights:

$$SW(S_1, S_2) = \frac{\sum_{a \in S_1 \cap S_2} 2 \cdot wt(a)}{\sum_{a \in S_1} wt(a) + \sum_{a \in S_2} wt(a)}$$

We also extend these measures to groups of any number of scenarios by taking the average of the similarity of each pair of scenarios. We extend S to S^*:

$$S^*(S_1, S_2, ..., S_n) = \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} S(S_i, S_j)}{n(n-1)/2}$$

For example, to calculate the similarity measure for S_1, S_2, and S_3, where S_1 and S_2 are given above, and

$$\begin{aligned} S_3 &= \{Actor_1, Actor_4, Actor_6, Actor_7, Goal_1, \\ &\quad Purpose_1, Viewpoint_3, ConcretenessLevel_0\}, \end{aligned}$$

we must compute

$$\begin{aligned} S^*(S_1, S_2, S_3) &= \frac{(S(S_1, S_2) + S(S_1, S_3) + S(S_2, S_3))}{3} \\ &= 0.478 \end{aligned}$$

Computed in this fashion, S^* gives an average of the similarities of each pair of scenarios in a group.

We may also extend SW to SW^* in a similar fashion:

$$SW^*(S_1, S_2, ..., S_n) = \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} SW(S_i, S_j)}{n(n-1)/2}$$

These two extensions, weighting and group comparisons, build a family of measures which are powerful tools for scenario management. The general measure can be used to search for and select scenarios with particular characteristics from a scenario database, quickly and automatically, as in [LR94]. It can also help identify accidental duplication in a large set of scenarios. Using the similarity measures, SMaRT will present similar scenarios, or episodes, to an analyst for evaluation and possible elimination. Because of their simplicity and flexibility, the similarity measures provide both quick automatic processing of data that would take hours by hand, and a measure that corresponds to our intuitions about scenario similarities.

3.6 Similarity and requirements coverage

It is difficult to determine if a collection of scenarios is complete in the sense of covering the required behaviors, yet coverage is an important desideratum of specifications [LW98]. Even for a relatively small collection of scenarios, it is virtually impossible to show that the collection is complete or to uncover all the specific areas in which it is incomplete. Thus, even an approximate indication of completeness would be valuable, especially if it indicated areas of incompleteness and thus provided process guidance for improving the completeness. Covering the required behaviors means more than ensuring that at least one scenario traces back to every requirement. The scenarios must also define all the behavior that is needed to implement that requirement.

A scenario similarity measure, such as the one presented in Section 3.5, may form the basis of an indication of completeness and requirements coverage. We describe below several ways this measure might show coverage. Validation of each of these ways requires data on the evolution of formalized scenarios during the course of software development projects, and no such data is available at this time; this validation is part of the future work in this area. Preliminary validation data is discussed in Section 5.6.

In describing the estimation of requirements coverage using similarity, we use a spatial metaphor in which similarity is distance, and similar behaviors are near each other. Scenarios that describe the same general area of behavior may be expected to share elements such as actors and actions. Thus an area of behavior that has been thoroughly covered is likely to result in a cluster of scenarios whose pairwise similarity is relatively high. Conversely, an area of behavior that has not been thoroughly covered is likely to be described by a scenario or scenarios that are relatively dissimilar to most of the scenarios in the collection. Thus the presence of scenarios that are dissimilar to all or nearly all other scenarios in a collection may be taken as a indication of an area of behavior that needs more coverage. An indicator of this would be a list of the scenarios in a collection sorted by the sum of the pairwise similarities of that scenario to all others in the collection; the scenarios whose sum is lowest describe behaviors in areas that are less likely to be completely covered. Thus this indicator, to the extent that it indicates lack of coverage, provides procedural guidance in improving coverage. Visual presentation of all this information in a manageable form is problematic. We propose as a candidate presentation a three-dimensional graph whose x-axis indicates each of the scenarios in the collection, whose y-axis covers the range of the similarity measure and is partitioned into bins for grouping similarity values, and whose z-axis is the number of scenarios whose pairwise similarity with the scenario at that interval of the x-axis falls into that bin of values.

A second indication of requirements coverage might be found in the change in distribution of pairwise similarities over time as the scenarios evolve and the development project proceeds. We hypothesize that this distribution may converge in a predictable manner to a final state with certain predictable characteristics. These characteristics are that each scenario is relatively quite similar to a small cluster of other scenarios in the collection, and relatively dissimilar to the vast majority of the scenarios; and that each new scenario is similar to those in some cluster, and dissimilar to all the others. Indication of requirements coverage in this way by a similarity measure would be useful even if it is an approximation, due to the importance of coverage and the difficulty of determining it.

3.7 SMaRT

Tool support is essential for ISA, especially for episodes and similarity. A research team at North Carolina State University has developed a software tool SMaRT (Scenario Management and Requirements Tool) which supports representing scenarios as attribute-value pairs, glossaries of attribute values, glossaries of terms, episode management, and syntactic similarity measures. SMaRT was used for the ISA portion of the Euronet case study discussed in Chapter 5.

SMaRT is currently implemented as a database on a server that is accessed over the Internet through a browser. The data presented by SMaRT is organized into projects. Each project can have one or more analysts allowed to edit or view the contents. For each project there is a set of scenarios, a set of episodes, and a glossary for each attribute that a scenario or episode can possess. Scenarios and episodes can be created and edited using attribute values in the glossaries. The attributes include: actors, actions, events, goals, obstacles, requirements, and conditions. New values can be created for an attribute; existing values can be edited (the changes appearing in every use of the value); and values that are not referenced can be deleted. A clickable list of every reference to the value can be generated automatically; clicking a reference takes the analyst to the scenario (episode, event) that uses the value. Event lists can include simple events, episodes, iteration of event sub-lists, and alternation between event sub-lists. Episode references in an event sequence can be expanded to show the episode's event sequence where it would occur, and the episode's event sequence can be edited in the context of a scenario referencing it. Clicking on episode and event names takes the analyst to the episode or event editor, where all the information about that episode or event is available.

A screen shot of the Scenario Editor is shown in Figure 3.3. The scenario being edited is Euronet S_2 "Retrieve Existing Quote".

Future work for SMaRT includes implementation of similarity measures, expanded cross-reference capability, "uses" hierarchies for episodes, and integration of scenario network support (see Chapter 4).

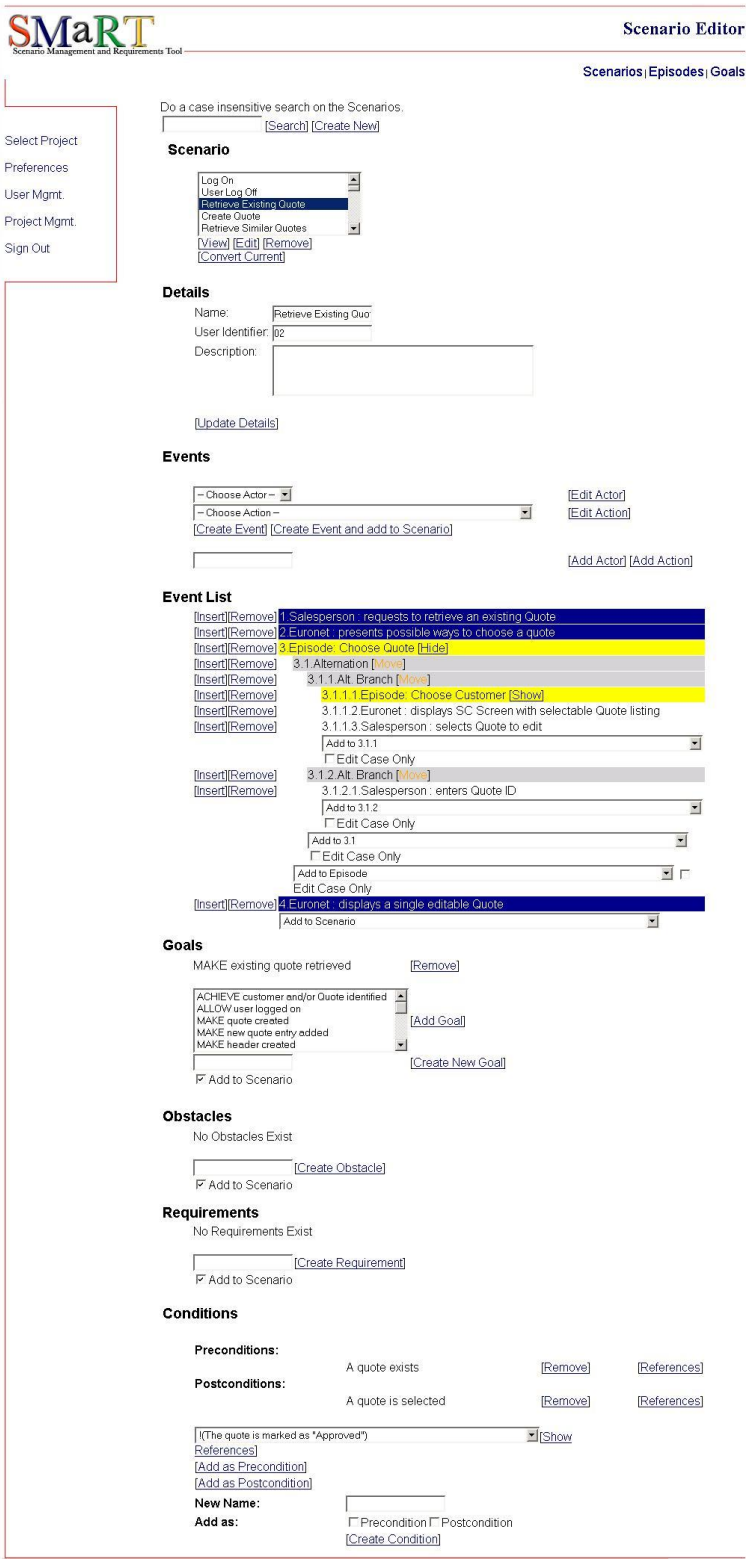


Figure 3.3: SMaRT Scenario Editor screen

3.8 Summary

In this chapter we have shown that it is possible to express scenarios in a form that provides a small degree of structure and formality, and that this degree of formality is arguably minimal. This formalization of scenarios provides a basis for addressing some of the challenges of specification using scenarios (Table 1.1, page 8), and typical scenario management problems (Table 1.2, page 9):

- Term glossaries can help reduce ambiguity in individual scenarios.
- Attribute value glossaries and a scenario representation based upon them provide a basis for automated analysis of scenarios and scenario collections.
- Episodes can express dependency relationships among scenarios, and support the maintenance of these relationships as the scenarios evolve.
- Similarity measures help eliminate duplicate and near-duplicate scenarios in a collection, support searching for a specific scenario, and may provide an indication of completeness and requirements coverage of a scenario collection.

A meta-model relating the concepts introduced in this chapter is given in Figure 3.4.

In the next chapter, we present the semantic approach to scenario formalization. This approach is much more powerful, and affects nearly all the challenges and problems we have listed in the use of scenarios.

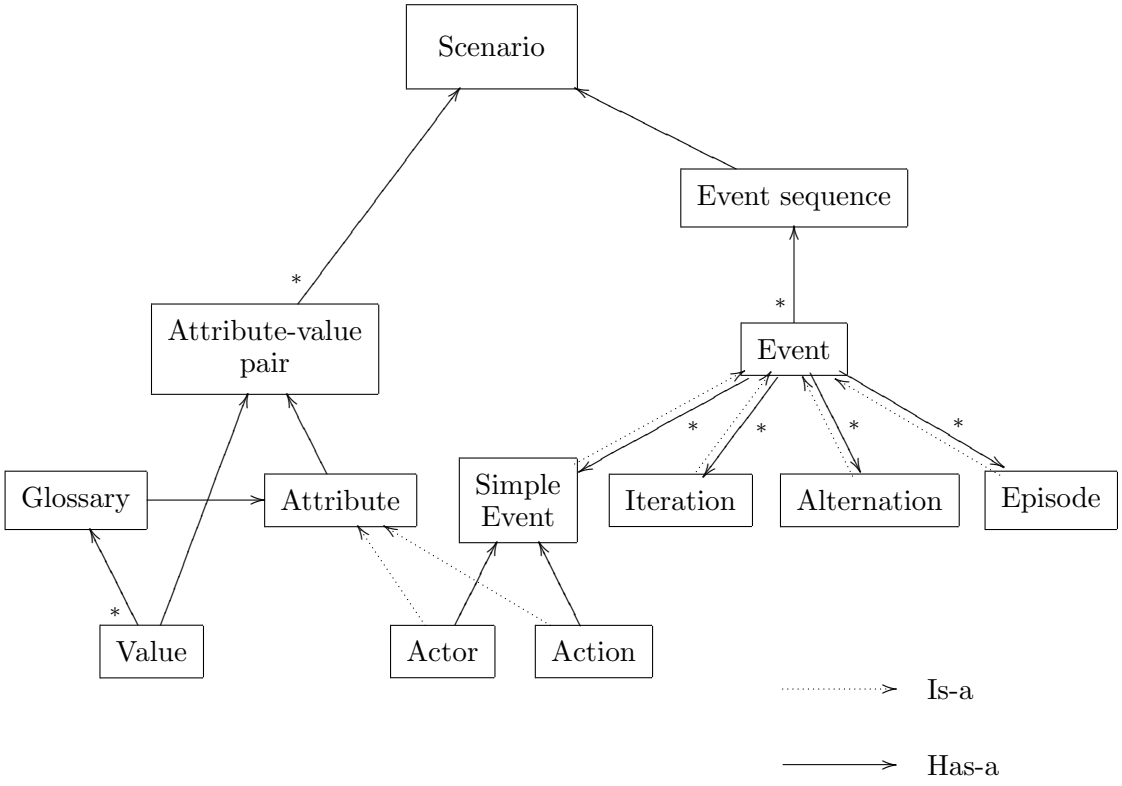


Figure 3.4: Meta-model for scenario formalization

Chapter 4

Scenario networks

> One starts by seeing the person who poses, but little by little all the possible sculptures of him intervene. *Alberto Giacometti*

A scenario is inherently partial; each scenario offers a single view of a single part of a system's behavior. We make a first step toward expressing a system's entire behavior by writing a collection of scenarios. It is as if we modelled a complex surface by making little patches, each one shaped like a small part of the surface. But our task is still not done; we must place each patch relative to every other patch. Only then can we say we have modelled the entire surface. To express the entire behavior of a system, we must place each scenario relative to every other scenario; not spatially, because we are not making a spatial model, but temporally. We must say when each scenario can occur relative to all the other scenarios, else our specification is incomplete. This has not been done with a specification that is a collection of scenarios. The rest of this chapter describes a way of taking that final step.

Consider how the scenarios in a collection can describe what happens during a single execution of a system. There may be a scenario that describes how the system starts; then another that describes what occurs next; and so on. At some point, if our system exhibits concurrency, a scenario would begin while another one is still in progress, and there may be a succession of scenarios following that one in a concurrent thread. The system runs along and the scenarios describe its interactions; more scenarios follow, and new concurrent threads of scenarios are spawned. Eventually a concurrent thread runs its course

and terminates; and finally, after a short time or a long time, all the concurrent threads terminate and the system stops. We call this pattern of concurrent sequences of scenarios a ***multipath***, to signify that the system follows paths through the scenarios that describe it, and that at any one time the system may be following multiple paths simultaneously. Figure 4.1 shows a multipath involving the scenarios in Table 4.1. Time runs from left to right, and each horizontal path indicates a sequence of scenarios that form a transaction, and are connected by the "intent" of some actor interacting with the system. The multipath begins at the initial scenario S_0; ramifies into seven concurrent branches, three beginning with S_2 and four with S_{30}; each of those branches eventually reaches either S_{29} or S_{39} and terminates; and finally the main trunk terminates at S_1 and ends the multipath.

Invisible in the multipath, but present in the system, are ***causal relationships*** between the scenarios that are not related to intent: one scenario may allow another (as the arrival of a message in S_{30} allows the hearing of that message in S_{12}), or prevent another (as listening to the last message in S_{12} prevents any further occurrences of S_{12} until another occurrence of S_{30}), or make another inevitable, possibly after a long time (as the startup of EMS in S_0 makes its eventual shutdown in S_1 inevitable), or place a limit on the span when another can occur (again, possibly a limit distant in time). These causal relationships embody behavior of the system that spans several scenarios, and thus is not completely expressed by any one scenario.

This multipath and the multipaths in Figure 4.2 describe a possible execution of part of a real system, the Enhanced Messaging System (EMS), which is a voice messaging system used by BellSouth Telecommunications to prototype new features [AA01]. Part of the EMS will be used throughout this chapter as an example. Eight scenarios from the

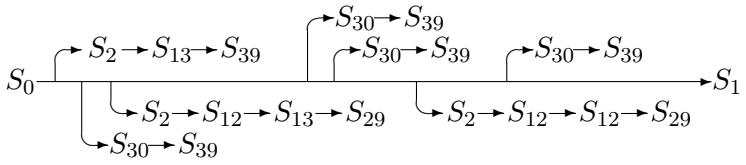


Figure 4.1: A multipath for the EMS-8 scenarios

full EMS specification are listed in Table 4.1; these scenarios are referred to as the EMS-8 example. Scenario S_{12}, "Subscriber listens to the next message", is shown in Table 4.2. The full EMS is described in Appendix A, and our EMS case study is presented in Chapter 5.

Describing a system in terms of its multipaths is reminiscent of the quotation by Giacometti that begins this chapter. We start by seeing just the individual scenarios for the system; but as we learn how the scenarios interact and fit together, little by little all the possible multipaths intervene, and we no longer think in terms of isolated scenarios. Inconveniently, "all the possible multipaths" is typically infinitely many. This chapter describes a conceptual structure, a *scenario network*, that expresses all the possible multipaths in a scenario collection, but in a finite form.

A scenario network expresses several kinds of relationships between scenarios. The causal and temporal relationships discussed above might be considered ***execution-time relationships***, because they are evident while the system is running. The other relationships might be considered ***specification-time relationships***, because they are evident while the scenarios are being specified. Some of these relationships are ***dependency relationships*** that express the fact that the events chosen for a scenario, the scope of its effects, its pre- or postcondition, or something else about they way it is expressed depends on how another scenario is expressed. We saw in Chapter 3 that episodes express a dependency relationship between the scenarios that use an episode. Another kind of dependency relationship arises from choosing a consistent scope for each of the scenarios in a collection, so that the scenarios do not overlap but neither do they miss any behavior in a gap between them. A second group of specification-time relationships are the equivalence relationships that are

S_0	EMS startup.
S_1	EMS shutdown.
S_2	Subscriber calls EMS, authenticates him/herself.
S_{12}	Subscriber listens to the next message.
S_{13}	Subscriber has no more messages to listen to.
S_{29}	Subscriber disconnects from EMS.
S_{30}	Caller calls a subscriber and leaves a message.
S_{39}	Caller disconnects from EMS.

Table 4.1: The EMS-8 scenarios

S_{12}. Subscriber listens to the next message.	
Requirements:	R3.2.1, R3.2.2.
Precondition:	$0 < s.rem$
Postcondition:	$s.rem' = s.rem - 1$
1. Subscriber s dials the *next message* command.	
2. EMS plays s's next message.	
3. EMS changes the state of that message to 'old' if it had been 'new'.	

Table 4.2: EMS scenario S_{12}

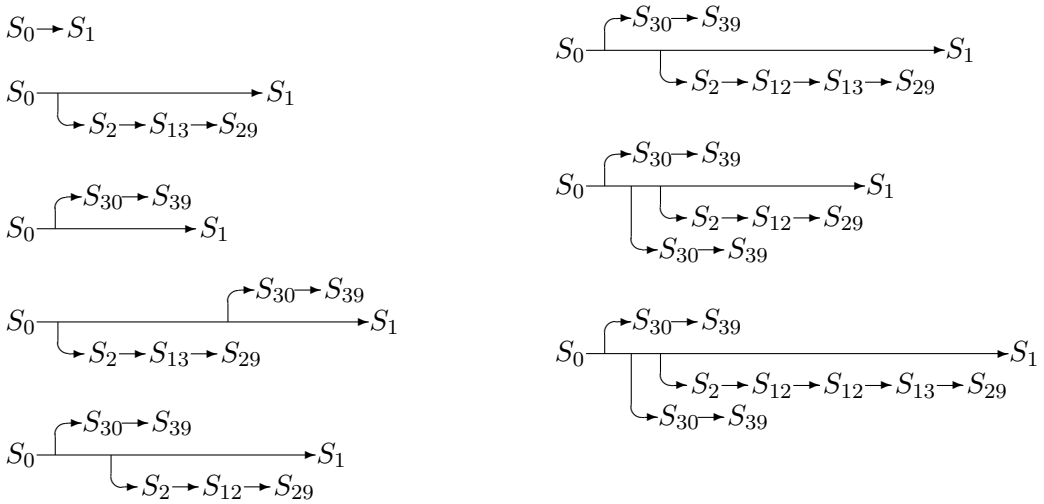


Figure 4.2: Eight possible multipaths for the EMS-8 scenarios

evident when a scenario network is examined. Two scenarios may be ***follow-equivalent*** if they can follow the same scenarios; or ***precede-equivalent*** if they can precede the same scenarios; or ***sequence-equivalent*** if they can both precede and follow the same scenarios. Later in this chapter we will define these more rigorously.

The main benefits of scenario networks may not be their use as specifications, but rather their use as a means of validating a collection of scenarios. The process of constructing a scenario network, and the analyses that are possible on a scenario network, together form a powerful technique for validating scenaro collections which we term Scenario Network Construction and Analysis (SNCA). Its components are

- The use of the scenario network construction process to provide guidance in identifying and (if desired) resolving inconsistencies between scenarios.
- The use of the scenario network construction process to provide guidance in identifying and filling in areas of incompleteness.
- Analysis of a scenario network to identify equivalence classes of scenarios, and use of these equivalence classes to identify and resolve further inconsistencies between scenarios.
- Examination of equivalence classes to identify potential dependency relationships among scenarios.
- Identification of scenario relationships arising from the network that should be preserved during evolution of the specification, and use of the network to track and preserve these relationships.

The remainder of this chapter is organized as follows. We define a number of terms used in discussing scenario networks. A gentle introduction to scenario networks follows, using the EMS-8 scenarios, and a discussion of how to construct a scenario network. We show how to invert a scenario network and what an inverted scenario network indicates. The scenario relationships that arise in a scenario network are defined and discussed. Next are presented some thoughts on the scope of applicability of scenario networks, and on what

kinds of system they are or are not useful. Automated support is needed for accurate work with scenario networks, and we discuss a tool that analyzes scenario networks and converts between their forms. We end by summarizing what this chapter has presented, and relating the benefits of scenario networks to the specification challenges and scenario management problems discussed in Chapter 1.

4.1 Terminology

We define the following key terms. Several of these terms have already been defined in earlier chapters, and their definitions are repeated here for the reader's convenience.

- A ***scenario*** is a sequence of events, plus possibly some associated attributes such as goals, requirements, viewpoint, author, and pre- and postconditions [AABM99].
- An ***event*** consists of an actor-action pair. An actor may be a specific person, component, or system, or may be an unbound role or parameter that can be filled by any of several specific actors.
- A ***scenario network*** is comprised of a group of scenarios and the temporal relationships between them that indicate the allowed scenario sequences and concurrency. At least one of the scenarios is distinguished as initial, and at least one other as terminal.
- A ***multipath*** is a possibly ramified path from scenario to scenario in a scenario network. In its simplest form (without concurrency), a multipath is simply a sequence of scenarios. Where concurrency is possible, a multipath can ramify into several concurrent sequences.
- A network's ***initial scenarios*** are those that can begin a multipath in the network.
- A network's ***terminal scenarios*** are those that can end a branch of a multipath in the network, and can appear nowhere but at the end of a branch.
- The ***precondition*** of a scenario is a logical expression that must be true for the scenario to begin. The precondition of scenario S_A is denoted $\mathrm{Pre}(S_A)$.

- The ***postcondition*** of a scenario is a logical expression that is guaranteed to be true after the scenario concludes. The postcondition of scenario S_A is denoted Post(S_A).

- A network's ***primitive terms*** are a set of variables of Boolean, integer, or other types, in which the pre- and postconditions of the network's scenarios are expressed.

- The ***follow set*** of a scenario is the set of scenarios in its network that can follow it in a sequence. The follow set of scenario S_A is denoted Follow(S_A).

- The ***precede set*** of a scenario is the set of scenarios in its network that can precede it in a sequence; that is, the set of scenarios whose follow sets contain it. The precede set of scenario S_A is denoted Precede(S_A).

- The ***ramification set*** of a scenario is the set of scenarios in its network that can begin a new concurrent sequence after it. The ramification set of scenario S_A is denoted Ramify(S_A).

- The ***trunk set*** of a scenario is the set of scenarios in its network whose ramification sets contain it (so called because it identifies where branches meet the trunk they ramify from). The trunk set of scenario S_A is denoted Trunk(S_A).

4.2 Scenario network example

A scenario network is a "rolled-up" form of a (possibly infinite) set of multipaths, in which repeated sequences of scenarios in multipaths are mapped onto a loop in the network. In this form each scenario appears exactly once, and all the possible next scenarios are connected to it. The next scenarios are separated into those that continue the same sequence of scenarios, and those that begin a new concurrent sequence. The scenario network for the EMS-8 scenarios (Table 4.1) is shown in diagram form in Figure 4.3. Each scenario is represented by a circle labelled with the scenario's identifier. Arrows connect each scenario to those that can follow it. Plain arrows denote continuation of sequence, and arrows beginning from a "T" end denote ramification into new concurrent sequences. Initial

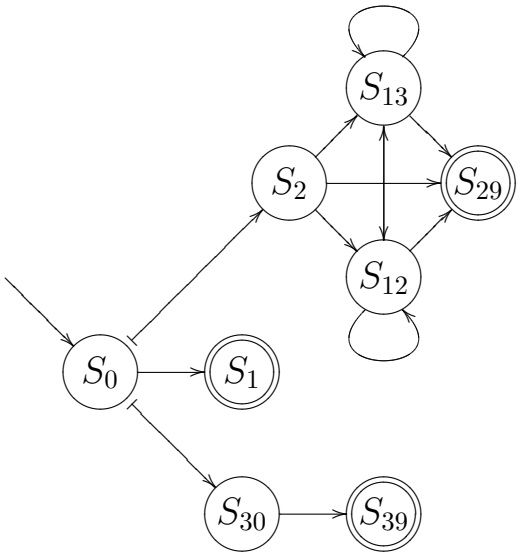


Figure 4.3: Diagram for EMS-8 scenario network

scenarios are indicated by an arrow from no scenario; terminal scenarios are indicated by doubled circles. A key for scenario network diagrams is given in Figure 4.4.

Each of the multipaths in Figures 4.1 and 4.2 can be traced on the scenario network diagram. Trace through the scenarios in the diagram by running a finger or sliding a dime along the arrows that connect them; as many fingers (or dimes) will be needed as the largest number of concurrent branches active at one time. Each finger traces the sequence of a separate branch of the multipath, and stops when it reaches a terminal scenario. For example, the multipath of Figure 4.1 begins at the initial scenario S_0 (place a finger on S_0 in the scenario network diagram); ramifies into seven concurrent branches, three beginning with S_2 and four with S_{30} (put another fingertip down on the diagram as each branch begins); each of those branches eventually reaches either S_{29} or S_{39} and terminates (take back the corresponding finger); and finally the main trunk terminates at S_1 and ends the multipath (all fingers released). One of the S_2 branches repeats S_{12} by following the self-loop. Any of the S_2 branches could have repeated S_{13} the same way, but none did. Perhaps the second S_2 branch could have taken the arrow from S_{13} to S_{12}, which is present in case

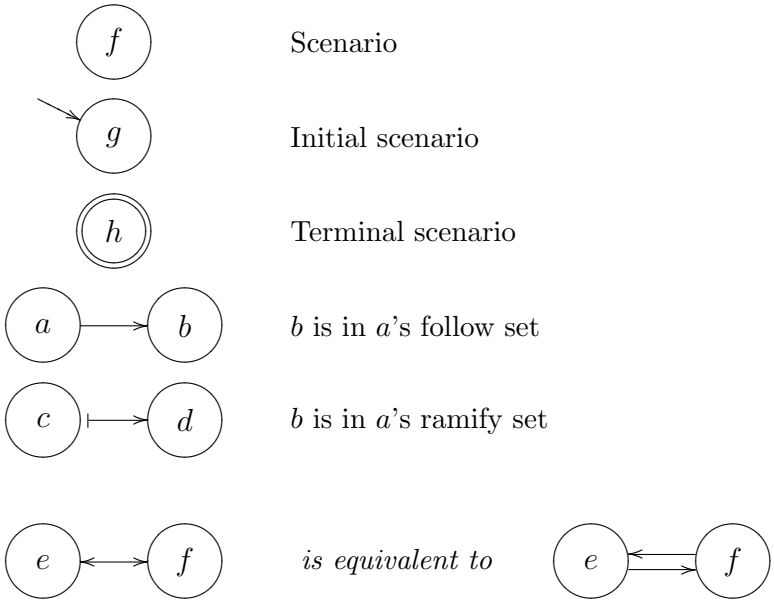


Figure 4.4: Key for the scenario network diagram notation

a caller leaves a message after the subscriber has found no messages but before he or she disconnects, in which case he or she can then listen to the new message. But this multipath does not give us enough information to know who the various messages were left for, so we cannot say.

The same information may be presented in tabular form, as shown in Table 4.3. In this form, we list the network's scenarios, identify those that are initial or terminal, and give each scenario's follow set and ramification set. Table 4.3 provides this information for the EMS-8 scenarios in Table 4.1.

A more complex scenario network appears in Figure 4.5, which shows the entire scenario network diagram for the full EMS. A shorthand notation for simplifying scenario network diagrams is used in this figure. A dotted circle indicates the Cartesian product of all arrows into it with all arrows out of it, and all arrows out with all arrows in. For example, at the dotted Cartesian product circle in the upper right, the arrow from S_{37} into the product stands for seven arrows: one from S_{37} to each of S_{31} through S_{35}, S_{38}, and S_{39}. A key for this notation is given in Figure 4.6.

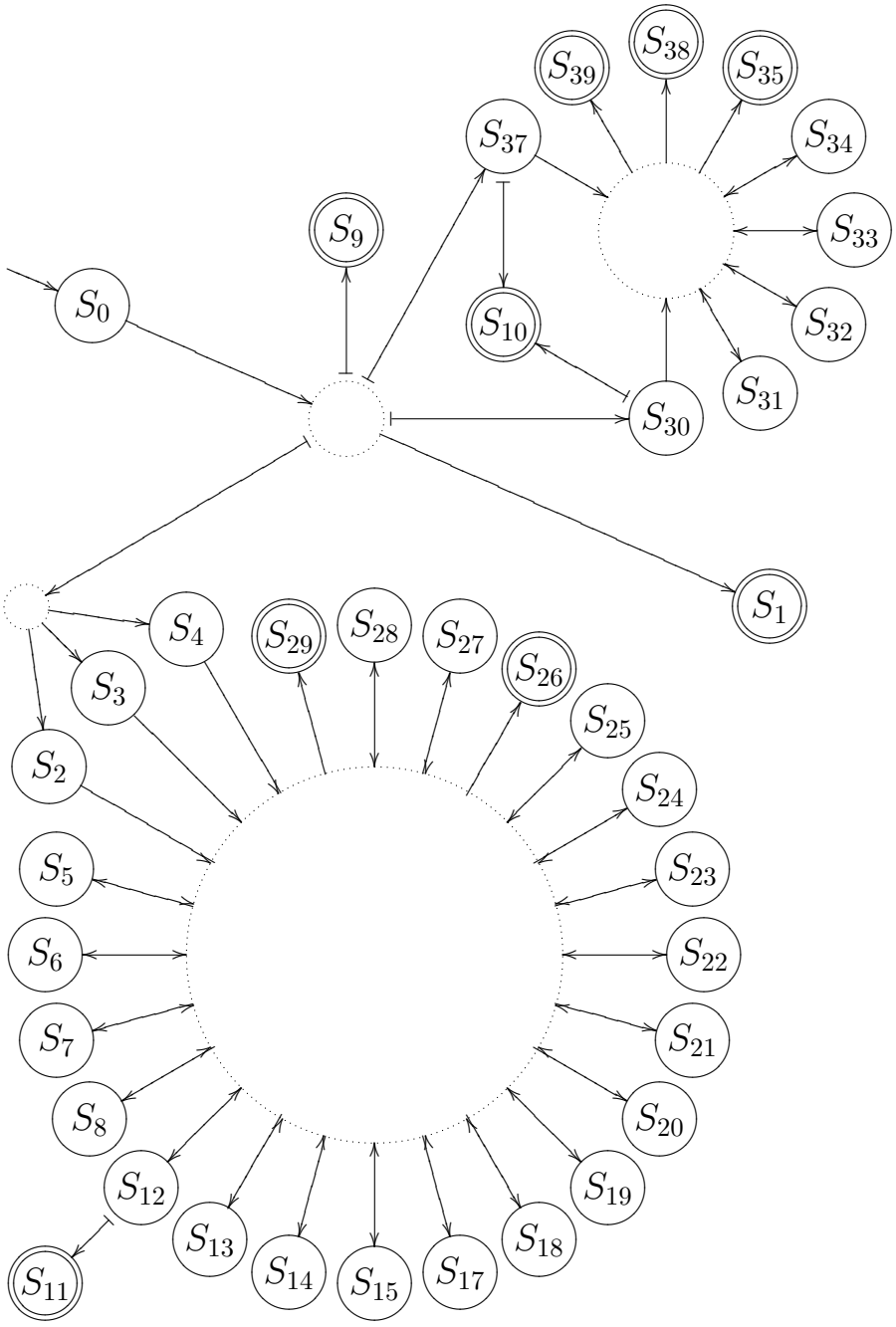


Figure 4.5: EMS scenario network diagram

Initial	S_0	
Terminal	S_1 S_{29} S_{39}	
Scenario	Follow set	Ramification set
S_0	S_1	S_2 S_{30}
S_1	$\emptyset$	$\emptyset$
S_2	S_{12} S_{13} S_{29}	$\emptyset$
S_{12}	S_{12} S_{13} S_{29}	$\emptyset$
S_{13}	S_{12} S_{13} S_{29}	$\emptyset$
S_{29}	$\emptyset$	$\emptyset$
S_{30}	S_{39}	$\emptyset$
S_{39}	$\emptyset$	$\emptyset$

Table 4.3: Tabular form of EMS-8 scenario network

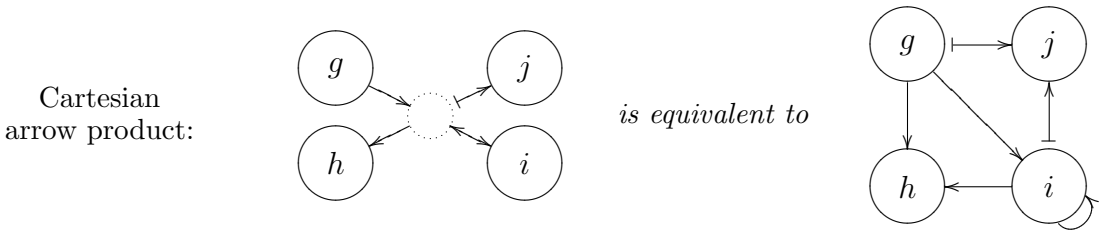
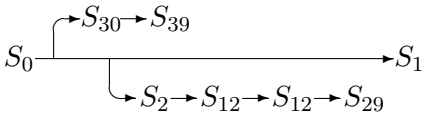


Figure 4.6: Key for Cartesian arrow product notation

The scenario network in Figure 4.3 supports all the allowed scenario multipaths listed in Table 4.2, and an infinite number of others. However, it also supports an infinite number of scenario multipaths that should not be allowed, such as



(caller left one message but subscriber listened to two)

Such undesired multipaths are ruled out by the scenarios' pre- and postconditions. As is common practice with anything that is given pre- and postconditions, a scenario's precondition is required to be true in order for the scenario to begin, and its postcondition is guaranteed to be true when the scenario ends. At any point in a multipath we can determine which scenarios can occur by comparing their preconditions with the postconditions fulfilled by scenarios already completed. The pre- and postconditions for the EMS-8 scenarios of Table 4.1 are listed in Table 4.4.

On the surface, scenario networks may appear similar to finite state machines, but there are important differences between them. Notably, each scenario's conditions refer to a state that can be arbitrarily complex, consisting of any combination of the network's primitive terms. Thus a scenario network is capable of much more complex transition sequences than a finite state machine. Also, a scenario network addresses concurrency in a manner different from the way a finite state machine does. When a nondeterministic finite state machine is in more than one state, the transitions from each of those states to the next occur together. Also there is not necessarily any notion of paths between states of a nondeterministic finite state machine, only of sets of states that the machine is in. In a scenario network, transitions between scenarios on different branches of a multipath can take place independently, and the state of the system is in no way represented by the set of scenarios that are occurring at that time. Finally, each transition between scenarios is part of some path or of some branch of the multipath, and only occurs as part of the sequence of scenarios on that branch. For these reasons scenario networks are distinct from finite state machines.

Equivalence relationships are discussed in Section 4.4, and the process of con-

Precondition		Postcondition
–	S_0	$(acc = tt) \wedge \forall s \in S : (s.tot' = 0)$
tt	S_1	$acc = ff$
$!s.ckg \wedge acc$	S_2	$(s.ckg' = tt) \wedge (s.rem' = s.tot)$
$0 < s.rem$	S_{12}	$s.rem' = s.rem - 1$
$0 = s.rem$	S_{13}	tt
tt	S_{29}	$s.ckg' = ff$
acc	S_{30}	$(s.tot' = s.tot + 1) \wedge (s.rem' = s.rem + 1)$
tt	S_{39}	tt

Term	Meaning
acc	True while EMS is accepting calls.
S	The finite set of subscribers.
s	A subscriber in S.
$s.ckg$	True while s is checking message
$s.tot$	The number of messages s has.
$s.rem$	The number of unheard messages s has.

Table 4.4: Pre- and postconditions of the EMS-8 scenarios

structing a scenario network is described in Section 4.5.

4.3 Scenario networks as system simulations

Scenario networks simulate a system by presenting multipaths of scenarios that correspond to desired behaviors of the system. Each individual scenario describes part of the system's behavior in the usual way scenarios do: actors correspond to people, components, or systems, and may represent all or part of the system whose behavior is of interest, or actors that are part of its environment. The events of the scenario occur in an order the scenario specifies. A scenario is allowed to begin only when it is a possible next scenario and its precondition is true; then the scenario can be triggered by its first event. The possible next scenarios are the ones that have an arrow to them from a scenarios that have a finger (or dime) on it (see page 67); initially there is a single finger on an initial scenario. As many as desired of the ramify set scenarios can be triggered, those that have a "T"'d arrow to them from a scenario with a finger on it. But only one of the follow set scenarios can be triggered, because then the finger moves to the newly triggered scenario. After a scenario is triggered, its postcondition takes effect and may change the values of the primitive

terms of the scenario network; if two scenarios or more are triggered simultaneously, their postconditions are applied in some sequence, but we do not define which of the possible sequences is used.

We note that it is possible for a scenario network simulation to "hang", because its scenarios can set the primitive terms so no possible next scenario's precondition is satisfied. In such a situation the multipath does not terminate and the result of the simulation is undefined.

The primitive terms of a scenario network may be individual booleans or integers or values of other sets. For example, in the EMS, there are primitive terms whose values are drawn from the set of all subscribers, a finite but dynamically changing set; others whose values are strings, lists, or other containers; and other terms whose values are attributes of a particular subscriber, analogous to object-oriented fields or mappings from the set of subscribers onto another set of values.

4.4 Relationships between scenarios

Several kinds of scenario relationships are evident from a scenario network. We discuss ordering and equivalence relationships that express which scenarios are similar because they can follow the same scenarios; or because they can precede the same scenarios; or because they can follow the same ones and precede the same ones. Examples of these relationships can be seen in the scenario network for the EMS-8 scenarios of Table 4.1. The scenario network diagram in Figure 4.3 shows that:

- S_2 and S_{30} are follow-equivalent because they can both ramify from S_1, and from no other scenario.
- S_2, S_{12}, and S_{13} are precede-equivalent because they can all precede S_{12}, S_{13}, and S_{29}, and no other scenario.
- S_{12} and S_{13} are sequence-equivalent because they can follow exactly the same scenarios (S_2, S_{12}, and S_{13}) and also precede exactly the same scenarios (S_{12}, S_{13}, and S_{29}).

The ordering relationships are analogous. Instead of following or preceding the same scenarios, for the ordering relationship we speak of following or preceding at least the same scenarios, and possibly additional ones.

The equivalence classes for the EMS-8 scenario network are listed in Table 4.7. We use the standard bracket notation $[S_e]$ for "the equivalence class containing e", and use subscripts to distinguish follow $[S_a]_f$, precede $[S_b]_p$, and sequence $[S_c]_s$ equivalence classes.As in this case, there are always at least as many sequence-equivalence classes as there are follow- and precede-equivalence classes, because sequence equivalence is a refinement of both follow and precede equivalence. For the same reason, each sequence-equivalence class is a subset of one follow-equivalence class and one precede-equivalence class.

The follow, precede, and sequence ordering and equivalence relationships are significant for several reasons:

- The equivalence relationships partition the scenarios for a system into equivalence classes. This is a novel result already useful for scenario management, with potential for wider application.
- The relationships can be determined automatically from examination of a scenario network.
- The relationships can be monitored and preserved during specification evolution by maintaining the scenario network as a system specification.

Each of these relationships can be expressed formally in terms of follow sets and ramification sets. In the definitions below, S_A, S_B, and S_C represent arbitrary scenarios. These relationships are listed in Table 4.5 and defined below.

The elementary relations *can be followed by* ($\rightharpoonup$) and *can be followed by in sequence* or *in ramification* ($\overset{s}{\rightharpoonup}$ or $\overset{r}{\rightharpoonup}$) express follow and ramification sets as relations.

$$S_A \overset{s}{\rightharpoonup} S_B \quad \overset{\triangle}{=} \quad S_B \in \text{Follow}(S_A)$$

$$S_A \overset{r}{\rightharpoonup} S_B \quad \overset{\triangle}{=} \quad S_B \in \text{Ramify}(S_A)$$

$$S_A \rightharpoonup S_B \quad \overset{\triangle}{=} \quad S_B \in \text{Follow}(S_A) \cup \text{Ramify}(S_A)$$

Follow-equivalence	
Class	Members
$[S_0]_f$	S_0
$[S_1]_f$	S_1
$[S_2]_f$	S_2 S_{30}
$[S_{12}]_f$	S_{12} S_{13} S_{29}
$[S_{39}]_f$	S_{39}

Precede-equivalence	
Class	Members
$[S_0]_p$	S_0
$[S_1]_p$	S_1 S_{29} S_{39}
$[S_2]_p$	S_2 S_{12} S_{13}
$[S_{30}]_p$	S_{30}

Sequence-equivalence	
Class	Members
$[S_0]_s$	S_0
$[S_1]_s$	S_1
$[S_2]_s$	S_2
$[S_{12}]_s$	S_{12} S_{13}
$[S_{29}]_s$	S_{29}
$[S_{30}]_s$	S_{30}
$[S_{39}]_s$	S_{39}

Figure 4.7: EMS-8 follow, precede, and sequence equivalence classes

S_A *can precede* S_B	$\rightharpoonup$
S_A *can precede* S_B in sequence	$\overset{s}{\rightharpoonup}$
S_A *can precede* S_B in ramification	$\overset{r}{\rightharpoonup}$
S_A *can begin whenever* S_B *can*	$\sqsupseteq_{\text{fol}}$
S_A *can precede anything* S_B *can*	$\sqsupseteq_{\text{pre}}$
S_A *can be substituted for* S_B	$\sqsupseteq_{\text{seq}}$
S_A *is follow-equivalent to* S_B	$\equiv_{\text{fol}}$
S_A *is precede-equivalent to* S_B	$\equiv_{\text{pre}}$
S_A *is sequence-equivalent to* S_B	$\equiv_{\text{seq}}$

Table 4.5: Scenario relationships arising from scenario networks

Extended by universal quantification, these relations produce the *follow subtype* relation $\sqsupseteq_{\text{fol}}$ and the *precede subtype* relation $\sqsupseteq_{\text{pre}}$. $S_A \sqsupseteq_{\text{fol}} S_B$ ($S_A \sqsupseteq_{\text{pre}} S_B$) if S_A can follow (precede) any scenario that S_B can:

$$S_A \sqsupseteq_{\text{fol}} S_B \triangleq \forall S_C.(S_C \xrightarrow{s} S_B) \Longrightarrow (S_C \xrightarrow{s} S_A) \wedge (S_C \xrightarrow{r} S_B) \Longrightarrow (S_C \xrightarrow{r} S_A)$$

$$S_A \sqsupseteq_{\text{pre}} S_B \triangleq \forall S_C.(S_C \xleftarrow{s} S_B) \Longrightarrow (S_C \xleftarrow{s} S_A) \wedge (S_C \xleftarrow{r} S_B) \Longrightarrow (S_C \xleftarrow{r} S_A)$$

When S_A is both a follow subtype and precede subtype of S_B, then S_A may be substituted in any place where S_B appears. The combination of these two relationships produces the *can be substituted for* relationship $\sqsupseteq_{\text{seq}}$.

$$S_A \sqsupseteq_{\text{seq}} S_B \triangleq (S_A \sqsupseteq_{\text{fol}} S_B) \wedge (S_A \sqsupseteq_{\text{pre}} S_B)$$

Each of these reflexive and transitive relations $\sqsupseteq_{\text{fol}}$, $\sqsupseteq_{\text{pre}}$, and $\sqsupseteq_{\text{seq}}$ is the basis of an equivalence relation. The *follow equivalence* relation $\equiv_{\text{fol}}$ is true when $\sqsupseteq_{\text{fol}}$ holds in both directions. Two scenarios are follow-equivalent when they can follow all the same scenarios.

$$S_A \equiv_{\text{fol}} S_B \triangleq (S_A \sqsupseteq_{\text{fol}} S_B) \wedge (S_B \sqsupseteq_{\text{fol}} S_A)$$

Two scenarios are *precede equivalent* ($\equiv_{\text{pre}}$) when they can precede all the same scenarios.

$$S_A \equiv_{\text{pre}} S_B \triangleq (S_A \sqsupseteq_{\text{pre}} S_B) \wedge (S_B \sqsupseteq_{\text{pre}} S_A)$$

A third and stronger equivalence relation *sequence equivalence* $\equiv_{\text{seq}}$ is true when both $\equiv_{\text{fol}}$ and $\equiv_{\text{pre}}$ hold. Two scenarios are sequence-equivalent when either can be substituted for the other in a scenario sequence.

$$S_A \equiv_{\text{seq}} S_B \triangleq (S_A \equiv_{\text{fol}} S_B) \wedge (S_A \equiv_{\text{pre}} S_B)$$

$\sqsupseteq_{\text{fol}}$ and $\sqsupseteq_{\text{pre}}$, $\sqsupseteq_{\text{seq}}$, $\equiv_{\text{fol}}$ and $\equiv_{\text{pre}}$, and $\equiv_{\text{seq}}$ indicate a range of possibilities of substitution of one scenario for another. Each of these substitution relationships offers the possibility of generating plausible new sequences of scenarios by substituting into sequences already deemed acceptable, and directs attention to related scenarios that may need to change together if either is changed.

The scenario relationships that arise in a scenario network are valuable in addressing scenario management because they are easily-obtained approximations to the difficult-to-obtain semantic relationships that are needed in scenario management. The relationships may be used as starting points from which to refine the exact relationships, or may be used directly as approximations that are valuable in directing the attention of a person. These relationships may be used to classify scenarios into equivalence classes, as a starting point from which to locate implicit dependency relationships, and as a framework to support the consistent percolation of changes to the specification during evolution.

4.5 Constructing a scenario network

We construct scenario networks in a process of recursive decomposition described as follows. Begin with a simple, high-level narrative that describes the general operation of the system. For the EMS, that simple narrative was "EMS takes messages from callers, and subscribers retrieve their messages from EMS." Recursively decompose each narrative into two or more more detailed narratives of smaller scope that occur in sequence, concurrently, or as alternatives. Whenever possible, decompose a narrative into the narratives related by individual scenarios. The individual scenarios are the stopping points of the decomposition. Continue until all the narratives have been decomposed into scenarios. In our studies so far, this process has terminated quickly after only a few iterations. There will probably be leftover scenarios that were not used in any decomposition. Incorporate these into the network by identifying which network scenarios they are alternatives to, in the sense of the follow-, precede-, and sequence-equivalence relationships defined in Section 4.4 and link them in correspondingly.

As we discuss in Chapter 5, the process of constructing a scenario network has proven valuable as a means of validating a scenario collection. This process focuses the analyst's attention on boundaries between scenarios, what each scenario expects and produces, and which scenarios are approximately follow-, precede-, or sequence-equivalent to each other. Our experience has shown that this process is an effective and rapid means of

- determining the context of each scenario;
- locating gaps and overlaps between scenarios;
- establishing the appropriate scope of each scenario's event sequence;
- understanding the temporal and causal relationships between scenarios;
- making pre- and postconditions consistent with those of other scenarios and with a scenario's own event sequence;
- assessing and improving completeness; and
- guiding the analyst's attention and work.

4.6 Compressing a scenario network with equivalence classes

We have shown a few ways of making scenario network diagrams scaleable to large scenario collections: we can use the shorthand notations of bidirectional arrows and especially Cartesian products of arrows to reduce the clutter of a complex diagram. These measures are absolutely necessary for scenario networks such as that for the EMS, whose 514 transitions would be unmanageable if all had to appear on the diagram individually. In this section we discuss a method for reducing the number of individual nodes that appear on the diagram and in tables, as well as the number of transitions, by using sequence-equivalence classes.

All the members of a sequence-equivalence class can be preceded by exactly the same scenarios and followed by exactly the same scenarios, respecting the distinction between sequence and ramification. Then in a scenario network, which relates scenarios based on precedes and follows, all the nodes for scenarios in the same sequence-equivalence class can be merged into one node whose incoming and outgoing arrows will match those of all members of the class. Having done this, we can then merge all transitions that come from the same node and go to the same node, respecting the distinction between sequence and ramification.

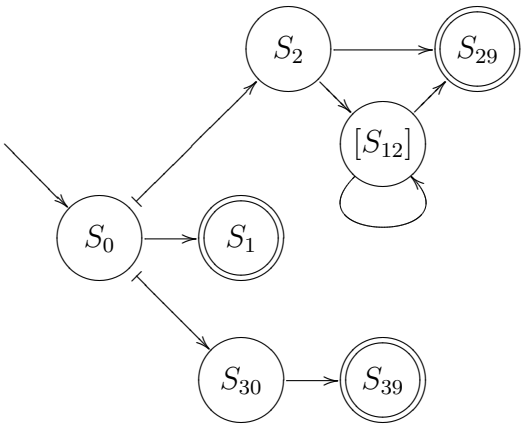


Figure 4.8: Compressed scenario network diagram for the EMS-8

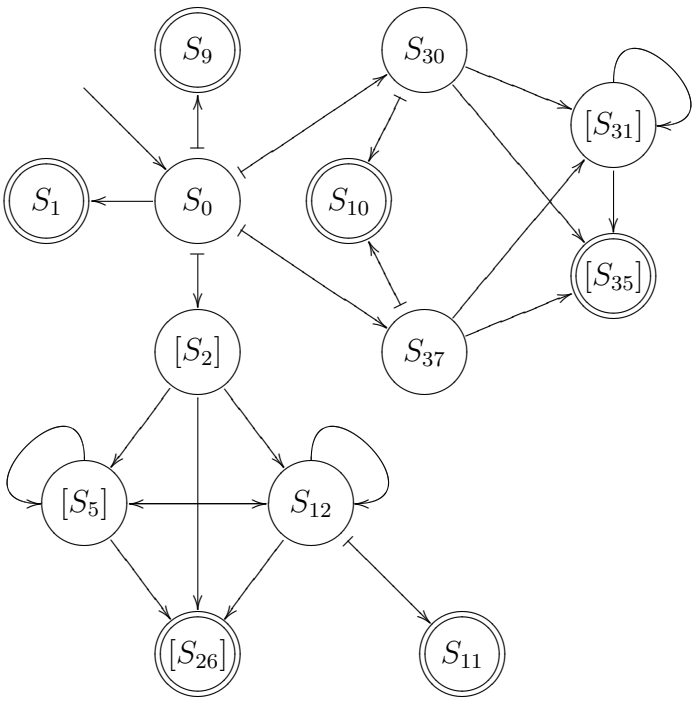


Figure 4.9: Compressed scenario network diagram for the EMS

The result of using this compression on large scenario network diagrams is substantial, and is discussed in Section 5.7.

4.7 Inverting a scenario network

Scenario networks may also be presented in *inverted form*, in which the scenarios are represented by transitions between nodes, and nodes represent externally visible system states that distinguish which scenarios can occur next. An inverted scenario network is considerably more abstract than a scenario network, which makes it more powerful but at the same time more difficult to grasp. Inverted scenario networks are of interest for several reasons.

- An inverted scenario network is much simpler than the corresponding scenario network, while conveying the same information.
- An inverted scenario network identifies high-level system states (or state classes).
- Equivalence classes of scenarios are visually evident in an inverted scenario diagram.
- Inverted scenario network diagrams clarify the significance of equivalence classes of scenarios.

The EMS-8 scenario network of Figure 4.3 is shown in a compressed inverted scenario network diagram in Figure 4.10. A key to the inverted scenario diagram notation is given in Figure 4.11.

Uncompressed inverted scenario networks are not of much interest; as a diagram, they look like a regular scenario network whose node labels have been moved back onto the transitions leading into the nodes. We briefly discuss how an inverted scenario network can be compressed. It is clear that two nodes whose *out* transitions are the same (respecting concurrency) can be merged, without consideration of *in* transitions. All of the *in* transitions can then lead on to the same *out* transitions as they did before. When all such pairs have been merged, each node has a unique group of *out* transitions and thus corresponds to a precede-equivalence class of the scenario network. Once the nodes have been merged,

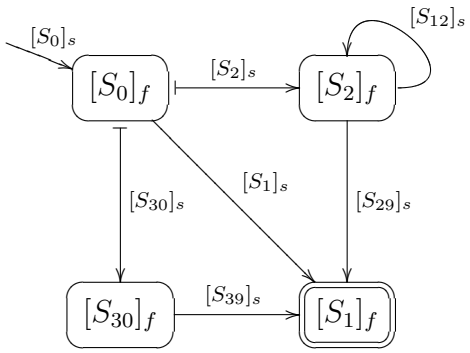


Figure 4.10: Inverted scenario network diagram of EMS-8 scenario network

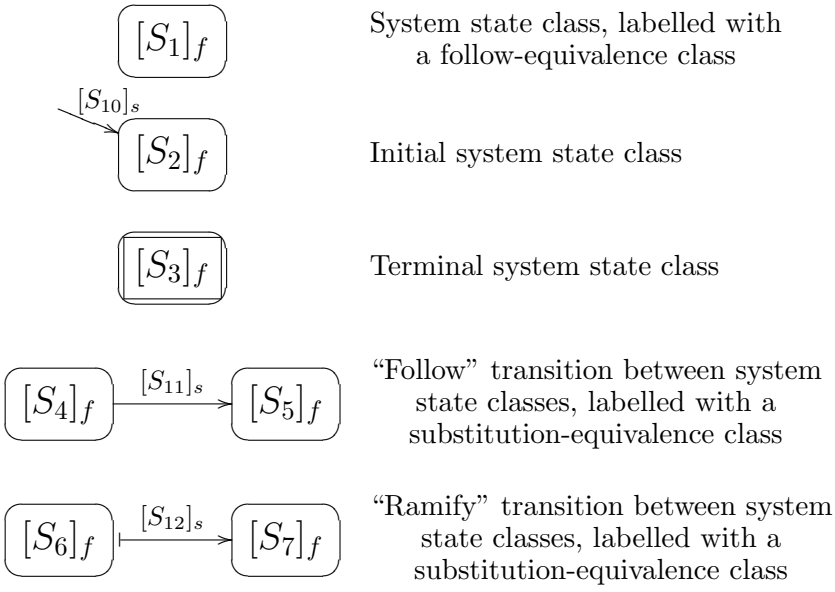


Figure 4.11: Key for the inverted scenario network diagram notation

we can merge pairs of transitions that come from the same node, go to the same node, and are both either sequence or ramification transitions. When all such pairs have been merged, each transition corresponds to a sequence-equivalence class of the scenario network. For convenience, we label the nodes and transitions with the corresponding equivalence classes.

The context of scenarios in the network, and the sequence and ramification connections between them, are preserved during the inversion and compression because both processes respect the relationships between the scenarios. The relationships are merely presented in a different form. Since the uncompressed inverted scenario networks are not of interest, when we refer to inverted scenario networks it is assumed we mean compressed networks. In the next few paragraphs, we will compare inverted scenario networks with ordinary scenario networks.

In a scenario network diagram, there is a node for each scenario S_A, and a transition between nodes for each scenario S_B that can follow it sequentially or in ramification. Nodes are labelled with the corresponding scenario, and transitions are not labelled at all because there is no need. In an inverted scenario network diagram, in contrast, there is a transition arrow for each sequence-equivalence class of scenarios $[S_C]_s$, and a node for each precede-equivalence class of scenarios $[S_D]_f$. We label the transitions and nodes with the corresponding equivalence classes. Table 4.6 summarizes these differences.

In a scenario network diagram, if S_B is in S_A's follow set or ramification set then there is a transition from node S_A to node S_B. In an inverted diagram, if S_b is in S_a's follow or ramification set then

there is a node that has an *in* transition for $[S_a]_s$ and an *out* transition for $[S_b]_s$.

If S_b is in S_a's ramification set then the *out* arrow for S_b is marked with a bar, and is termed a ramifying transition. The remaining transitions, for which S_b is in S_a's follow set, are

	In a scenario network:	*In an inverted scenario network:*
A node	for each scenario	for each precede-equivalence class
A transition	for each "can follow" relationship	for each sequence-equivalence class

Table 4.6: Comparison of scenario network and inverted scenario network

sequential and are drawn with a plain arrow.

One or more transitions of an inverted scenario network are initial; these begin from "nowhere" rather than from a node, and are labelled with the sequence-equivalence class $[S_c]_s$ of an initial scenario S_c. One node of an inverted scenario network is terminal; this node has no *out* transitions, and is emphasized with a double outline. The terminal node is labelled with the precede-equivalence class of all the terminal scenarios; there is exactly one such class, because no terminal scenario may be followed by any scenario.

As an example, compare the EMS-8 scenario network diagram in Figure 4.3 with the EMS-8 inverted scenario network diagram in Figure 4.10. Consider scenario S_2, whose follow set is $\{S_{12}, S_{13}, S_{29}\}$ and whose ramification set is empty. The scenario network has a node for S_2, and a transition from S_2 to S_{12}. The node is labelled "S_2", and the transition is not labelled. The inverted scenario network has a node for $[S_2]_f$, and a transition for $[S_2]_s$. Since $S_2 \rightharpoonup S_{12}$, there is a node ($[S_2]_f$) that has an *in* transition labelled $[S_2]_s$ and an *out* transition labelled $[S_{12}]_s$. The *out* transition is a plain arrow because $S_2 \stackrel{s}{\rightharpoonup} S_{12}$.

S_0 is the initial scenario in the scenario network, so the inverted scenario network has one initial transition and it is labelled $[S_0]_s$. S_1 is the terminal scenario in the scenario network, so the inverted scenario network's sole terminal node is labelled $[S_1]_f$.

The scenario network has four precede-equivalence classes and seven sequence-equivalence classes, and the inverted scenario network has four nodes, labelled with the precede-equivalence classes, and seven transitions, labelled with the sequence-equivalence classes.

A multipath can be traced on an inverted scenario network diagram analogously to the way it is traced on a scenario network diagram. Recall that each scenario corresponds to a transition of the inverted scenario network. We trace the multipath of Figure 4.1 by placing a finger on the initial transition $[S_0]_s$ and sliding it to state $[S_0]_p$; the sliding corresponds to the occurrence of S_0's event sequence. Leaving a finger on $[S_0]_p$, we send out additional fingers for the seven multipath branches; some along $[S_2]_s$ for the branches that begin with S_2, the rest along $[S_{30}]_s$ for the branches that begin with S_{30}. The S_{30} branches terminate when the finger slides along the transition for terminal scenario S_{39}. When the finger arrives at terminal node $[S_1]_f$, the scenarios for that branch have already

finished and nothing remains but to take the finger off the diagram. Similarly, the fingers for branches beginning with S_2 slide to $[S_2]_f$ and then follow the self-loop $[S_{12}]_s$ for both S_{12} and S_{13}; recall that S_{12} and S_{13} are sequence-equivalent and are both in $[S_{12}]_s$. Sliding a finger along transition $[S_{29}]_s$ terminates the branch, and the finger leaves after touching node $[S_1]_p$. The last finger traces the occurrence of S_1 as it slides along transition $[S_1]_s$ and terminates the trunk; the finger arrives at $[S_1]_p$ and leaves the diagram.

4.8 Scope of application

For what classes of systems are scenario networks useful, for validation and for specification? We believe they are useful for the vast majority of systems for which a collection of scenarios is a plausible specification. There are many systems for which scenarios are not a good choice, such as concurrent systems and high-reliability systems. These kinds of systems have more rigorous and mathematical specification methods, and repay the time and effort spent on a more formal specification. Since scenario networks are based upon scenarios, scenario networks will not be effective for classes of systems that scenarios themselves do not address effectively.

There remains one class of systems for which scenario networks are less useful for specification, even if scenarios are an appropriate choice: systems whose behavior has little visible relation to a system state (or equivalently to its history of interactions). An example would be an elevator, or a simple electronic wristwatch. Such systems have a relatively small number of high-level states, and although their behavior may be fairly complex, they appear to allow almost any possible interaction at almost any time, or at times that are not predictable by a particular actor interacting with them. An elevator's call buttons can be pressed at any time, for example; and although its doors do not open and shut at any time, the times when they do are relatively unpredictable.

We can more easily characterize this group of systems from the "inside out", in terms of equivalence classes of scenario networks. A scenario network expresses the "can follow" relationship among scenarios, and distinguishes scenarios that do not follow the same scenarios, or are not followed by the same scenarios. The extremes of distribution of

the "can follow" relationship among a collection of scenarios are:

1. No scenario can follow any other.

2. Each scenario can follow every other.

A system for which (1) is true can accomplish virtually nothing, and systems for which it is nearly true do not have much flexibility in their actions. Systems such as those are of little interest in requirements engineering. We need not be concerned that scenario networks would not be useful for them. We do need to consider case (2), however. If nearly every scenario in a collection can follow every other, then the "can follow" relationship does not express much information about any individual scenario, and a scenario network for that collection is not effective in helping to specify the behavior of the system. The network will indicate an arrow between almost every pair of scenarios. Unless the exceptional scenarios are of considerable importance, a scenario network will not convey much information for systems of that class. Such systems are pathologically unsuited for effective specification using a scenario network.

In terms of validation, scenario walkthroughs for such pathological systems will not be aided much by scenario networks, for the same reasons. However, the other validation benefits of scenario networks accrue for pathological systems as well as for the far more numerous ordinary systems. So we can state that for pathological systems, scenario networks are useful, but in fewer ways, and of little use in specification.

Pathological systems are straightforwardly characterized in terms of the follow-equivalence classes of the scenario network. For a pathological system, nearly all the scenarios are in the same follow-equivalence class, except for the terminal scenario(s) which are in a class of their own. We see that such a system will have very few high-level system states that are distinguished by which scenarios can occur.

The other extreme of systems, (1), is also straightforwardly characterized in terms of equivalence classes: their equivalence classes are all singletons.

Table 4.7 compares the number of scenarios, episodes, and equivalence classes for the three systems used in case studies in Chapter 5. The Elevator system was chosen because of its pathological unsuitableness, indicated by its 2 follow-equivalence classes. The

other two systems display a more intermediate distribution of equivalence classes, and are well-suited to the use of scenario networks for both validation and specification.

4.9 SNeAT

As part of our research we have produced a tool to support the validation and analysis of scenario networks. SNeAT, the Scenario Network Analysis Tool, is a command-line filter that reads a scenario network in a text input language, transforms, analyzes, or summarizes it, and translates the result into one of several output formats. SNeAT automates many of the tedious tasks that are involved in working with scenario networks, such as checking for basic validity as a scenario network, transforming a network from one form to another, and analyzing it to produce statistics, equivalence classes, and other results.

SNeAT takes a scenario network and produces essentially all possible conversions and formattings of it. The possible outputs are listed in Table 4.8.

An example of SNeAT input for the EMS-8 scenario network is presented in Figure 4.12. The input language consists of tagged parenthesized lists; details of the input language and of the outputs and operation of SNeAT are given in Appendix D.

All of the diagrams, tables, analyses, and statistical summaries of scenario networks that appear in this dissertation were produced by SNeAT and (except for manual diagram layout) were directly `\input` into the LaTeX source.

			Equivalence classes			
System	Scenarios	Transitions	Episodes	Sequence	Follow	Precede
Elevator	9	54	0	3	2	2
EMS	38	514	2	12	7	6
Euronet	40	571	49	18	13	12

Table 4.7: Number of scenarios and episodes and sizes of equivalence classes

Number of scenarios and episodes and sizes of equivalence classes for specific systems

Product	*Inversion*	*Compression*	*Output format*
Scenario network diagram			LaTeX
Scenario network table	normal	expanded	(XY-pic for diagrams)
Equivalence classes (3 kinds)	*or*	*or*	Input format
Equivalence class cross-reference	inverted	compressed	*or*
Precede and trunk sets			Macro format (m4)

Table 4.8: SNeAT's output types and formats

```
(sn EMS-8node
 (i 0)
 (t 1 29 39)
 (s 0 (sortKey 00) (f 1) (r 2 30))
 (s 1 (sortKey 01))
 (s 2 (sortKey 02) (f 12 13 29))
 (s 12 (f 12 13 29))
 (s 13 (f 12 13 29))
 (s 29)
 (s 30 (f 39))
 (s 39)
)
```

Figure 4.12: EMS-8 scenario network in SNeAT input form

4.10 Summary

In this chapter we have defined what a scenario network is and how it can be used to validate a set of scenarios. A ***scenario network*** is comprised of a group of scenarios and the temporal relationships between them that indicate the allowed scenario sequences and concurrency. At least one of the scenarios is distinguished as initial, and at least one other as terminal. Scenario networks express the context that is absent from a collection of scenarios; the construction of scenario networks uncovers inconsistencies, incompleteness, and other problems in the scenarios and the collection and provides guidance in addressing them by focusing the analyst's attention successively and selectively; and the scenario relationships expressed in scenario networks partition the scenarios into equivalence classes and form a basis for attacking the problems of scenario management. The automated support provided by `SNeAT` makes scenario networks a practical approach for dealing with scenario problems. We have listed the components of our SNCA technique, Scenario Network Construction and Analysis, which comprises

- The use of the scenario network construction process to provide guidance in identifying and (if desired) resolving inconsistencies between scenarios.
- The use of the scenario network construction process to provide guidance in identifying and filling in areas of incompleteness.
- Analysis of a scenario network to identify equivalence classes of scenarios, and use of these equivalence classes to identify and resolve further inconsistencies between scenarios.
- Examination of equivalence classes to identify potential dependency relationships among scenarios.
- Identification of scenario relationships arising from the network that should be preserved during evolution of the specification, and use of the network to track and preserve these relationships.

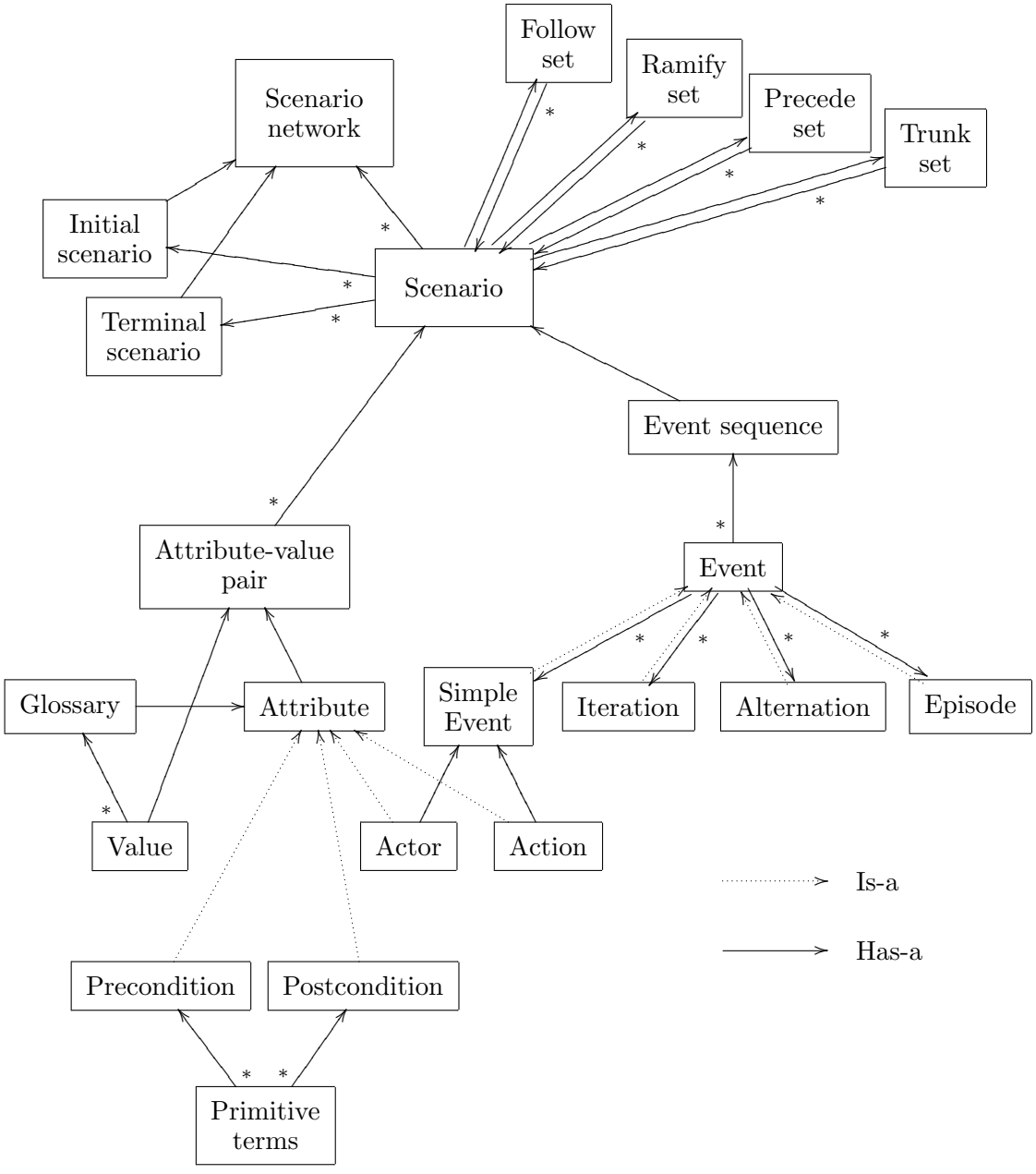


Figure 4.13: Meta-model for scenario networks and formalization

A meta-model relating the concepts introduced in this chapter and the previous one is given in Figure 4.13.

We discuss our experiences with SNCA again in Chapter 5.

Chapter 5

Validation

> Πρῶτον εἰπεῖν περὶ τί καὶ τίνος ἐστὶν ἡ σκέψις, ὅτι περι ἀπόδειχιν και ἐπιστήμης ἀποδεικτικῆς.
>
> As a start, we must say what this inquiry is about and to what subject it belongs; namely, that is is concerned with the way in which conclusions are to be established, and belongs to the science of their establishment. *Aristotle*

In Chapter 1.5 we situated this research in terms of Shaw's characterizations of validation techniques [Sha01]. Up to this point we have presented validation by persuasion, on the grounds of the specifics of the our techniques ISA (Integrated Syntactic Analysis, Section 3, page 34) and SNCA (Scenario Network Construction and Analysis, Section 4, page 64), and examples that illustrate these techniques. In this chapter we further validate these techniques by: presenting specifications produced using them; evaluating these specifications with respect to the criteria of usefulness for each of the scenario management problem components; comparing these specifications, in some cases, against the results of alternative techniques; and analyzing the formal models that result from applying our techniques.

Three case studies form the basis for our validation work to date:

- The *BellSouth Telecommunications EMS (Enhanced Messaging System)* case study. In Chapter 4 we discussed our scenario network techniques providing examples drawn from our initial work on the EMS. In this chapter we discuss our findings from the further analysis and validation of requirements and scenarios for the EMS using SNCA;

- The *Elevator* case study. The Elevator system is a small system whose scenarios and behavior were predicted to be pathologically unsuited for SNCA and for specification by a scenario network using the criteria of Section 4.8, because most of its scenarios may occur at almost any time.

- The *Asea Brown Boveri (ABB) Euronet* case study. Euronet is a medium-sized industrial system, used internally by ABB for quote management. The case study analyzed and validated its specification using ISA and SNCA.

Each of these case studies is discussed in its own section later in the chapter. We also present an experimental evaluation of ISA's weighted similarity measure from Section 3.5, using the EMS scenarios, and a summary of our results in improving scalability of scenario network diagrams using sequence-equivalence class compression.

5.1 Case studies as validation instruments

We used case studies to evaluate and validate ISA and SNCA for several reasons. A case study is particularly appropriate for initial evaluation because it allows an exploratory approach, in which the course of the study is modified and adjusted along the way to account for what is learned. This was true for the EMS case study, which was the formative case study for scenario networks and SNCA. Case studies are also appropriate for evaluating a process whose course unfolds over time, such as ISA, SNCA, or indeed nearly any software or requirements engineering method. The case study methodology is the most satisfactory approach where there are many variables of interest and few data points, where time and/or resources do not permit enough experiments to isolate the variables individually, and where there may not be enough control over the environment to permit effective repetition of experiments. Finally, case studies are well-suited to the examination of phenomena in context rather than in isolation, and it is in the context of their use that software and requirements engineering methods are of interest [Yin94].

5.2 Evaluation criteria for the case studies

In the sections that follow, we present the results of several case studies on the use of scenario formalization and scenario networks. This section summarizes how we use these case studies to validate ISA, SNCA, and scenario networks as specifications, and outlines how we set up and conducted the case studies to produce useful evidence.

We consider ISA to be the basis for automating scenario tasks that are difficult or impractical to do by hand. Therefore, we can evaluate ISA by showing that, when automated, its approaches match or improve the results that would be obtained by hand, if sufficient time and effort were expended to get those results by hand; or that its approaches are sufficiently close to the manual results when considered with the time and effort saved by not doing the manual analysis.

We consider SNCA to be a method for use in addition to a standard method in validating a collection of scenarios. Therefore, we can evaluate its usefulness by applying it to a system that has already been specified using a collection of scenarios, and noting what additional problems are uncovered during this additional validation process. If the process consistently uncovers problems that had not been identified using other methods, and if the uncovered problems are significant, and especially if a relatively small effort and amount of time are necessary to uncover them, then we assert that SNCA is shown to be an effective method for improving the quality of a specification in the form of a scenario collection.

We consider a scenario network to be an additional means of expressing system behaviors with scenarios. Therefore, we can evaluate the effectiveness of scenario networks by using them to express the behavior of a system that has already been specified using a scenario collection, and noting to what degree the scenario network specification is more extensive, and includes context and large-scale behavior patterns absent or implicit in the scenario collection. If the scenario network specifies additional context and behavior, and if these are significant in the system's requirements or function, and especially if the scenario network is an effective means of expressing them with a relatively small amount of time and effort, and the context and behaviors could not have been effectively expressed by the collection of scenarios, then we assert that scenario networks are shown to be an effective

means of expressing context and large-scale behavior that is either implicit or not present in a scenario collection.

5.3 The EMS case study

The Enhanced Messaging System (EMS) is a voice messaging system used internally by BellSouth Telecommunications to prototype new voice mail features. We performed the case study described here during the collection and specification of the EMS requirements for another research project. The EMS is a comprehensive telephone voice messaging system which supports a wide range of functionality, including: access and authentication; subscriber interactions with the EMS (e.g. notifications and message processing); caller interactions with the EMS (e.g. recording of incoming messages and the marking of certain messages as urgent); and subscriber configuration and management functions (e.g. recording announcements and archiving messages). It was used in a research project at North Carolina State University that compared the effectiveness of two object-oriented rapid-prototyping methodologies (Rational and Bridgepoint). The first task of that research project was to establish a clear requirements specification as a basis for the application of each methodology. This was done by producing a list of requirements and a collection of scenarios, and then augmenting them with a scenario network; this was the phase of the rapid-prototyping project during which the case study for this research was conducted. We conducted the case study introspectively to ensure that the data we collected would serve to validate our scenario network method. The case study is presented in this section. It was presented in less detail in an earlier publication [AA01].

The EMS was good material for a case study because it was a real system, of manageable size, whose requirements and scenarios were not determined by us [Pot93]. A domain expert from BellSouth Telecommunications controlled the content of the requirements and scenarios, thus limiting the effect of any biases on the part of the analysts at North Carolina State University and ensuring that the scenario collection was substantial and genuine.

The EMS was also a good case study subject because the requirements and sce-

narios were written by three experienced and capable requirements engineers (including the author), and received inspections, detailed walk-throughs, and reviews over a period of eight weeks. Thus the requirements and scenarios could reasonably be expected to be of higher quality than most specifications, and the more obvious errors would have already been eliminated. A case study in which a method identified problems in this specification would indicate that the method would also uncover problems in most other specifications.

5.3.1 Case study artifacts

The EMS specification (before construction of a scenario network) consisted of 42 requirements and 32 scenarios. Requirements were expressed in traditional form, and definitions were provided for 17 words and phrases used with specialized meanings in the requirements and scenarios; Table 5.1 gives several example requirements and definitions for terms used in them. For example, R2.2 specifies that a subscriber can change his/her announcement, and D1 defines what "announcement" means in this context. The scenarios ranged in length from 2 to 12 events (not counting iterations, alternations, and episodes), and were expressed in the format outlined in Table 5.2. Two episodes (identified manually) were defined for the scenarios. Each scenario was traced back to between one and four requirements. Ten primitive terms were defined for use in the scenario pre- and postconditions. The conditions were composed of from zero to three primitive terms (five conditions were simply "true"). Table 5.3 gives several example scenarios and definitions for primitive terms used in their pre- and postconditions. For example, both the pre- and postcondition for S_5 are "Authenticated", and the primitive term definition given for "Authenticated" defines when it is true. S_5 says it is traced back to requirement R2.2, which is one of the example requirements in Table 5.1. S_5's event sequence uses "announcement", a word also defined in Table 5.1, in (for example) events 1.2 and 1.4. The event sequence contains an iteration (event 1), whose effect is to repeat events 1.1, 1.2, 1.3, and 1.4 until event 2 signals termination of the iteration (termination of this iteration was not very clearly expressed in this version of the scenario). Notice that event 1.3, part of the iteration, is also a compound event, but this one is an alternation between choices 1.3.1 and 1.3.2. Compound events are discussed in Section 3.2; the other compound event in our scenario formalization is

D1.	A subscriber's *announcement* is a recording that a caller hears when he or she reaches the subscriber's voice mailbox.
D6.	A *message state* is one of the following: new message, held message, archived message, or erased message. Each message is initially a new message.
R2.2.	EMS shall allow a subscriber to configure the announcement a caller hears before leaving a message.
R3.2.1.	EMS shall present messages in chronological order (oldest to newest), except that EMS shall present all urgent new messages to the subscriber before presenting any new message that is not urgent.
R3.2.2.	EMS shall allow a subscriber to enter a command and listen to new and held messages one after another. Listening to a new message changes its state to "held".

Table 5.1: Example definitions and requirements for the EMS

an episode, analogous to a subroutine call to a separate subsequence of events. Note that current scenario representations do not support compound events, and use case represent event sequences in ways that do not handle compound events as well as we do, especially for what use cases term "uses," "includes," and "extends" of other use cases [Sim99]; our improvement is a primary benefit of our scenario formalization and is supported in SMaRT.

A "menu tree" that was intended to express the possible sequences of scenarios was included with the scenarios; the nodes of the tree were scenarios, and the indicated sequences were those that moved between adjacent scenarios in the tree, beginning at the root and continuing up and down; it was not clear whether sibling nodes were considered adjacent.

5.3.2 Case study design

The EMS case study examined the results of augmenting the use of standard requirements engineering methods with SNCA.

Our hypotheses for the EMS case study were:

H1. A scenario network will indicate the degree of completeness of the EMS scenario col-

Heading	Contents
"Requirements"	The requirements operationationalized by this scenario.
"Preconditions"	List of named conditions, defined elsewhere in a glossary.
"Postconditions"	List of named conditions, defined elsewhere in a glossary.
"Equivalent to"	List of scenarios considered equivalent to this one (if any).
(no heading)	The scenario's event sequence. Events in the sequence are numbered outline-fashion, and may include: – iterations of subsequences; – alternative subsequences; and – references to episodes.
(no heading)	Some scenarios have prose notes

Table 5.2: The form of the EMS scenarios

lection, and show which scenarios or behaviors need additional attention to achieve greater completeness.

H2. Construction of a scenario network will uncover inconsistencies between the scenarios (we assumed such inconsistencies were inevitably present).

H3. A scenario network will provide a way to specify relationships between the scenarios that had already been discovered but not effectively specified by the collection of scenarios or the menu tree associated with it.

H4. Construction of a scenario network will uncover significant relationships between the scenarios that were not been discovered earlier.

5.3.3 Course of the case study

We began the case study by assembling the EMS scenarios into a network. Our initial procedure was to link two scenarios when the postcondition of one and the postcondition of the other were satisfied by at least one configuration of the primitive terms, and continue linking until all the scenarios were connected into a scenario network. This procedure was done by hand and was tedious and time-consuming. A more significant problem was that the resulting network connected nearly every scenario to nearly every other; many pairs of scenarios were connected that should not have been able to follow each other. It

Primitive terms used in the pre- and postconditions
Authenticated: The subscriber is connected to EMS and is authenticated.
Has $(n > 0)$ *New*: The subscriber has (n) new messages, and $(n > 0)$.
Played: EMS has just played a message.

S_5. Subscriber configures his/her announcement
Requirements: R2.2.
Precondition: Authenticated.
Postcondition: Authenticated.
1. *Iteration:*
1.1. The subscriber dials the "configure announcement" command.
1.2. EMS asks the subscriber to say the new announcement or dial the "choose default announcement" command.
1.3. *Alternation:*
1.3.1. The subscriber says his/her new announcement and dials the "end of announcement" command.
1.3.2. The subscriber dials the "choose default announcement" command.
1.4. EMS plays back the chosen announcement and asks the subscriber to dial the "accept" command or the "configure announcement" command to try again.
2. The subscriber dials the "accept" command.
3. EMS uses that announcement as the subscriber's announcement.

S_{11}. Subscriber listens to a new or held message
Requirements: R3.2.1, R3.2.2.
Precondition: Authenticated, Has $(n > 0)$ New.
Postcondition: Authenticated, Has $(n - 1)$ New, Played.
1. The subscriber dials the "listen to next new or held message" command.
2. EMS plays the next new or held message. (The next new or held message is the oldest urgent new message, if there is an urgent new message, or the oldest new or held message if there isn't an urgent new message.)
3. If the message is a new message, EMS changes its state to "held".

Table 5.3: Example primitive terms and scenarios for the EMS

was clear that the pre- and postconditions were incorrect, and did not express what each scenario needed and provided.

This approach ignored the large-scale narratives that we intuitively knew described the system. We observed that in this approach these narratives were emergent properties. In the presence of errors in the conditions, the narratives were obscured and did not emerge. We worked to improve the conditions by iteratively repairing conditions, examining pairs of scenarios for the "can follow" relationship, and trying out narratives as test cases. This procedure was followed for two days and produced slow improvement.

Our second procedure, adopted after the first proved impractical, was to use the large-scale narratives as a starting point rather than as test cases. We began by describing the entire behavior of the system with one very general story ("EMS takes messages from callers"). We proceded to recursively decompose each story into two or more smaller, more detailed stories that occurred in sequence, concurrently, or as alternatives. Whenever possible, we decomposed a story into some of the scenarios. The scenarios were the stopping points of the decomposition, and each branch of the decomposition was terminated by a scenario. Termination of all branches was reached quickly, after only a few iterations. At this point we had a recognizable scenario network composed of about half of the EMS scenarios. We incorporated the remaining scenarios into the network by identifying which network scenarios they were alternatives to (i.e. follow-, precede- or sequence-equivalent to), and linking them in correspondingly. Completion of the scenario network took two days.

Constructing the scenario network using this procedure resulted in the discovery of errors in the scenarios and in the requirements. Interestingly, all these errors were discovered in the first four hours.

5.3.4 Lessons learned

Scenario networks help uncover missing scenarios

The process of constructing a scenario network requires that analysts walk through scenario multipaths, and to compare pairs of scenarios for degrees of equivalence. Making

the scenarios operational in this way draws more focused attention to them than reviews or walkthroughs, and some errors that had previously slipped past are caught.

The following missing scenarios were identified almost immediately during construction of the scenario network using the second procedure:

S0	EMS startup
S1	EMS shutdown
S13	Subscriber has no more messages to listen to
S15	Subscriber would skip to next message but has no more messages
S19	Subscriber erases the last message
S21	Subscriber archives the last message
S29	Subscriber disconnects from EMS
S37	Caller calls EMS directly and leaves a message
S38	Caller takes no action for a long time
S39	Caller disconnects from EMS

S_0, S_1, S_{29}, and S_{39} were discovered because the scenario network lacked appropriate initial and terminal scenarios.

S_{13}, S_{15}, S_{19}, and S_{21} was discovered when the postconditions of their duals (S_{12}, S_{14}, S_{18}, and S_{20}) were found not to distinguish having some messages from having none.

S_{37} and the corresponding requirement R4.7 were discovered to be missing during the process of matching up equivalent scenarios; this behavior had been discussed but for some reason never specified.

S_{38} was discovered by visually comparing the network configuration for subscriber scenarios to that for caller scenarios; the presence of the corresponding scenario for subscribers, S_{26}, was a prominent difference.

Scenario networks help uncover requirements errors

The same operationalization that draws focused attention to scenarios also focuses attention on requirements. It forces analysts to think about individual requirements concretely and in specific contexts. As with scenarios, this focused attention identifies errors that had slipped past reviews and walkthroughs.

We discovered one requirements error by noting unexpected stub branches in the scenario network, that is, scenarios that lead somewhere but have nowhere to come from, or vice versa. Initial and terminal scenarios are intentional stub branches. The presence of

other stub branches is a sign that something is not right. In this case, we observed that $\{S_{12}\}$ "Subscriber listens to an archived message" was a stub because it lacked appropriate pre- and postcondition. The braces $\{S\}$ indicate a pre-case study scenario number. While tracing what these conditions should be, we discovered that we had incorrectly separated $\{S_{12}\}$ from $\{S_{11}\}$ "Subscriber listens to a new or held message." These two behaviors were originally to have been distinct; later discussions with the domain expert indicated that they should instead be the same, but this change was somehow not recorded in the scenarios or in the corresponding requirements, and reviews and walkthroughs missed this error. While tracing these scenarios back to their requirements, we found that not only did the requirements R3.2.2 and R3.2.3 incorrectly separate those two cases, but that the other requirement both scenarios traced back to was also incorrect. That was R.3.2.1, which incorrectly specified that messages be presented oldest-to-newest, rather than newest-to-oldest. This mistake had also been missed despite reviews and walkthroughs by the domain expert and the other analysts.

Scenario networks express when each scenario can occur

A scenario network makes explicit the temporal relationships between scenarios. Before we constructed the scenario network, we had a certain degree of intuition and informal knowledge of the patterns in which the EMS scenarios could occur. Setting down the scenario network on paper made that knowledge explicit, and revealed the ways in which our intuitive and informal knowledge had been inadequate or incorrect, some of which are described above.

5.3.5 Discussion

At the beginning of this section we listed four hypotheses we expected this case study would support.

1. That the scenario network would indicate the degree of completeness of the EMS scenarios, and show which scenarios or behaviors needed additional attention to achieve

greater completeness.

We found that SNCA did indicate incompleteness and direct attention to the areas that would resolve it. We distinguished three kinds of incompleteness in this case study, based on how instances of them were identified. The easiest kind to find, and the kind that is probably of the least interest, was missing behavior whose presence is required by the structure of a scenario network, namely the initial and terminal scenarios such as S_0. The second kind was missing behavior that was turned up through the analysis of pre- and postconditions for the relationships required by the scenario network, such as S_{13}. The third and most interesting kind was behavior whose absence was indicated visually by analogy between parts of the scenario network, exemplified by S_{38}.

That construction of the scenario network would uncover inconsistencies between the scenarios (we assumed such inconsistencies were inevitably present).

The most notable inconsistencies we found were those that led us to discover missing scenarios such as S_{13}. We also uncovered a large number of inconsistencies that indicated errors in conditions or event sequences.

That the scenario network would provide a way to specify relationships between the scenarios that had already been discovered but not effectively specified by the collection of scenarios.

Some of the temporal and causal relationships were indicated by the "menu tree" of scenarios. The scenario network expressed these relationships more correctly and more effectively. We also found that the scenario network specified equivalence and "can follow" relationships that we had been informally aware of but which did not fit into the format of the "menu tree".

That construction of the scenario network would uncover significant relationships between the scenarios that had not been discovered earlier.

We found pairs of scenarios that we had not expected to be equivalent, and pairs

of scenarios that we had never thought of as following each other. While these relationships cannot be dismissed as insignificant, we had expected to find something more interesting and unexpected, but did not.

A valuable and unexpected result from the case study was the discovery of a rapid and effective procedure for constructing a scenario network by recursive refinement of narratives.

5.4 The elevator case study

The Elevator system is a hypothetical controller for a standard passenger elevator. It takes input from the destination buttons within the elevator car and the call buttons at each floor, and controls the sequence of floors at which the elevator car stops and the opening and closing of its doors.

The Elevator system was chosen for this case study in part because it was expected to be pathologically unsuitable for a scenario network, for two reasons.

Scenario networks express which scenarios can follow which others. The two extremes of the "can follow" relationship among scenarios are a collection of scenarios none of which can follow any others, or a collection all of which can follow any other. We hypothesized that for systems near either of these extremes, scenario networks would be the least effective as a method of specification. Nearly all the Elevator system scenarios can follow themselves or nearly all the others, so we expected that creating and analyzing its scenario network would be informative.

The specification for the Elevator system was a set of requirements and scenarios written as an exercise for a class at North Carolina State University. Its specification consisted of eight definitions of terms, seven requirements, and nine scenarios. An example definition, requirement, and scenario are shown in Table 5.4. The scenario pre- and post-conditions were inadequate, one scenario was much longer than the others and overlapped with most of them, and there was no terminal scenario.

Our primary hypothesis for this case study was that a scenario network for the Elevator system would not be an effective specification, and that SNCA would not spec-

Destination set: The set of destinations for which an elevator has received calls but has not yet answered.
R1. If someone presses a call button, the elevator car stops at that floor "before too long" heading in the direction the call button specified, and opens its doors.
S_3. Call from floor 1. Someone outside the elevator on a particular floor presses an "up" or "down" call button for the elevator. 2. That floor and direction are added to the elevator's destination set.

Table 5.4: Definition, requirement, and scenario from the Elevator System

ify any non-obvious relationships between the scenarios nor uncover any relationships not known already. We also hypothesized that SNCA would nevertheless be effective in indicating completeness of the scenario collection and providing process guidance to improve completeness, and that it would indicate the inconsistencies between the scenarios. Finally, we hypothesized that the long overlapping scenario could not be effectively integrated into the scenario network.

5.4.1 Course of the case study

Construction of the scenario network proceeded quickly due to the small number of scenarios and the fact that they fall into only a few equivalence classes. The long scenario was deleted since all its events were also present in the remaining scenarios. A shutdown scenario was added to provide a terminal scenario for the network. Walkthroughs of the scenarios, using the scenario network diagram as a map, were used to identify and then verify the pre- and postconditions for the scenarios.

5.4.2 Lessons learned

A scenario network is not an informative specification if any scenario can follow any other

The scenario network for the Elevator System conveyed little information, because there was little information to convey about the temporal relations of the scenarios. All the

non-terminal scenarios could follow themselves and any other non-terminal scenario. The diagram expressed this, but it was already obvious and its expression added little.

Long scenarios that overlap with others do not fit well into a scenario network

We thought that if it were possible to fit a long overlapping scenario into a scenario network, that a simple system such as the Elevator system would be the easiest context in which to do so. We tried a number of ways to integrate the long overlapping scenario into the network, but were unable to produce a network for which the long scenario was not superfluous. All its events were specified by other scenarios, and the sequence of its events was one of the possible ones specified by the network, so there was no additional information indicated by its presence in the network.

On the other hand, it was possible to fit it into the network, by making sure its beginning and ending were consistent with the endings and beginnings of the other scenarios. Fitting it in was made simpler by the fact that most of the other scenarios could follow each other, so there was no difficulting finding where to connect in the long one. However, in the final analysis there was no reason to put it there.

Scenario networks supply the context that long scenarios are often written to express, and do so more effectively

This was graphically illustrated by the fact that the long scenario was superfluous in this network. Its event sequence, which illustrated a context for most of the other scenarios, was expressed by the network, as were all the other possible contexts (which the long scenario did not express). In this case, the scenario network expressed the context much more effectively.

5.4.3 Discussion

The diagram itself proved unexpectedly useful in directing walkthroughs of the scenarios to identify and then verify the pre- and postconditions for the scenarios. As expected,

We confirmed that a scenario network, as a specification, was not more effective than (for example) a prose statement in combination with the collection of scenarios. The

network diagram made the scenario relationships visible, but did not uncover any relationships that were not already known. SNCA was effective in indicating incompleteness (the absence of a shutdown scenario) and guiding the completing of the individual scenarios by drawing attention to problems in their pre- and postconditions, and guiding walkthroughs to verify the corrected conditions. It was something of a surprise to see that the long scenario could be integrated into the network, but the process confirmed that integrating overlapping scenarios into a network is superfluous.

5.5 The Euronet case study

Euronet is a quote management system used internally by Asea Brown Boveri (ABB) to create and manage quotes for work to be done, the line items that give the details of the quotes, and the resulting orders from customers. It was implemented several years ago from a requirements specification consisting of 52 use cases, 26 screen sketches, and a general overview [ABB99]. During the implementation this requirements specification was the subject of a research study that applied a goal-based analysis to the use cases [ACD+01]. ABB graciously allowed the use of these use cases as a case study for the syntactic and semantic approaches.

Euronet was a good subject for a case study because it was a real system, more complex than the EMS but still of manageable size, we had access to the specification that was used to implement it, and that specification had been the subject of an earlier goal analysis whose results were available to us. This gave us two points of comparison, the original specification and the goal analysis.

The 52 Euronet use cases are listed in Table C.1. The requirements specification document presents the use cases in a relatively consistent format [ABB99]. Each use case is numbered (1 through 52) and has a short title. The content of each use case is organized as shown in Table 5.5. The event lists range in length from 1 to 19 events, with 3 events being the most common length. 25 of the use cases sketch one or more secondary (alternative) scenarios. Most of the use cases have pre- and postconditions (51 and 43 respectively). The use cases use each other as events; 27 of the use cases are referred to in event lists, and

individual use cases use as many as seven other use cases. The "uses" hierarchy (Figure 5.1) is acyclic and has a maximum top-to-bottom path of 5 use cases. The use cases vary from one to six pages in length, with two pages being the most common.

The results of the goal-based analysis done earlier on the Euronet use cases provides a standard against which to compare the results of ISA and SNCA. The goal-based analysis required approximately two months work by three researchers.

The goal of the Euronet case study was to compare the results of applying ISA and SNCA to the results of applying a goal-based analysis, and to the results of ABB's original requirements engineering work as embodied in the specification.

5.5.1 Course of the case study

A careful syntactic examination of the Euronet use cases immediately revealed a large number of problems, ranging from the annoying to the serious. The use cases named in the "Utilizes" section of each use case description frequently do not match the use cases that appear in the event list. Twelve apparent use case names are referred to in events but not defined. Ten screens named in the use cases are not defined with a screen shot like the others; four of the ten are probably equivalent to similarly-named defined screens, leaving six that are definitely undefined. Of the 26 defined screens, three are never referenced in any use case. The "Required GUI" lists for 35 of the use cases match the screens referenced in the use case, making this the most consistent of the easily-checked aspects of the Euronet use cases.

The "uses" hierarchy among the use cases highlights the presence of use cases that are referenced but not defined. The earlier goal-based analysis included the production of "uses" trees for individual use cases, and discovered the same undefined use cases [ACD+01]. The complete hierarchy for the entire collection (Figure 5.1) draws attention to other potential areas of interest: the depth of the hierarchy relative to the number of use cases involved, and two patterns of strongly interconnected groups within the hierarchy.

The hierarchy is surprisingly deep compared to the number of use cases. There are 64 use cases named in the Euronet specification, 52 with definitions and 12 without definitions. Of these 64 use cases, 53 appear in the hierarchy, and the remaining 11 neither

Heading	Contents
"Overview"	A one- to three-sentence summary
"Preconditions"	List of prose preconditions, usually numbered
"Main Scenario"	List of prose events, usually numbered; some events name one to four other use cases
"Scenario Notes"	Information that doesn't fit elsewhere
"Postconditions"	List of prose postconditions, usually numbered
"Required GUI"	List of GUI screens used in the use case
"Secondary Scenarios"	Numbered list of alternative scenarios; each is given as a prose paragraph
"Utilizes"	List of use cases used by this use case
"Extends"	(No Euronet use case extends another one)
"Other Requirements"	(No Euronet use case references any)
"Revision History"	Date (11 August 1999), description "Initial release", and the author's name

Table 5.5: The form of the Euronet use cases

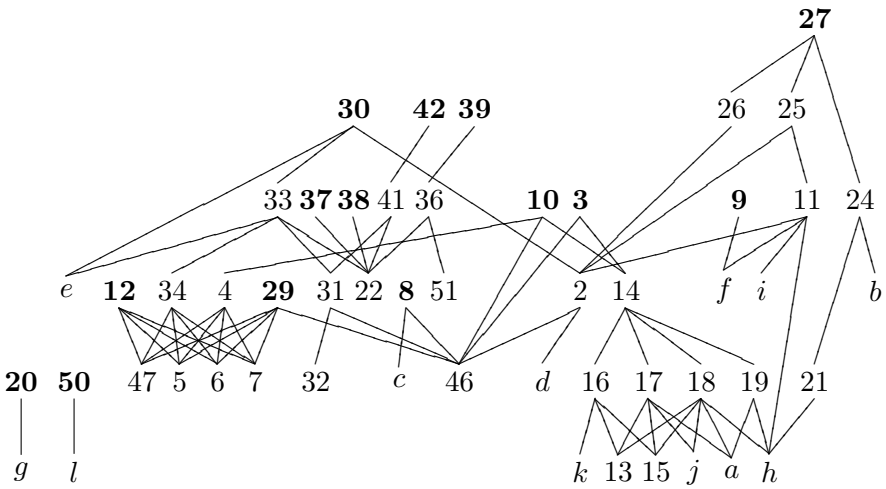


Figure 5.1: Euronet "uses" hierarchy

use nor are used by any other use case. The four deepest "uses" paths connects five use cases each, or about 9% of the nodes in the hierarchy. Twenty-six paths connect four use cases each, or about 8% of the nodes. While we cannot draw any specific conclusions from this depth, we can state that it indicates possible concerns.

We can draw more specific conclusions from the interconnections in some areas of the hierarchy. The complete connections from use cases 12, 34, 4, and 29 to 47, 5, 6, and 7 indicate that the upper use cases should be similar in other ways, and raise the question of whether 47, 5, 6, and 7 should be separate use cases since they only appear together (and in the same sequence, although the hierarchy does not show this). Use cases 16, 17, 18, 19, and 21 are less completely connected to k, 13, 15, j, g, and h, but still indicate that attention should be paid to them; 16, 17, and 18 should be similar in other ways, and one should inquire whether 13, 15, j, and a should be separate use cases.

5.5.2 Lessons learned

Parameterizing actions, events, and episodes enables substantial re-use

We noted a number of instances of actions, events, and episodes that were identical except for one or two words. For example, we found several actions of the form "requests X" for various X's, and corresponding events of the form "U : requests X" for various users U and the same X's. A concrete example was "BA Engineer : requests Report". In a few cases, this continued up to the level of episodes; for example the episodes Ep_{F1} "Save/Close Item," Ep_{F2} "Save/Close Order," Ep_{F3} "Save/Close Quote," and Ep_{F4} "Save/Close Quote Package." If actions, events, and episodes could be parameterized, then fewer individual ones would be needed, and related ones could be standardized in a way that could be automatically supported.

Standardizing on a relatively small number of important words to use in actions makes it easier to find the right action and to express the actions effectively

The events in the original Euronet use cases were worded in no consistent fashion. We found that choosing a set of standard words to use wherever possible greatly reduced the number of separate actions, and at the same time made the events clearer and easier to

understand relative to each other. Examples were the use of "select" in place of "choose," "find," "click," etc. whenever it was appropriate.

This lesson was analogous to the similar finding for goal wordings by Anton *et al.* [ACD+01].

Glossaries help reduce undesirable ambiguity in the specification

The original Euronet specification lacked a glossary, and we found it difficult to understand the significance of certain terms in the use cases (for example, the verb "close" which was clearly used with a special significance which we recognized but did not understand). We also found it difficult to separate out the appropriate meanings of terms that appeared to be used in two senses (for example, "order" was used to mean an order from a customer to ABB, and also an order from ABB to a supplier, and the distinction was not always clear from context). Finally, we found it difficult to tell when two terms were being used for the same concept (for example, it was unclear at first whether a "Planner" and a "BA Planner" were the same).

Automated episode identification finds duplication that people have not

Although SMaRT's episode identification was not implemented at the time we performed this case study, the use of SMaRT helped us notice that UC_{42} "Change Language" partially duplicates a secondary scenario of UC_1 "Log On"; and the use case hierarchy and our episode cross-reference showed us that the four episodes Ep_{47}, Ep_5, Ep_6, and Ep_7 always appeared together in the same sequence, and never otherwise, and thus that they should have been linked in a single episode. The Euronet analysts were apparently unaware of these repetitions.

Asking questions about alternatives and obstacles helps identify missing scenarios

While examining $UC_{2(}$ "Create Order Without Quote" and considering alternatives to its events with respect to its pre- and postconditions, we realized that there were use cases to look up or identify a customer, but no use case to create a new customer.

This heuristic parallels Anton and Pott's finding that obstacles in goal analysis help uncover scenarios [AP98b], and Potts *et al.*'s finding that examination of alternatives helps uncover requirements [PTA94].

Syntactic similarity depends on the use of glossaries

We manually detected the similarity between UC_8 and UC_9 but realized that SMaRT's algorithms for finding episodes and measuring similarity would not have identified the similarity because the events of the two use cases were worded so differently although their meaning was the same. This confirms our hypothesis that the effectiveness of these two algorithms depends on the appropriate use of glossaries so that the same meanings are expressed in the same words.

Many inconsistencies should be immediately reconciled, not managed

In the inconsistency-management approach, inconsistencies are identified and then not reconciled (necessarily) but managed for some period of time in order to extract the most information from them. However, the great majority of the inconsistencies we identified with SMaRT and ISA were not informative but simply indicated missing information. For example, the use cases referred to a large number of statuses of various items, but these statuses were never defined and in many cases an entity's status was set but never examined, or examined but never set. We believed many of these were due to the fact that no analyst's attention had been drawn to the inconsistencies before. SMaRT and ISA are effective at identifying such inconsistencies and drawing attention to them.

Long paths in a "uses" hierarchy may indicate problems

In the "uses" hierarchy of uses cases, we found two overlapping paths of 5 use cases. Several of the use cases in these two paths (UC_{25}, UC_{26}, and UC_{27}) were later found to have disagreements between their events and the events in the use cases they used, several levels down. We believe a deep "uses" hierarchy (more than 3) indicates likely inconsistencies between the episodes being used and the context they are used in.

5.5.3 Heuristics identified

We identified and validated a number of heuristics while analyzing the Euronet specification. Since that specification is written using the term "use case", we use that term here as well in referring to examples where each heuristic was effective. These heuristics

offer process guidance for working on scenarios. Many of these heuristcs have the advantage that the situations they match can be automatically identified by SMaRT.

Every event should have an actor that performs the action

EXAMPLE: The first event of UC_{25} "Approve Quote" is

1. Euronet DB contains a complete Quote with a status of "Submitted."

A scenario with only a single event is incomplete

The first event of a scenario triggers the rest of the scenario's event sequence; if the first event is the only event, nothing is being triggered. Also, a scenario records an interaction, and one event is only an action, not an interaction.

EXAMPLE: UC_{40} "Acknowledge Customer Order" has the sole event

1. Salesperson sends an acknowledgement to the customer.

The event list was changed to specify this interaction:

1. Salesperson : requests Acknowledgement be sent to Customer.
2. Euronet : sends Acknowledgement to Customer

EXAMPLE: UC_{48} "Get Address Information" has a single event, which examination shows is actually two events conflated into one:

1. Salesperson manually enters a new address: Euronet adds the new address to the List of Existing Addresses.

A scenario with only a single actor is incomplete

An actor can't interact with him/herself; another actor is required.

EXAMPLE: UC_{37} "Acknowledge Item Build Order" consists of 2 events, but both involve the same actor:

1. Plant Planner reviews the design and deliver time information in Euronet for item build order(s).
2. Plant Planner changes item status to acknowledged.

We changed the event list to

1. Planner : requests item build order design and delivery time information for review.
2. Euronet : presents item build order design and delivery time information.
3. Planner : reviews the design for item build order.
4. Alternation:
    4.1. Alternation branch:
        4.1.1. Planner : changes item status to Acknowledged.
        4.1.2. Euronet : sends Acknowledgement to Salesperson.
    4.2. Alternation branch:
        4.2.1. Planner does not acknowledge the item build order.

The actor for each scenario-triggering event must be one whose actions are not predictable

The first event of each scenario should not be the system whose behavior is being specified, or any other actor whose actions are causally determined. Otherwise, the scenario might as well be part of another scenario whose events cause the triggering event to occur. EXAMPLE: UC_{21} "Check BA Rules" begins with an event whose actor is the system being specified:

1. Euronet displays Check Violations Screen showing all Approval Rules violated by Quote.

Therefore this use case, as it stands, is merely a continuation of whatever events cause Euronet to display that screen, not an independent scenario.

Each scenario's second event must be a response to the first, from a different actor

In the Euronet specification, this heuristic is violated in a number of use cases whose nominal first event is two conflated events

EXAMPLE: UC_{46} "Choose Customer" appears to begin with two events whose actor is "Salesperson", but in fact the first is the conflation of a Salesperson event and a Euronet event:

1. Salesperson selects List Box Button for top field; Euronet displays list of 10 most recent customers.
2. Salesperson enters customer search string in Search field ....

Triggering events should occur at a definite time, and cover a brief timespan

A triggering event should have a well-defined time at which it occurs, rather than covering a large or undefined span of time.

EXAMPLE: The first event of UC_{38} "Reject Item Build Order" may last a very long time, and has no particular point in time associated with it:

1. Plant Planner reviews the design and delivery time information in Euronet for item build order(s).

We revised the event list to begin with a "sharper" event that occurs at a particular time:

1. Planner : requests to review item build order.
2. Euronet : presents item build order design and delivery time information.

A scenario's precondition must allow its triggering event

EXAMPLE: The second precondition of UC_{12} "Edit Header" is

2. Header Screen is displayed on the Salesperson's PC with existing customer information displayed in the proper fields.

but the Header Screen has no place to enter an alternate Ship-To Address as the first event requires:

1. Salesperson enters alternate Ship-To Addresses for customer.

UC_{12} prevents itself from occurring.

A scenario's precondition must be strong enough to ensure what the scenario needs

EXAMPLE: The events of UC_{37} "Acknowledge Item Build Orders" require an item build order with design and delivery time information:

1. Plant Planner reviews the design and delivery time information in Euronet for item build order(s).
2. Plant Planner changes item status to acknowledged.

and perhaps one whose item status is not already "Acknowledged" or at least not "Cancelled", although this is unclear. The precondition for UC_{37} does not ensure any of these things:

(Precondition)

1. Plant Planner has received an item build order.

and in fact examination of other use cases shows that some item build orders are given design information, and others apparently are not.

A scenario's postcondition should express its result, not just what comes next

EXAMPLE: The postcondition of UC_{20} "Add Alternate Item" expresses what can occur next (albeit in terms of the GUI), but says nothing about what the scenario accomplished:

(Postcondition)

Shopping Cart Screen is displayed.

Each scenario's postcondition must correspond to at least one scenario's precondition

EXAMPLE: The postcondition of UC_{41} "Cancel Item Build Order" refers to an item's status being changed to "cancel"

(Postcondition)

1. Cancellation notice sent to Plant Planner and Euronet DB item(s) status changed to cancel.

but no use case's precondition refers to an item status of "cancel", either directly or indirectly, so no use case can take advantage of what UC_{41} has done.

Each scenario's precondition must correspond to at least one scenario's postcondition

EXAMPLE: A precondition of UC_{42} "Cancel Order" is

(Precondition)

2. Salesperson has created and transmitted an order.

but no other use case mentions transmission of an order, either in postconditions or events, so there is no way UC_{42} can occur.

EXAMPLE: A preconditions of UC_{12} "Edit Header" is

2. Header Screen is displayed on the Salesperson's PC with existing customer information displayed in the proper fields.

but no use case's postcondition mentions customer information, so there is no way to achieve UC_{12}'s precondition.

Pre- and postconditions and scenario networks express scenario context more effectively than extra events added at the beginning or end of a scenario

EXAMPLE: UC_{27} "Revise Quote" describes an event sequence that ends with uses of UC_{25} "Approve Quote" and UC_{26} "Assemble Quote Package" and then a final event. It appears that UC_{25} and UC_{27} are included as a way of expressing a context in which the three use cases can interact, since

- the included use cases may occur days or weeks later, a scope of time much longer than that of any other use case in the specification; and

- the included use cases do not affect UC_{27}'s postconditions or help achieve the goals described in its overview;
- the pre- and postconditions of the three scenarios are not sufficient to express the context and results of any of the three.

This interaction could be expressed with pre- and postconditions for which

$$\text{Post}(UC_{25}) \Longrightarrow \text{Pre}(UC_{26})$$

$$\text{Post}(UC_{26}) \Longrightarrow \text{Post}(UC_{27})$$

but the conditions given for the use cases do not have those relationships. UC_{25}' relevant postcondition is

(Postcondition)

Status of Quote is set to "Approved".

but no use case's precondition mentions a quote status of "Approved"; UC_{26}'s relevant precondition is

(Precondition)

1. No unapproved BA Rule violations exist in the quote.

which is clearly related but implies that approval is per-violation, not per-quote. UC_{26}'s postconditions are

(Postcondition)

1. The Quote status is set to "Delivered".

(Postcondition)

2. A file exists for the Salesperson to deliver to the Customer.

neither of which is related to UC_{27}'s postconditions

(Postcondition)

1. Status of Quote has been changed to "Revised."

(Postcondition)

2. Euronet Main Screen is displayed on Salesperson's PC.

or in fact to the conditions of any other use case.

Note that the use of this heuristic drew attention to a potentially significant ambiguity in the specification, namely whether quote approval is done for the entire quote, or whether each violation is approved (or not) individually.

A scenario's events should achieve its goal and postcondition

EXAMPLE: The events of UC_{10} "No-Quote RFQ" do not distinguish the RFQ it creates from a Quote as created by UC_3 "Create Quote"; UC_{10}'s events are almost identical to the first events of UC_3, and there is no event that distinguishes the end of UC_{10} from the beginning of the remaining events of UC_3.

Cross-referencing (of actors, episodes, conditions, etc.) is valuable

EXAMPLE: A "uses" cross-reference between use cases showed that UC_{21} "Check BA Rules" is used only by UC_{24} "Submit Quote".

Scenario events describing what the GUI does should be rewritten to express the intents or reactions that lie behind them

The earlier goal-based analysis of the Euronet identified the need to express the use cases independently of the user interface [ACD+01]. We revised the events and conditions to eliminate references to the user interface.

EXAMPLE: UC_{29} "Create Order Without Quote" began with

1. Salesperson selects Orders function; Euronet displays Main Order Screen.
2. Salesperson : requests to create new Order; Euronet prompts the user for a customer identification.

which we converted into

1. Salesperson : requests to create new Order.
2. Euronet : prompts the user for a customer identification.

5.6 Similarity calculations on the EMS case study scenarios

Bode conducted an experiment on a subset of the EMS scenarios comparing the similarity that our similarity measure calculated with the perceived similarity that an analyst would see [Bod02]. In brief, she found that the similarity algorithm was effective in matching scenarios that she perceived as similar from reading them beforehand. The effectiveness was affected by several factors that she studied:

- The algorithm is more effective if synoymous terms in the scenarios are reconciled before the comparison. Synonymous terms are different words or phrases used with the same meaning.
- The algorithm is more effective on scenarios each of whose events performs a single atomic action, rather than if consecutive actions by the same actor are expressed as a single action.
- The algorithm was not effective at identifying potential episodes, and in the cases where it failed to identify them Bode found the reason was that the algorithm does not account for the sequence of events.
- The algorithm was more effective when more attributes were compared. There was no benefit to selecting attributes that might be more significant, in her results.

Bode also confirmed that automation of the similarity algorithm is essential, as the manual calculations required several days of careful work and introduced at least the 25 errors that she found in later checks. Her technical report provides full details.

5.7 Results from sequence-equivalence compression of scenario network diagrams

Compression using sequence-equivalence classes merges all the nodes for scenarios in the same sequence-equivalence class into one node whose incoming and outgoing arrows match those of all members of the class. Having done this, we can then merge all transitions

that come from the same node and go to the same node, respecting the distinction between sequence and ramification. The benefit is substantial for larger systems such as the EMS, whose compressed network diagram is shown in Figure 4.9. Large systems are the ones whose uncompressed diagrams are the most complex, so they show the greatest improvement. For comparison, the uncompressed diagram is given in Figure 4.5.

Table 5.6 summarizes the results of compressing several scenario networks. In all cases a reduction in the number of nodes and transitions was obtained, and for the larger scenario networks the reduction was very substantial. Using equivalence compression on scenario networks in this size range, which includes two medium-sized real systems, brings the level of complexity down to a range that is quite manageable.

	Uncompressed		Compressed	
	Nodes	Transitions	Nodes	Transitions
EMS-8	8	13	7	8
Elevator	9	54	3	3
EMS	38	514	12	19
Euronet	40	571	18	86

Table 5.6: Results of compressing four scenario networks

Chapter 6

Summary

> Peter Landin remarked long ago that the goal of his research was "to tell beautiful stories about computation." *John Reynolds*

The research in this dissertation uses the "industry-as-laboratory" approach that emphasizes requirements engineering and software engineering practice, and focuses on what is useful rather than simply what is possible [Pot93]. The author has been interested in the requirements and specification of software since the late 1970's, including work on the A-7E or Software Cost Reduction project at the Naval Research Laboratory from 1983 to 1985 [AFB+92]. The problem area that we specifically address here was identified as a result of our dissatisfaction with scenario collections that were claimed to be specifications, and as a result of work by other researchers on the practical use of scenarios and the problems and challenges that resulted. The insightful survey by Weidenhaupt *et al.* on scenario use in European software projects was a crystallizing factor [WPJH98]. We developed the foundations of the syntactic approach based on that survey combined with our experience in software development projects, and published early results [AABM99]. As this work was refined, we realized that there were additional practical benefits that could be obtained by widening the scope beyond individual scenarios. The work on scenario networks was the result. Fortuitously, a suitable case study with an industrial partner became possible at that time as part of another research project, and the EMS case study with BellSouth Telecommunications resulted. During this case study the concept of scenario networks was developed and expanded, and the syntactic techniques and approaches of ISA were put into

practice. The techniques and analyses that are most effective with scenario networks were developed and extended during the Elevator and Euronet case studies, and development of the automated tools supporting ISA and SNCA reached a point at which the tools could be useful in practice. The Elevator case study was an opportunity to examine what kinds of systems scenario networks and SNCA are and are not useful for. The Euronet case study provided an opportunity to investigate how to scale ISA, SNCA, and especially scenario networks as specifications to larger systems, as the original Euronet specification contained 52 defined and 12 undefined use cases and thus was a substantial collection. The use of Cartesian products and equivalence classes was matured at this time, and episodes were discovered to be an effective way of dealing with complexity in this case, reducing the number of scenarios in the scenario network to 36. The EMS and Euronet use cases also allowed a comparison of our results with the results of other methods, and provided a basis of comparison for evaluating how effective ISA, SNCA, and scenario network specification are in practice.

The remainder of this chapter provides a synopsis of earlier chapters, summarizes the contributions of this work, provides an overview of future work, and presents our conclusions.

6.1 Chapter synopsis

Chapter 1 introduced the challenges of using scenarios for specification and the component parts of the scenario management problem, and placed the general problem in its context in software and requirements engineering. Scenarios are widely used in the practice of requirements engineering, but their use involves some definite challenges. In addition, if more than a handful of scenarios are used, the problem of scenario management becomes prominent. This chapter also summarized Shaw's classifications of software engineering research, and placed this research in terms of her categories.

Chapter 2 presented a survey of related work in the area of software engineering and requirements engineering.

Chapter 3 introduced our syntactic approach to expressing and analyzing individ-

ual scenarios and collections of scenarios. The components of this approach are a formalized structure for expressing scenarios as collections of attribute-value pairs and event sequences as recursively defined lists of simple and compound events; the use of glossaries to define attribute values; the concept of episodes and a cluster of techniques for identifying and using them; and a syntactically-based similarity measure between scenarios. We call this approach ISA (Integrated Syntactic Analysis).

Chapter 4 presented our semantic approach to relating scenarios in a collection and integrating them into a single specification using scenario networks. Scenario networks provide practical benefits through the insights and process guidance obtained by constructing them, through the relationships that arise between scenarios in a scenario network, especially for determining and improving completeness and consistency; we call these techiques SNCA (Scenario Network Construction and Analysis). We also show how a scenario network forms a specification of the large-scale temporal relations between its component scenarios.

Chapter 5 presents three case studies validating SNCA and ISA, and one experiment validating the similarity measure component of ISA. The first case study was formative in nature and the latter two were summative.

6.2 Summary of contributions

The principal contributions of this work are the ISA and SNCA techniques and approaches for analysis and process guidance, and the scenario network approach for integrating a collection of scenarios. Research for these techniques and approaches included evaluation of existing scenario-based specification approaches, examination of analogous approaches in other areas of computer science and logic, a survey of problems found in the use of scenarios in general and their use specifically for specification, and a study of requirements and specifications in object-oriented methodologies. The syntactic formalization of scenarios and the concept of scenario networks were developed as a solution to several of the problems that were found.

The experiences described in this dissertation demonstrate that:

- It is possible to add an unobtrusive formal structure to scenarios that will provide a

basis for algorithmic support of the requirements engineering process.

- It is possible to relate the scenarios that describe a system into a scenario network that expresses the context in which each scenario occurs.
- There are relationships between the components of a scenario, and between scenarios that describe the same system, that are generally overlooked but that contain useful information.
- Automated analyses and operations that use the scenario structure and the scenario network produce results that are valuable for requirements engineering and which are in advance of what can be done manually.

The primary contributions of this dissertation are:

- the formal structure of scenarios as attribute-value pairs and event sequences.
- recursively-defined event sequences which can contain episodes, iterations, and alternations as well as simple events.
- the Integrated Syntactic Analyses and Scenario Network Construction and Analysis, which assist analysts in assessing and improving completeness, consistency, and other measures of the quality of a specification, and produce results quickly, sometimes within a matter of hours.
- a set of heuristics for using Scenario Network Construction and Analysis to validate and improve a scenario-based specification.
- novel scenario relationships, arising from the structure of scenario networks, which partition a system's scenarios into equivalence classes and aid in maintaining consistency and dependencies among scenarios during specification evolution.

6.3 Future work

This research lays the foundation for formally structuring scenarios and analyzing them based on the syntax of this structure, combining scenarios into a scenario network

and analyzing the resulting relationships, and making use of all these in evaluating and improving completeness and consistency, addressing the challenges of using scenarios for specification, and using them effectively in the face of the scenario management problem. There are several areas of future work remaining to be done.

Tool support for ISA and SNCA is not entirely complete at the date of this writing, and the support we have implemented so far has opened up some new possibilities for automated support we had not originally considered. Expansion of automated support, and integration of the separate tools for ISA and SNCA, are areas of future work for us.

We hypothesize that our syntactic similarity measure provides an indication of requirements coverage and specification completeness. Evaluation of this hypothesis requires more extensive automated support than has been available to date, as the similarity measure is too time-consuming to do by hand except for groups of scenarios that are too small for their coverage and completeness to be interesting. We have identified a scenario collection whose evolution has been recorded and which could form the basis of a case study on coverage, and our future work includes extending our tool support to provide the necessary calculations and applying it to this scenario collection.

We have demonstrated the scalability of our scenario network techniques to collections of 30 to 40 scenarios; these are collections of substantial size, but industrial collections range larger than this, and scalability of our techniques is a potential concern. Future work includes the application of ISA, SNCA, and scenario network specification to larger systems described by more scenarios.

One of the characteristics of a scenario is its concreteness level. Informally, a scenario's concreteness level can range from concrete to abstract. An extremely concrete scenario would have actors bound to specific individuals, actions bound to a specific situation, timings bound to particular moments, and in general more circumstance and no generality. An extremely abstract scenario would have actors specified only as abstract roles, general actions not specific to any one situation, relative timings and no tie to specific moments, and in general more abstraction and generality and less specific detail. Concreteness level has received little attention, but it has been noted that in practice it can be difficult to align the scenarios in a collection to a specific level of concreteness, and that

different levels of concreteness are needed for different uses of a scenario. A scenario to be used in generating test data might need to be highly concrete, yet a scenario that is clearly related to it might need to be much more abstract as part as the system specification. We believe that our formal scenario structure and our syntactic analysis approach may prove fruitful in this area of investigation.

Our syntactic structure for representing scenarios was chosen with the idea that each scenario represented using it would map straightforwardly back to the original prose narrative form of the scenario, or to a prose form that was close to the original. We have kept this desired property in mind during the course of our research, without ever taking time from other research to pursue it, and we have seen that the mapping is more difficult than we had originally hoped for. The difficulties include the need to obtain a consistent formal representation of all the scenarios, despite the potentially large variability in their original prose form, and the need to reorganize the boundaries between scenarios, so that one prose scenario may map piecewise onto many scenarios expressed in our form. However the potential for a useful, practical, and effective mapping is still there, and development of such mappings and techniques to support them is an area of future work.

6.4 Conclusions

Scenarios are informal in nature but need to be worked with as though they were more structured. We have shown that our attribute-value representation of scenarios leaves them in a form that is fairly close to prose, yet it allows analyses and operations on the syntactic structure. These include an automated similarity measure between scenarios, and the identification, extraction, and maintenance of episodes shared among several scenarios.

The context of a scenario must be known in order to understand it. We have shown that scenario networks are a means of determining, expressing, and preserving the context of scenarios in a collection. A scenario network also forms an integrated specification of an entire scenario collection, linking the behaviors described by individual scenarios into larger contexts and into relationships that unfold over a length of time longer than any one scenario. The relationships between scenarios that a scenario network represents are

effective in classifying and organizing scenarios, and are a means of preserving some of the desired qualities of a particular specification as the specification and its component scenarios evolve. The process of constructing a scenario network and the analyses possible on a completed scenario network indicate the degree of completeness, requirements coverage, and consistency of a collection of scenarios, and provide procedural guidance for improving all three measures.

The use of our approaches and techniques in the practice of requirements engineering with scenarios results in better specifications, produced more efficiently, that express requirements that are absent or left implicit in an ordinary collection of scenarios.

Appendix A

EMS case study data

This appendix provides details from the EMS case study that are too long to place in the main text but which may be of interest to readers.

$\mathbf{S_1}$.	Subscriber authentication
$\mathbf{S_2}$.	Subscriber authentication from an unsubscribed telephone
$\mathbf{S_3}$.	Subscriber authentication from some other subscriber's telephone
$\mathbf{S_4}$.	Subscriber changes his/her passcode
$\mathbf{S_5}$.	Subscriber configures his/her announcement
$\mathbf{S_6}$.	Subscriber sets up a group of phone numbers as a recipient
$\mathbf{S_7}$.	Subscriber checks for new messages
$\mathbf{S_8}$.	New message notification by stuttered dial tone
$\mathbf{S_9}$.	New message notification, by indicator
$\mathbf{S_{10}}$.	No new message notification, by indicator
$\mathbf{S_{11}}$.	Subscriber listens to a new or held message
$\mathbf{S_{12}}$.	Subscriber listens to an archived message
$\mathbf{S_{13}}$.	Subscriber skips to next message
$\mathbf{S_{14}}$.	Subscriber would skip to next message but has no more messages
$\mathbf{S_{15}}$.	Subscriber skips around in a message while listening to it
$\mathbf{S_{16}}$.	Subscriber listens to the time a message was received
$\mathbf{S_{17}}$.	Subscriber erases a message
$\mathbf{S_{18}}$.	Subscriber archives a message
$\mathbf{S_{19}}$.	Subscriber replies to a message
$\mathbf{S_{20}}$.	Subscriber forwards a message
$\mathbf{S_{21}}$.	Subscriber forwards a message with a preface
$\mathbf{S_{22}}$.	Subscriber records a message and sends it to someone
$\mathbf{S_{23}}$.	Subscriber doesn't take any action for a long time
$\mathbf{S_{24}}$.	Subscriber is prompted about old held messages
$\mathbf{S_{25}}$.	Subscriber is prompted about old archived messages
$\mathbf{S_{26}}$.	Caller calls subscriber and leaves a message
$\mathbf{S_{27}}$.	Caller reviews his/her message
$\mathbf{S_{28}}$.	Caller reviews and re-records his/her message
$\mathbf{S_{29}}$.	Caller distinguishes his/her message as urgent
$\mathbf{S_{30}}$.	Caller distinguishes his/her message as private
$\mathbf{S_{31}}$.	Caller decides he/she needs to speak to a receptionist
$\mathbf{S_{32}}$.	Caller doesn't want to listen to the subscriber's announcement

Table A.1: List of EMS scenarios (before case study)

$\mathbf{S_0}$.	EMS startup
$\mathbf{S_1}$.	EMS shutdown
$\mathbf{S_2}$.	Subscriber authentication
$\mathbf{S_3}$.	Subscriber authentication from unsubscribed telephone
$\mathbf{S_4}$.	Subscriber authentication from another subscriber's telephone
$\mathbf{S_5}$.	Subscriber changes his/her passcode
$\mathbf{S_6}$.	Subscriber configures his/her announcement
$\mathbf{S_7}$.	Subscriber sets up a group of phone numbers as a recipient
$\mathbf{S_8}$.	Subscriber checks for new messages
$\mathbf{S_9}$.	New message notification by stuttered dial tone
$\mathbf{S_{10}}$.	New message notification, by indicator
$\mathbf{S_{11}}$.	No new message notification, by indicator
$\mathbf{S_{12}}$.	Subscriber listens to a message
$\mathbf{S_{13}}$.	Subscriber has no more messages to listen to
$\mathbf{S_{14}}$.	Subscriber skips to next message
$\mathbf{S_{15}}$.	Subscriber would skip to next message but has no more messages
S_{16}.	(Absorbed into other scenarios as an episode)
$\mathbf{S_{17}}$.	Subscriber listens to the time a message was received
$\mathbf{S_{18}}$.	Subscriber erases a message
$\mathbf{S_{19}}$.	Subscriber erases the last message
$\mathbf{S_{20}}$.	Subscriber archives a message
$\mathbf{S_{21}}$.	Subscriber archives the last message
$\mathbf{S_{22}}$.	Subscriber forwards a message
$\mathbf{S_{23}}$.	Subscriber forwards a message with a preface
$\mathbf{S_{24}}$.	Subscriber replies to a message
$\mathbf{S_{25}}$.	Subscriber records a message and sends it to someone
$\mathbf{S_{26}}$.	Subscriber takes no action for a long time
$\mathbf{S_{27}}$.	Subscriber is prompted about old held messages
$\mathbf{S_{28}}$.	Subscriber is prompted about old archived messages
$\mathbf{S_{29}}$.	Subscriber disconnects from EMS
$\mathbf{S_{30}}$.	Caller calls subscriber and leaves a message
$\mathbf{S_{31}}$.	Caller reviews his/her message
$\mathbf{S_{32}}$.	Caller reviews and re-records his/her message
$\mathbf{S_{33}}$.	Caller distinguishes his/her message as urgent
$\mathbf{S_{34}}$.	Caller distinguishes his/her message as private
$\mathbf{S_{35}}$.	Caller decides he/she needs to speak to a receptionist
S_{36}.	(Absorbed into other scenarios as an episode)
$\mathbf{S_{37}}$.	Caller calls EMS directly and leaves a message
$\mathbf{S_{38}}$.	Caller takes no action for a long time
$\mathbf{S_{39}}$.	Caller disconnects from EMS

Table A.2: List of EMS scenarios (after case study)

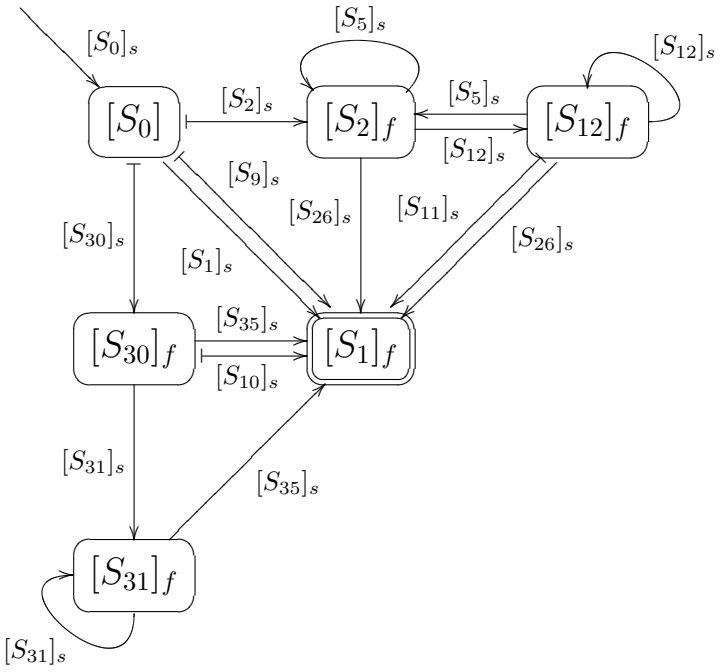


Figure A.1: EMS inverted scenario network diagram

Initial	S_0	
Terminal	S_1 S_9 S_{10} S_{11} S_{26} S_{29} S_{35} S_{38} S_{39}	
Scenario	Follow set	Ramification set
S_0	S_1	S_2 S_3 S_4 S_9 S_{30} S_{37}
S_1	$\emptyset$	$\emptyset$
S_2	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_3	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_4	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_5	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_6	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_7	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_8	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_9	$\emptyset$	$\emptyset$
S_{10}	$\emptyset$	$\emptyset$
S_{11}	$\emptyset$	$\emptyset$
S_{12}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	S_{11}
S_{13}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{14}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{15}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{17}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{18}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{19}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{20}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{21}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{22}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{23}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{24}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{25}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{26}	$\emptyset$	$\emptyset$
S_{27}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{28}	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17}–S_{29}	$\emptyset$
S_{29}	$\emptyset$	$\emptyset$
S_{30}	S_{31} S_{32} S_{33} S_{34} S_{35} S_{38} S_{39}	S_{10}
S_{31}	S_{31} S_{32} S_{33} S_{34} S_{35} S_{38} S_{39}	$\emptyset$
S_{32}	S_{31} S_{32} S_{33} S_{34} S_{35} S_{38} S_{39}	$\emptyset$
S_{33}	S_{31} S_{32} S_{33} S_{34} S_{35} S_{38} S_{39}	$\emptyset$
S_{34}	S_{31} S_{32} S_{33} S_{34} S_{35} S_{38} S_{39}	$\emptyset$
S_{35}	$\emptyset$	$\emptyset$
S_{37}	S_{31} S_{32} S_{33} S_{34} S_{35} S_{38} S_{39}	S_{10}
S_{38}	$\emptyset$	$\emptyset$
S_{39}	$\emptyset$	$\emptyset$

Table A.3: EMS scenario network table

Class	Members
$[S_0]_s$	S_0
$[S_1]_s$	S_1
$[S_2]_s$	S_2 S_3 S_4
$[S_5]_s$	S_5 S_6 S_7 S_8 S_{13} S_{14} S_{15} S_{17} S_{18} S_{19} S_{20} S_{21} S_{22} S_{23} S_{24} S_{25} S_{27} S_{28}
$[S_9]_s$	S_9
$[S_{10}]_s$	S_{10}
$[S_{11}]_s$	S_{11}
$[S_{12}]_s$	S_{12}
$[S_{26}]_s$	S_{26} S_{29}
$[S_{30}]_s$	S_{30} S_{37}
$[S_{31}]_s$	S_{31} S_{32} S_{33} S_{34}
$[S_{35}]_s$	S_{35} S_{38} S_{39}

Table A.4: EMS sequence-equivalence classes

Class	Members
$[S_0]_f$	S_0
$[S_1]_f$	S_1
$[S_2]_f$	S_2 S_3 S_4 S_9 S_{30} S_{37}
$[S_5]_f$	S_5 S_6 S_7 S_8 S_{12} S_{13} S_{14} S_{15} S_{17} S_{18} S_{19} S_{20} S_{21} S_{22} S_{23} S_{24} S_{25} S_{26} S_{27} S_{28} S_{29}
$[S_{10}]_f$	S_{10}
$[S_{11}]_f$	S_{11}
$[S_{31}]_f$	S_{31} S_{32} S_{33} S_{34} S_{35} S_{38} S_{39}

Table A.5: EMS follow-equivalence classes

Class	Members
$[S_0]_p$	S_0
$[S_1]_p$	S_1 S_9 S_{10} S_{11} S_{26} S_{29} S_{35} S_{38} S_{39}
$[S_2]_p$	S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_{13} S_{14} S_{15} S_{17} S_{18} S_{19} S_{20} S_{21} S_{22} S_{23} S_{24} S_{25} S_{27} S_{28}
$[S_{12}]_p$	S_{12}
$[S_{30}]_p$	S_{30} S_{37}
$[S_{31}]_p$	S_{31} S_{32} S_{33} S_{34}

Table A.6: EMS precede-equivalence classes

Class_{sub}	Classes_{fol}	Classes_{pre}
$[S_0]_s$	$[S_0]_f$	$[S_0]_p$
$[S_1]_s$	$[S_1]_f$	$[S_1]_p$
$[S_2]_s$	$[S_2]_f$	$[S_2]_p$
$[S_5]_s$	$[S_5]_f$	$[S_2]_p$
$[S_9]_s$	$[S_2]_f$	$[S_1]_p$
$[S_{10}]_s$	$[S_{10}]_f$	$[S_1]_p$
$[S_{11}]_s$	$[S_{11}]_f$	$[S_1]_p$
$[S_{12}]_s$	$[S_5]_f$	$[S_{12}]_p$
$[S_{26}]_s$	$[S_5]_f$	$[S_1]_p$
$[S_{30}]_s$	$[S_2]_f$	$[S_{30}]_p$
$[S_{31}]_s$	$[S_{31}]_f$	$[S_{31}]_p$
$[S_{35}]_s$	$[S_{31}]_f$	$[S_1]_p$

Table A.7: EMS follow- and precede-equivalence classes corresponding to each sequence-equivalence class

Class	Subset equivalence classes
$[S_0]_f$	$[S_0]_s$
$[S_1]_f$	$[S_1]_s$
$[S_2]_f$	$[S_2]_s$ $[S_9]_s$ $[S_{30}]_s$
$[S_5]_f$	$[S_5]_s$ $[S_{12}]_s$ $[S_{26}]_s$
$[S_{10}]_f$	$[S_{10}]_s$
$[S_{11}]_f$	$[S_{11}]_s$
$[S_{31}]_f$	$[S_{31}]_s$ $[S_{35}]_s$

Table A.8: EMS sequence-equivalence classes corresponding to each follow-equivalence class

Class	Subset equivalence classes
$[S_0]_p$	$[S_0]_s$
$[S_1]_p$	$[S_1]_s$ $[S_9]_s$ $[S_{10}]_s$ $[S_{11}]_s$ $[S_{26}]_s$ $[S_{35}]_s$
$[S_2]_p$	$[S_2]_s$ $[S_5]_s$
$[S_{12}]_p$	$[S_{12}]_s$
$[S_{30}]_p$	$[S_{30}]_s$
$[S_{31}]_p$	$[S_{31}]_s$

Table A.9: EMS sequence-equivalence classes corresponding to each precede-equivalence class

Initial	S_0	
Terminal	S_1 S_9 S_{10} S_{11} $[S_{26}]$ $[S_{35}]$	
Scenario	Follow set	Ramification set
S_0	S_1	$[S_2]$ S_9 $[S_{30}]$
S_1	$\emptyset$	$\emptyset$
$[S_2]$	$[S_5]$ S_{12} $[S_{26}]$	$\emptyset$
$[S_5]$	$[S_5]$ S_{12} $[S_{26}]$	$\emptyset$
S_9	$\emptyset$	$\emptyset$
S_{10}	$\emptyset$	$\emptyset$
S_{11}	$\emptyset$	$\emptyset$
S_{12}	$[S_5]$ S_{12} $[S_{26}]$	S_{11}
$[S_{26}]$	$\emptyset$	$\emptyset$
$[S_{30}]$	$[S_{31}]$ $[S_{35}]$	S_{10}
$[S_{31}]$	$[S_{31}]$ $[S_{35}]$	$\emptyset$
$[S_{35}]$	$\emptyset$	$\emptyset$

Table A.10: EMS equivalence-compressed scenario network table

Initial	$[S_0]_s$
Nodes	Transitions:Nodes
$[S_0]_f$	$[S_1]_s \rightarrow [S_1]_f$ $[S_2]_s \rightarrow [S_2]_f$ $[S_9]_s \rightarrow [S_2]_f$ $[S_{30}]_s \rightarrow [S_2]_f$
$[S_1]_f$	$\emptyset$
$[S_2]_f$	$[S_5]_s \rightarrow [S_5]_f$ $[S_{12}]_s \rightarrow [S_5]_f$ $[S_{26}]_s \rightarrow [S_5]_f$
$[S_5]_f$	$[S_5]_s \rightarrow [S_5]_f$ $[S_{12}]_s \rightarrow [S_5]_f$ $[S_{26}]_s \rightarrow [S_5]_f$
$[S_{10}]_f$	$\emptyset$
$[S_{11}]_f$	$\emptyset$
$[S_{31}]_f$	$[S_{31}]_s \rightarrow [S_{31}]_f$ $[S_{35}]_s \rightarrow [S_{31}]_f$

Table A.11: EMS inverted scenario network table

Appendix B

Elevator case study data

This appendix provides details from the EMS case study that are too long to place in the main text but which may be of interest to readers.

	(Scenario S_1 was deleted.)
S_2.	Initialization.
S_3.	Call from floor.
S_4.	Call from within car.
S_5.	Next floor heading down.
S_6.	Next floor heading up.
S_7.	Change direction from up to down.
S_8.	Change direction from down to up.
S_9.	Idle.
S_{10}.	Shutdown.

Table B.1: List of Elevator Problem scenarios

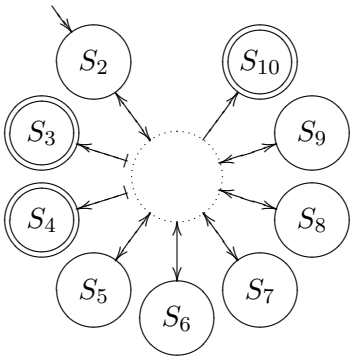


Figure B.1: Elevator Problem scenario network diagram

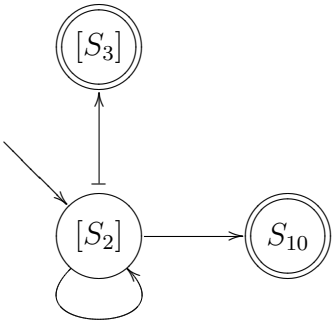


Figure B.2: Elevator Problem compressed scenario network diagram

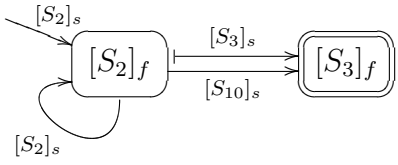


Figure B.3: Elevator Problem inverted scenario network diagram

Initial	S_2	
Terminal	S_3 S_4 S_{10}	
Scenario	Follow set	Ramification set
S_2	S_2 S_5 S_6 S_7 S_8 S_9 S_{10}	S_3 S_4
S_3	$\emptyset$	$\emptyset$
S_4	$\emptyset$	$\emptyset$
S_5	S_2 S_5 S_6 S_7 S_8 S_9 S_{10}	S_3 S_4
S_6	S_2 S_5 S_6 S_7 S_8 S_9 S_{10}	S_3 S_4
S_7	S_2 S_5 S_6 S_7 S_8 S_9 S_{10}	S_3 S_4
S_8	S_2 S_5 S_6 S_7 S_8 S_9 S_{10}	S_3 S_4
S_9	S_2 S_5 S_6 S_7 S_8 S_9 S_{10}	S_3 S_4
S_{10}	$\emptyset$	$\emptyset$

Table B.2: Elevator Problem scenario network table

Class	Members
$[S_2]_s$	S_2 S_5 S_6 S_7 S_8 S_9
$[S_3]_s$	S_3 S_4
$[S_{10}]_s$	S_{10}

Table B.3: Elevator Problem sequence-equivalence classes

Class	Members
$[S_2]_f$	S_2 S_5 S_6 S_7 S_8 S_9 S_{10}
$[S_3]_f$	S_3 S_4

Table B.4: Elevator Problem follow-equivalence classes

Class	Members
$[S_2]_p$	S_2 S_5 S_6 S_7 S_8 S_9
$[S_3]_p$	S_3 S_4 S_{10}

Table B.5: Elevator Problem precede-equivalence classes

Class_{sub}	Classes_{fol}	Classes_{pre}
$[S_2]_s$	$[S_2]_f$	$[S_2]_p$
$[S_3]_s$	$[S_3]_f$	$[S_3]_p$
$[S_{10}]_s$	$[S_2]_f$	$[S_3]_p$

Table B.6: Elevator Problem follow- and precede-equivalence classes corresponding to each sequence-equivalence class

Class	Subset equivalence classes
$[S_2]_f$	$[S_2]_s$ $[S_{10}]_s$
$[S_3]_f$	$[S_3]_s$

Table B.7: Elevator Problem sequence-equivalence classes corresponding to each follow-equivalence class

Class	Subset equivalence classes
$[S_2]_p$	$[S_2]_s$
$[S_3]_p$	$[S_3]_s$ $[S_{10}]_s$

Table B.8: Elevator Problem sequence-equivalence classes corresponding to each precede-equivalence class

Initial	$[S_2]$	
Terminal	$[S_3]$ S_{10}	
Scenario	Follow set	Ramification set
$[S_2]$	$[S_2]$ S_{10}	$[S_3]$
$[S_3]$	$\emptyset$	$\emptyset$
S_{10}	$\emptyset$	$\emptyset$

Table B.9: Elevator Problem equivalence-compressed scenario network table

Initial	$[S_2]_s$
Nodes	Transitions:Nodes
$[S_2]_f$	$[S_2]_s \rightarrow [S_2]_f$ $[S_3]_s \rightarrow [S_3]_f$ $[S_{10}]_s \rightarrow [S_2]_f$
$[S_3]_f$	$\emptyset$

Table B.10: Elevator Problem inverted scenario network table

Appendix C

Euronet case study data

Changes from the original list of use cases during ISA

All the use cases that were referenced by other use cases were made into episodes. For those that were useful as independent scenarios, we created a scenario whose only event was the use case.

We examined UC_{36} "Allocate Order Items" and decided that it was unclear whether its events allocated a single item or a group of items. We decided to rename it in the singular ("Allocate Order Item"), change its events to clearly allocate one item only, and used it in place of the undefined Ep_a "Allocate Item".

UC_{44} "Change Language" had no function except as part of the log-on process, so its events were absorbed into S_1 "Log On".

Ep_{45} "Enter Date with Popup" is problematic; we know enough from reading its original specification to see that it must be an episode, but we don't know enough about the other scenarios to say where it is used.

Ep_e "Close Order" was undefined in the original specification; we found an analogous sequence of events in UC_{30} "Create Order From Quote" and based Ep_e's definition on that.

We replaced all references to the undefined Ep_i "Edit Quote Header" with references to UC_{12} "Edit Header", because UC_{12}'s events described the editing of a quote header.

We renamed Ep_{12} to "Edit (Quote) Header" to make its effect clearer.

Ep_l "Retrieve Quote" was referenced but not defined in the original specification. We decided by considering its name and its uses that it was close enough to either Ep_d "Choose Quote" or S_8 "Retrieve Similar Quotes" to not be needed as a separate episode. This was especially true since in the original specification it was used in no event; it appeared only in a precondition of UC_{50} "Get Competitive Feedback".

Ep_A "Create Customer" was created to cover the gap left at each use of Ep_{46} "Choose Customer", where a particular customer was selected but there was no reason that the selected customer had to exist before the selection was made.

Ep_E "Retrieve Similar Quotes" was identified as the common portion of the nearly identical UC_8 "Retrieve Similar Quotes" and UC_9 "Send Duplication Notice".

Four "Save/Close" episodes Ep_{F1} through Ep_{F4} were identified, identical except for what is being closed: an item, order, quote, or quote package. The events of these episodes are based upon the original UC_{35} "Save Order".

Ep_G "Submit Quote" was identified as the common portion of the nearly identical UC_{24} "Submit Quote" and UC_{27} "Revise Quote".

Changes to the set of scenarios and episodes during semantic analysis

Several scenarios were added as placeholders for required behavior that we do not know. In each case we do not know how Euronet should accomplish the scenario's goal or ensure its postcondition; we only know that Euronet will have to do that.

The lack of an initial and final scenario was remedied by adding S_0 "Euronet Startup" and S_{99} "Euronet Shutdown".

S_{53} "Transmit Order" was added because S_{42} "Cancel Order" has a precondition requiring the order to have been transmitted, but no existing scenario transmitted an order.

S_{54} "Create New Customer" was added because the precondition of S_{29} "Create Order Without Quote" requires a customer, but not necessarily a customer of long standing.

S_{55} "Select Item Build Order" was added because S_{37} "Acknowledge Item Build

Order" and S_{38} "Reject Item Build Order" both need to have an Item Build Order selected when they begin, and no other scenario selects one. We chose to create an additional scenario, rather than create an episode shared by S_{37} and S_{38} which would select a build order as part of those scenarios, because the selection of the build order seemed to be independent of what was done with it later; and also by analogy to the scenarios that select a quote or order.

Numerous changes were made within individual scenarios to event sequences and pre- and postconditions. Those changes were too numerous to include here.

1.	User Log On	27.	Revise Quote
2.	Retrieve Existing Quote	28.	Get Quote History
3.	Create Quote	29.	Create Order Without Quote
4.	Create Header	30.	Create Order From Quote
5.	Get Header General Information	31.	Retrieve Existing Order
6.	Get Header Terms	32.	Choose Order
7.	Get Header Notes	33.	Edit Order
8.	Retrieve Similar Quotes	34.	Edit Order Header
9.	Send Duplication Notice	35.	Save Order
10.	No-Quote RFQ	36.	Allocate Order Items
11.	Edit Quote	37.	Acknowledge Item Build Order
12.	Edit Header	38.	Reject Item Build Order
13.	Retrieve Similar Items	39.	Reallocate Order Item
14.	Add New Item	40.	Acknowledge Customer Order
15.	Get Ratings	41.	Cancel Item Build Order
16.	Add Standard Item	42.	Cancel Order
17.	Add Non-Engineered Item	43.	Report Euronet Bug
18.	Add Engineered Item	44.	Change Language
19.	Add Special Item	45.	Enter Date With Popup
20.	Add Alternate Item	46.	Choose Customer
21.	Check BA Rules	47.	Get Ship-To Address
22.	Edit Item	48.	Get Address Information
23.	Save Quote	49.	Add Item Note
24.	Submit Quote	50.	Get Competitive Feedback
25.	Approve Quote	51.	Get Capacity Utilization
26.	Assemble Quote Package	52.	Create Reports

The use cases below were referenced but not defined; we assigned the identifying letters:

a.	Allocate Item.	g.	Create New Item.
b.	Choose Approver.	h.	Edit Existing Item.
c.	Choose End User.	i.	Edit Quote Header.
d.	Choose Quote.	j.	Get Adders.
e.	Close Order.	k.	Get Item Inventory.
f.	Close Quote.	l.	Retrieve Quote.

Table C.1: List of Euronet use cases from original specification

S_1.	User Log On
S_2.	Retrieve Existing Quote
S_3.	Create Quote
S_8.	Retrieve Similar Quotes
S_{10}.	No-Quote RFQ
S_{11}.	Edit Quote
S_{13}.	Retrieve Similar Item
S_{20}.	Add Alternate Item
S_{23}.	Save Quote
S_{24}.	Submit Quote
S_{25}.	Approve Quote
S_{26}.	Assemble Quote Package
S_{27}.	Revise Quote
S_{28}.	Get Quote History
S_{29}.	Create Order Without Quote
S_{30}.	Create Order From Quote
S_{31}.	Retrieve Existing Order
S_{32}.	Choose Order
S_{33}.	Edit Order
S_{35}.	Save Order
S_{37}.	Acknowledge Item Build Order
S_{38}.	Reject Item Build Order
S_{39}.	Reallocate Order Item
S_{40}.	Acknowledge Customer Order
S_{41}.	Cancel Item Build Order
S_{42}.	Cancel Order
S_{43}.	Report Euronet Bug
S_{45}.	Enter Date With Popup
S_{48}.	Get Address Information
S_{49}.	Add Item Note
S_{50}.	Get Competitive Feedback
S_{51}.	Get Capacity Utilization
S_{52}.	Create Reports
New:	
S_{9a}.	Find Duplicate Quotes
S_{9b}.	Send Duplication Notice
S_{53}.	Transmit Order
S_{54}.	Create New Customer
S_{55}.	Select Item Build Order
S_{98}.	User Log Off
S_0.	Euronet Startup
S_{99}.	Euronet Shutdown

Table C.2: List of Euronet scenarios, after our analysis

Defined in [ABB99]:		*Referenced in [ABB99] but not defined:*	
Ep_2	Retrieve Existing Quote Episode	Ep_b	Choose Approver
Ep_3	Create Quote Episode	Ep_c	Choose End User
Ep_4	Create (Quote) Header	Ep_d	Choose Quote
Ep_5	Get Header General Information	Ep_e	Close Order
Ep_6	Get Header Terms	Ep_f	Close Quote
Ep_7	Get Header Notes	Ep_g	Create New Item
Ep_8	Retrieve Similar Quotes Episode	Ep_h	Edit Existing Item
Ep_{11}	Edit Quote Episode	Ep_j	Get Adders
Ep_{12}	Edit (Quote) Header	Ep_k	Get Item Inventory
Ep_{14}	Add New Item	*New:*	
Ep_{15}	Get Ratings	Ep_A	Create Customer
Ep_{16}	Add Standard Item	Ep_{C1}	Edit Order Information
Ep_{17}	Add Non-Engineered Item	Ep_{C2}	Edit Quote Information
Ep_{18}	Add Engineered Item	Ep_{D1}	No-Quote A Quote
Ep_{19}	Add Special Item	Ep_{D2}	No-Quote An RFQ
Ep_{21}	Check BA Rules	Ep_{F1}	Save/Close Item
Ep_{22}	Edit Item	Ep_{F2}	Save/Close Order
Ep_{24}	Submit Quote Episode	Ep_{F3}	Save/Close Quote
Ep_{31}	Retrieve Existing Order Episode	Ep_{F4}	Save/Close Quote Package
Ep_{32}	Choose Order Episode	Ep_{33}	Edit Order Episode
Ep_{34}	Edit Order Header		
Ep_{36}	Allocate Order Item		
Ep_{41}	Cancel Item Build Order Episode		
Ep_{46}	Choose Customer		
Ep_{47}	Get Ship-To Address		
Ep_{51}	Get Capacity Utilization Episode		

Table C.3: List of Euronet episodes, after our analysis

Initial	S_0	
Terminal	S_{43} S_{45} S_{98} S_{99}	
Scenario	Follow set	Ramification set
S_0	S_{99}	S_1
S_1	S_2 S_3 S_8 S_{9a} S_{10} S_{30} S_{31} S_{32} S_{43} S_{45} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	$\emptyset$
S_2	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30} S_{31} S_{32} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	S_{43} S_{45}
S_3	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30} S_{31} S_{32} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	S_{43} S_{45}
S_8	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30} S_{31} S_{32} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	S_{43} S_{45}
S_{9a}	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30} S_{31} S_{32} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	S_{43} S_{45}
S_{9b}	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30} S_{31} S_{32} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	S_{43} S_{45}
S_{10}	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30} S_{31} S_{32} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	S_{43} S_{45}
S_{11}	S_{13} S_{23}	S_{43} S_{45}
S_{13}	S_{20} S_{23} S_{41} S_{49} S_{50}	S_{43} S_{45}
S_{20}	S_{20} S_{23} S_{41} S_{49} S_{50}	S_{43} S_{45}
S_{23}	S_{13} S_{20} S_{23} S_{41} S_{49} S_{50}	S_{43} S_{45}
S_{24}	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30} S_{31} S_{32} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	S_{43} S_{45}
S_{25}	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30} S_{31} S_{32} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	S_{43} S_{45}
S_{26}	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30} S_{31} S_{32} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	S_{43} S_{45}
S_{27}	S_{13} S_{23}	S_{43} S_{45}
S_{28}	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30} S_{31} S_{32} S_{48} S_{51} S_{52} S_{54} S_{55} S_{98}	S_{43} S_{45}
S_{30}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{53} S_{55} S_{98}	S_{43} S_{45}
S_{31}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{53} S_{55} S_{98}	S_{43} S_{45}
S_{32}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{53} S_{55} S_{98}	S_{43} S_{45}
S_{33}	S_{35}	S_{43} S_{45}
S_{35}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{53} S_{55} S_{98}	S_{43} S_{45}
S_{37}	S_2 S_3 S_8 S_{9a} S_{10} S_{30} S_{31} S_{32} S_{37} S_{38} S_{39} S_{55} S_{98}	S_{43} S_{45}
S_{38}	S_2 S_3 S_8 S_{9a} S_{10} S_{30} S_{31} S_{32} S_{37} S_{38} S_{39} S_{55} S_{98}	S_{43} S_{45}
S_{39}	S_2 S_3 S_8 S_{9a} S_{10} S_{30} S_{31} S_{32} S_{37} S_{38} S_{39} S_{55} S_{98}	S_{43} S_{45}
S_{40}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{53} S_{55} S_{98}	S_{43} S_{45}
S_{41}	S_{20} S_{23} S_{41} S_{49} S_{50}	S_{43} S_{45}
S_{42}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{53} S_{55} S_{98}	S_{43} S_{45}
S_{43}	$\emptyset$	$\emptyset$
S_{45}	$\emptyset$	$\emptyset$
S_{48}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{48} S_{51}–S_{55} S_{98}	S_{43} S_{45}
S_{49}	S_{20} S_{23} S_{41} S_{49} S_{50}	S_{43} S_{45}
S_{50}	S_{20} S_{23} S_{41} S_{49} S_{50}	S_{43} S_{45}
S_{51}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{48} S_{51}–S_{55} S_{98}	S_{43} S_{45}
S_{52}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{48} S_{51}–S_{55} S_{98}	S_{43} S_{45}
S_{53}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{53} S_{55} S_{98}	S_{43} S_{45}
S_{54}	S_2 S_3 S_8 S_{9a} S_{10} S_{30}–S_{33} S_{40} S_{42} S_{48} S_{51}–S_{55} S_{98}	S_{43} S_{45}
S_{55}	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{11} S_{24}–S_{28} S_{30}–S_{33} S_{37}–S_{40} S_{42} S_{43} S_{45} S_{48} S_{51}–S_{55} S_{98}	S_{43} S_{45}
S_{98}	$\emptyset$	$\emptyset$
S_{99}	$\emptyset$	$\emptyset$

Table C.4: Euronet scenario network table

Class	Members
$[S_0]_s$	S_0
$[S_1]_s$	S_1
$[S_2]_s$	$S_2\ S_3\ S_8\ S_{9a}\ S_{10}$
$[S_{9b}]_s$	$S_{9b}\ S_{24}\ S_{25}\ S_{26}\ S_{28}$
$[S_{11}]_s$	$S_{11}\ S_{27}$
$[S_{13}]_s$	S_{13}
$[S_{20}]_s$	$S_{20}\ S_{41}\ S_{49}\ S_{50}$
$[S_{23}]_s$	S_{23}
$[S_{30}]_s$	$S_{30}\ S_{31}\ S_{32}$
$[S_{33}]_s$	S_{33}
$[S_{35}]_s$	S_{35}
$[S_{37}]_s$	$S_{37}\ S_{38}\ S_{39}$
$[S_{40}]_s$	$S_{40}\ S_{42}\ S_{53}$
$[S_{43}]_s$	$S_{43}\ S_{45}$
$[S_{48}]_s$	$S_{48}\ S_{51}\ S_{52}\ S_{54}$
$[S_{55}]_s$	S_{55}
$[S_{98}]_s$	S_{98}
$[S_{99}]_s$	S_{99}

Table C.5: Euronet sequence-equivalence classes

Class	Members
$[S_0]_f$	S_0
$[S_1]_f$	S_1
$[S_2]_f$	$S_2\ S_3\ S_8\ S_{9a}\ S_{10}\ S_{30}\ S_{31}\ S_{32}\ S_{55}\ S_{98}$
$[S_{9b}]_f$	$S_{9b}\ S_{11}\ S_{24}\ S_{25}\ S_{26}\ S_{27}\ S_{28}$
$[S_{13}]_f$	S_{13}
$[S_{20}]_f$	$S_{20}\ S_{41}\ S_{49}\ S_{50}$
$[S_{23}]_f$	S_{23}
$[S_{33}]_f$	$S_{33}\ S_{40}\ S_{42}\ S_{53}$
$[S_{35}]_f$	S_{35}
$[S_{37}]_f$	$S_{37}\ S_{38}\ S_{39}$
$[S_{43}]_f$	$S_{43}\ S_{45}$
$[S_{48}]_f$	$S_{48}\ S_{51}\ S_{52}\ S_{54}$
$[S_{99}]_f$	S_{99}

Table C.6: Euronet follow-equivalence classes

Class	Members
$[S_0]_p$	S_0
$[S_1]_p$	S_1
$[S_2]_p$	S_2 S_3 S_8 S_{9a} S_{9b} S_{10} S_{24} S_{25} S_{26} S_{28}
$[S_{11}]_p$	S_{11} S_{27}
$[S_{13}]_p$	S_{13} S_{20} S_{41} S_{49} S_{50}
$[S_{23}]_p$	S_{23}
$[S_{30}]_p$	S_{30} S_{31} S_{32} S_{35} S_{40} S_{42} S_{53}
$[S_{33}]_p$	S_{33}
$[S_{37}]_p$	S_{37} S_{38} S_{39}
$[S_{43}]_p$	S_{43} S_{45} S_{98} S_{99}
$[S_{48}]_p$	S_{48} S_{51} S_{52} S_{54}
$[S_{55}]_p$	S_{55}

Table C.7: Euronet precede-equivalence classes

Class_{sub}	Classes_{fol}	Classes_{pre}
$[S_0]_s$	$[S_0]_f$	$[S_0]_p$
$[S_1]_s$	$[S_1]_f$	$[S_1]_p$
$[S_2]_s$	$[S_2]_f$	$[S_2]_p$
$[S_{9b}]_s$	$[S_{9b}]_f$	$[S_2]_p$
$[S_{11}]_s$	$[S_{9b}]_f$	$[S_{11}]_p$
$[S_{13}]_s$	$[S_{13}]_f$	$[S_{13}]_p$
$[S_{20}]_s$	$[S_{20}]_f$	$[S_{13}]_p$
$[S_{23}]_s$	$[S_{23}]_f$	$[S_{23}]_p$
$[S_{30}]_s$	$[S_2]_f$	$[S_{30}]_p$
$[S_{33}]_s$	$[S_{33}]_f$	$[S_{33}]_p$
$[S_{35}]_s$	$[S_{35}]_f$	$[S_{30}]_p$
$[S_{37}]_s$	$[S_{37}]_f$	$[S_{37}]_p$
$[S_{40}]_s$	$[S_{33}]_f$	$[S_{30}]_p$
$[S_{43}]_s$	$[S_{43}]_f$	$[S_{43}]_p$
$[S_{48}]_s$	$[S_{48}]_f$	$[S_{48}]_p$
$[S_{55}]_s$	$[S_2]_f$	$[S_{55}]_p$
$[S_{98}]_s$	$[S_2]_f$	$[S_{43}]_p$
$[S_{99}]_s$	$[S_{99}]_f$	$[S_{43}]_p$

Table C.8: Euronet follow- and precede-equivalence classes corresponding to each sequence-equivalence class

Class	Subset equivalence classes
$[S_0]_f$	$[S_0]_s$
$[S_1]_f$	$[S_1]_s$
$[S_2]_f$	$[S_2]_s$ $[S_{30}]_s$ $[S_{55}]_s$ $[S_{98}]_s$
$[S_{9b}]_f$	$[S_{9b}]_s$ $[S_{11}]_s$
$[S_{13}]_f$	$[S_{13}]_s$
$[S_{20}]_f$	$[S_{20}]_s$
$[S_{23}]_f$	$[S_{23}]_s$
$[S_{33}]_f$	$[S_{33}]_s$ $[S_{40}]_s$
$[S_{35}]_f$	$[S_{35}]_s$
$[S_{37}]_f$	$[S_{37}]_s$
$[S_{43}]_f$	$[S_{43}]_s$
$[S_{48}]_f$	$[S_{48}]_s$
$[S_{99}]_f$	$[S_{99}]_s$

Table C.9: Euronet sequence-equivalence classes corresponding to each follow-equivalence class

Class	Subset equivalence classes
$[S_0]_p$	$[S_0]_s$
$[S_1]_p$	$[S_1]_s$
$[S_2]_p$	$[S_2]_s$ $[S_{9b}]_s$
$[S_{11}]_p$	$[S_{11}]_s$
$[S_{13}]_p$	$[S_{13}]_s$ $[S_{20}]_s$
$[S_{23}]_p$	$[S_{23}]_s$
$[S_{30}]_p$	$[S_{30}]_s$ $[S_{35}]_s$ $[S_{40}]_s$
$[S_{33}]_p$	$[S_{33}]_s$
$[S_{37}]_p$	$[S_{37}]_s$
$[S_{43}]_p$	$[S_{43}]_s$ $[S_{98}]_s$ $[S_{99}]_s$
$[S_{48}]_p$	$[S_{48}]_s$
$[S_{55}]_p$	$[S_{55}]_s$

Table C.10: Euronet sequence-equivalence classes corresponding to each precede-equivalence class

Initial	S_0	
Terminal	$[S_{43}]$ S_{98} S_{99}	
Scenario	Follow set	Ramification set
S_0	S_{99}	S_1
S_1	$[S_2]$ $[S_{30}]$ $[S_{43}]$ $[S_{48}]$ S_{55} S_{98}	$\emptyset$
$[S_2]$	$[S_2]$ $[S_{9b}]$ $[S_{11}]$ $[S_{30}]$ $[S_{48}]$ S_{55} S_{98}	$[S_{43}]$
$[S_{9b}]$	$[S_2]$ $[S_{9b}]$ $[S_{11}]$ $[S_{30}]$ $[S_{48}]$ S_{55} S_{98}	$[S_{43}]$
$[S_{11}]$	S_{13} S_{23}	$[S_{43}]$
S_{13}	$[S_{20}]$ S_{23}	$[S_{43}]$
$[S_{20}]$	$[S_{20}]$ S_{23}	$[S_{43}]$
S_{23}	S_{13} $[S_{20}]$ S_{23}	$[S_{43}]$
$[S_{30}]$	$[S_2]$ $[S_{30}]$ S_{33} $[S_{40}]$ S_{55} S_{98}	$[S_{43}]$
S_{33}	S_{35}	$[S_{43}]$
S_{35}	$[S_2]$ $[S_{30}]$ S_{33} $[S_{40}]$ S_{55} S_{98}	$[S_{43}]$
$[S_{37}]$	$[S_2]$ $[S_{30}]$ $[S_{37}]$ S_{55} S_{98}	$[S_{43}]$
$[S_{40}]$	$[S_2]$ $[S_{30}]$ S_{33} $[S_{40}]$ S_{55} S_{98}	$[S_{43}]$
$[S_{43}]$	$\emptyset$	$\emptyset$
$[S_{48}]$	$[S_2]$ $[S_{30}]$ S_{33} $[S_{40}]$ $[S_{48}]$ S_{55} S_{98}	$[S_{43}]$
S_{55}	$[S_2]$ $[S_{9b}]$ $[S_{11}]$ $[S_{30}]$ S_{33} $[S_{37}]$ $[S_{40}]$ $[S_{43}]$ $[S_{48}]$ S_{55} S_{98}	$[S_{43}]$
S_{98}	$\emptyset$	$\emptyset$
S_{99}	$\emptyset$	$\emptyset$

Table C.11: Euronet equivalence-compressed scenario network table

Initial	$[S_0]_s$
Nodes	Transitions:Nodes
$[S_0]_f$	$[S_1]_s \rightarrow [S_1]_f$ $[S_{99}]_s \rightarrow [S_{99}]_f$
$[S_1]_f$	$[S_2]_s \rightarrow [S_2]_f$ $[S_{30}]_s \rightarrow [S_2]_f$ $[S_{43}]_s \rightarrow [S_{43}]_f$ $[S_{48}]_s \rightarrow [S_{48}]_f$ $[S_{55}]_s \rightarrow [S_2]_f$ $[S_{98}]_s \rightarrow [S_2]_f$
$[S_2]_f$	$[S_2]_s \rightarrow [S_2]_f$ $[S_{9b}]_s \rightarrow [S_{9b}]_f$ $[S_{11}]_s \rightarrow [S_{9b}]_f$ $[S_{30}]_s \rightarrow [S_2]_f$ $[S_{43}]_s \rightarrow [S_{43}]_f$ $[S_{48}]_s \rightarrow [S_{48}]_f$ $[S_{55}]_s \rightarrow [S_2]_f$ $[S_{98}]_s \rightarrow [S_2]_f$
$[S_{9b}]_f$	$[S_2]_s \rightarrow [S_2]_f$ $[S_{9b}]_s \rightarrow [S_{9b}]_f$ $[S_{11}]_s \rightarrow [S_{9b}]_f$ $[S_{30}]_s \rightarrow [S_2]_f$ $[S_{43}]_s \rightarrow [S_{43}]_f$ $[S_{48}]_s \rightarrow [S_{48}]_f$ $[S_{55}]_s \rightarrow [S_2]_f$ $[S_{98}]_s \rightarrow [S_2]_f$
$[S_{13}]_f$	$[S_{20}]_s \rightarrow [S_{20}]_f$ $[S_{23}]_s \rightarrow [S_{23}]_f$ $[S_{43}]_s \rightarrow [S_{43}]_f$
$[S_{20}]_f$	$[S_{20}]_s \rightarrow [S_{20}]_f$ $[S_{23}]_s \rightarrow [S_{23}]_f$ $[S_{43}]_s \rightarrow [S_{43}]_f$
$[S_{23}]_f$	$[S_{13}]_s \rightarrow [S_{13}]_f$ $[S_{20}]_s \rightarrow [S_{20}]_f$ $[S_{23}]_s \rightarrow [S_{23}]_f$ $[S_{43}]_s \rightarrow [S_{43}]_f$
$[S_{33}]_f$	$[S_{35}]_s \rightarrow [S_{35}]_f$ $[S_{43}]_s \rightarrow [S_{43}]_f$
$[S_{35}]_f$	$[S_2]_s \rightarrow [S_2]_f$ $[S_{30}]_s \rightarrow [S_2]_f$ $[S_{33}]_s \rightarrow [S_{33}]_f$ $[S_{40}]_s \rightarrow [S_{33}]_f$ $[S_{43}]_s \rightarrow [S_{43}]_f$ $[S_{55}]_s \rightarrow [S_2]_f$ $[S_{98}]_s \rightarrow [S_2]_f$
$[S_{37}]_f$	$[S_2]_s \rightarrow [S_2]_f$ $[S_{30}]_s \rightarrow [S_2]_f$ $[S_{37}]_s \rightarrow [S_{37}]_f$ $[S_{43}]_s \rightarrow [S_{43}]_f$ $[S_{55}]_s \rightarrow [S_2]_f$ $[S_{98}]_s \rightarrow [S_2]_f$
$[S_{43}]_f$	$\emptyset$
$[S_{48}]_f$	$[S_2]_s \rightarrow [S_2]_f$ $[S_{30}]_s \rightarrow [S_2]_f$ $[S_{33}]_s \rightarrow [S_{33}]_f$ $[S_{40}]_s \rightarrow [S_{33}]_f$ $[S_{43}]_s \rightarrow [S_{43}]_f$ $[S_{48}]_s \rightarrow [S_{48}]_f$ $[S_{55}]_s \rightarrow [S_2]_f$ $[S_{98}]_s \rightarrow [S_2]_f$
$[S_{99}]_f$	$\emptyset$

Table C.12: Euronet inverted scenario network table

Appendix D

SNeAT

SNeAT, the Scenario Network Analysis Tool, was discussed briefly in Section 4.9. This appendix gives more details about SNeAT and its use.

SNeAT's input language is in the form of tagged parenthesized lists; its grammar appears in Table D.1. Each list is parenthesized, and the type of list is identified by a tag that follows the opening parenthesis. Lists may contain strings and/or sublists. In the grammar below, italicized elements stand for strings or names (of scenario networks or scenarios); a "*" after a name or parenthesized list indicates the names or lists may occur 0 or more times; a "?" indicates it is optional. /* */ and // comments may appear anywhere whitespace appears.

```
(sn ScenarioNetworkName?
  (i InitialScenario* )
  (t TerminalScenario* )
  (s ScenarioName
   (f SortKey )?
   (f FollowSetScenarioName* )
   (r RamificationSetScenarioName* )
  )*
  (eqvTo ScenarioName ScenarioName* )*
)
```

Table D.1: SNeAT input format

The input format gives the name of the scenario network, the network's initial scenarios and terminal scenarios, a list of the scenarios in the network with each ones follow and ramification sets, and optionally one or more lists of equivalent scenarios. In each list of equivalent scenarios, the first name must appear elsewhere in the input, and the remaining names must not appear anywhere else, in order to guard against unintended results.

SNeAT's normal output form mirrors the input format, with the addition of tags for the additional information. The output form can be fed back in as input; the additional tags that are not accepted as input are guarded within comments in the output form.

The results of the checks and analyses are displayed as part of the normal output form. The tags for the various results, as well as the input tags, are presented in the table below.

The next several tables present the input file that defines the EMS scenario network, and several results that were produced from it using SNeAT. Table reftbl:ems shows the input file. The normal SNeAT output from this file, containing the analysis results, is given in Table D.4. Other SNeAT outputs from this input file appear in Chapter 4 in Figures 4.5 and 4.9, and in Appendix A in Figure A.1 and Tables A.3 through A.11.

SNeAT checks the scenario network for fundamental properties:

- That it has at least one initial and at least one terminal scenario.
- That every scenario is reachable from an initial scenario.
- That a terminal scenario is reachable from every scenario.

Additionally, SNeAT analyzes the network to determine the precede and trunk sets for each scenario, and the equivalence classes of scenarios for sequence equivalence, follow equivalence, and precede equivalence.

Other output forms and outputs are also available:

- The equivalence classes for sequence-, follow-, and precede-equivalence.
- Tables relating the equivalence classes for the three relationships.

Input tag	*What it represents*
`sn`	A scenario network
`i`	The initial scenarios
`t`	The terminal scenarios
`s`	A scenario and its follow and ramification sets
`sortKey`	A sort key
`f`	A follow set
`r`	A ramification set
`eqvTo`	An equivalence class, by which the table is to be expanded
Output-only tag	*What it represents*
`P`	A precede set
`T`	A trunk set
`noPathFromInitial`	A list of scenarios not reachable from initial scenarios
`noPathToTerminal`	A list of scenarios from which no terminal scenario is reachable
`eqv`	A sequence-equivalence class
`folEqv`	A follow-equivalence class
`preEqv`	A precede-equivalence class

Table D.2: Tags in SNeAT's input and output forms

- The scenario network tables in equivalence-compressed form, with each equivalence class represented in the table by a single examplar in square brackets.
- The inverted scenario network in input form.
- The scenario network diagram (in XY-pic form).
- The inverted scenario network diagram (in XY-pic form).

Each output is available in LaTeX form for inclusion and formatting in documents such as this one. All the scenario network diagrams and tables in this work were produced using SNeAT.

```
(sn EMS
 (i 0)
 (t 1 9 10 11 26 29 35 38 39)
 (s 0 (sortKey 00) (f 1) (r 2 3 4 9 30 37))
 (s 1 (sortKey 01))
 (s 2 (sortKey 02) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 3 (sortKey 03) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 4 (sortKey 04) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 5 (sortKey 05) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 6 (sortKey 06) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 7 (sortKey 07) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 8 (sortKey 08) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 9 (sortKey 09))
 (s 10)
 (s 11)
 (s 12 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r 11))
 (s 13 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 14 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 15 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 // There is no 16.
 (s 17 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 18 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 19 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 20 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 21 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 22 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 23 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 24 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 25 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 26)
 (s 27 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 28 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29))
 (s 29)
 (s 30 (f 31 32 33 34 35 38 39) (r 10))
 (s 31 (f 31 32 33 34 35 38 39))
 (s 32 (f 31 32 33 34 35 38 39))
 (s 33 (f 31 32 33 34 35 38 39))
 (s 34 (f 31 32 33 34 35 38 39))
 (s 35)
 // There is no 36.
 (s 37 (f 31 32 33 34 35 38 39) (r 10))
 (s 38)
 (s 39)
)
```

Table D.3: SNeAT input for the EMS

```
(sn "EMS"
 (i 0)
 (t 1 9 10 11 26 29 35 38 39)
  (s 0 (sortKey 00) (f 1) (r 2 3 4 9 30 37) /* (P) (T) */)
  (s 1 (sortKey 01) (f) (r) /* (P 0) (T) */)
  (s 2 (sortKey 02) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P) (T 0) */)
  (s 3 (sortKey 03) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P) (T 0) */)
  (s 4 (sortKey 04) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P) (T 0) */)
  (s 5 (sortKey 05) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15
      17 18 19 20 21 22 23 24 25 27 28) (T) */)
  (s 6 (sortKey 06) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15
      17 18 19 20 21 22 23 24 25 27 28) (T) */)
  (s 7 (sortKey 07) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15
      17 18 19 20 21 22 23 24 25 27 28) (T) */)
  (s 8 (sortKey 08) (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15
      17 18 19 20 21 22 23 24 25 27 28) (T) */)
  (s 9 (sortKey 09) (f) (r) /* (P) (T 0) */)
  (s 10 (f) (r) /* (P) (T 30 37) */)
  (s 11 (f) (r) /* (P) (T 12) */)
  (s 12 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r 11) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19
      20 21 22 23 24 25 27 28) (T) */)
  (s 13 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 14 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 15 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 17 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 18 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 19 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 20 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 21 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 22 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 23 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 24 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 25 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 26 (f) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 27 28) (T) */)
  (s 27 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 28 (f 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20
      21 22 23 24 25 27 28) (T) */)
  (s 29 (f) (r) /* (P 2 3 4 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 27 28) (T) */)
  (s 30 (f 31 32 33 34 35 38 39) (r 10) /* (P) (T 0) */)
  (s 31 (f 31 32 33 34 35 38 39) (r) /* (P 30 31 32 33 34 37) (T) */)
  (s 32 (f 31 32 33 34 35 38 39) (r) /* (P 30 31 32 33 34 37) (T) */)
  (s 33 (f 31 32 33 34 35 38 39) (r) /* (P 30 31 32 33 34 37) (T) */)
  (s 34 (f 31 32 33 34 35 38 39) (r) /* (P 30 31 32 33 34 37) (T) */)
  (s 35 (f) (r) /* (P 30 31 32 33 34 37) (T) */)
  (s 37 (f 31 32 33 34 35 38 39) (r 10) /* (P) (T 0) */)
  (s 38 (f) (r) /* (P 30 31 32 33 34 37) (T) */)
  (s 39 (f) (r) /* (P 30 31 32 33 34 37) (T) */)
 // (noPathFromInitial)
 // (noPathToTerminal)
 // (eqv 2 3 4)
 // (eqv 5 6 7 8 13 14 15 17 18 19 20 21 22 23 24 25 27 28)
 // (eqv 26 29)
 // (eqv 30 37)
 // (eqv 31 32 33 34)
 // (eqv 35 38 39)
 // (folEqv 2 3 4 9 30 37)
 // (folEqv 5 6 7 8 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29)
 // (folEqv 31 32 33 34 35 38 39)
 // (preEqv 1 9 10 11 26 29 35 38 39)
 // (preEqv 2 3 4 5 6 7 8 13 14 15 17 18 19 20 21 22 23 24 25 27 28)
 // (preEqv 30 37)
 // (preEqv 31 32 33 34)
 // 38 scenarios, 12 seq-equiv classes, 7 fol-equiv classes, 6 pre-equiv classes.
 // 514 transitions.
)
```

Table D.4: SNeAT output for the EMS, containing results of checks and analyses

Bibliography

[AA01] Thomas A. Alspaugh and Annie I. Antón. Scenario networks: A case study of the enhanced messaging system. In *Seventh International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'01)*, June 2001.

[AABM99] Thomas A. Alspaugh, Annie I. Antón, Tiffany Barnes, and Bradford W. Mott. An integrated scenario management strategy. In *RE '99: Fourth IEEE International Symposium on Requirements Engineering*, pages 142–149, June 1999.

[ABB99] Software requirements specification for Euronet v.2. Asea Brown Boveri – Electric Systems Technology Institute, March 1999. Confidential.

[ACD+01] Annie I. Antón, Ryan A. Carter, Aldo Dagnino, John H. Dempster, and Devon F. Siege. Deriving goals from a use-case based requirements specification. *Requirements Engineering Journal*, 6(1):63–73, 2001.

[AFB+92] Thomas A. Alspaugh, Stuart R. Faulk, Kathryn Heninger Britton, R. Alan Parker, David L. Parnas, and John E. Shore. Software requirements for the A-7E aircraft. NRL Memorandum Report 3876, Naval Research Laboratory, Washington, DC, August 1992.

[ALBG99] D. Amyot, L. Logrippo, R. J. A. Buhr, and T. Gray. Use case maps for the capture and validation of distributed systems requirements. In *RE'99: Fourth*

IEEE International Symposium on Requirements Engineering, pages 44–53, 1999.

[Ame87] Pierre America. Inheritance and subtyping in a parallel object-oriented language. In J. Bézivin et al., editors, *ECOOP '87, European Conference on Object-Oriented Programming*, number 276 in Lecture Notes in Computer Science, pages 234–242. Springer-Verlag, June 1987.

[Ant97] Annie I. Antón. *Goal Identification and Refinement in the Specification of Soft-ware-Based Information Systems*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, June 1997.

[AP98a] A. I. Antón and C. Potts. A representational framework for scenarios of systems use. *Requirements Engineering Journal*, 3(3–4):219–241, December 1998.

[AP98b] Annie I. Antón and Colin Potts. The use of goals to surface requirements for evolving systems. In *Proceedings of the 1998 International Conference on Software Engineering (ICSE'98)*, pages 157–166, April 1998.

[BC96] R. J. A. Buhr and R. S. Casselman. *Use case maps for object-oriented systems*. Prentice Hall, 1996.

[BdPL98] Karin Koogan Breitman and Julio Cesar Sampaio do Prado Leite. A framework for scenario evolution. In *ICRE'98: Third International Conference on Requirements Engineering*, pages 214–223, 1998.

[BGMM99] Elisa Bertino, Giovanna Guerrini, Isabella Merlo, and Marco Mesiti. An approach to classify semi-structured objects. In *ECOOP'99 - Object-Oriented Programming, 13th European Conference, Lisbon, Portugal, June 14-18, 1999, Proceedings*, pages 416–440, 1999.

[Bod02] Laura J. Bode. A scenario management case study: measuring scenario similarity in the EMS. Technical Report TR-2002-10, North Carolina State University, 2002.

[Boe81] Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[Boo93] G. Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings, second edition, 1993.

[Bræ96] Rolv Bræk. SDL basics. *Computer Networks and ISDN Systems*, 28(12):1585–1602, June 1996.

[Bro87] Fred Brooks. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, April 1987.

[Buh98] R. J. A. Buhr. Use case maps as architectural entities for complex systems. *IEEE Transactions on Software Engineering*, 24(12):1131–1155, December 1998.

[CEG+90] Glenn L. Coleman, Charles P. Ellison, Gentry G. Gardner, Daniel L. Sandini, and John W. Brackett. Experience in modeling a concurrent software system using STATEMATE. In *CompuEuro '90. Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering*, pages 104–108, May 1990.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CPS93] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The Concurrency Workbench: A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[DBB97] Bénédicte Dano, Henri Briand, and Franck Barbier. An approach based on the concept of use case to produce dynamic object-oriented specifications. In *RE'97: Third IEEE International Symposium on Requirements Engineering*, pages 54–64, 1997.

[DFKM98] Jules Desharnais, Marc Frappier, Ridha Khédri, and Ali Mili. Integration of sequential scenarios. *IEEE Transactions on Software Engineering*, 24(9):695–708, September 1998.

[Dij72] Edsger W. Dijkstra. The humble programmer. *Communications of the ACM*, 15(10):859–866, October 1972. 1972 ACM Turing Award Lecture.

[DLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, April 1993.

[EN95] Steve Easterbrook and Bashar Nuseibeh. Managing inconsistencies in an evolving specification. In *RE'95: Second IEEE International Symposium on Requirements Engineering*, pages 48–55, 1995.

[FFK94] Stuart Faulk, Lisa Finneran, and James Kirby, Jr. Experience applying the CoRE method to the Lockheed C-130J software requirements. In *Compass'94: 9th Annual Conference on Computer Assurance*, pages 3–8, Gaithersburg, MD, 1994. National Institute of Standards and Technology.

[GFS98] Juergen Gausemeier, Alexander Fink, and Oliver Schlake. Scenario management: An approach to develop future potentials. *Technological Forecasting and Social Change*, 59(2):111–130, 1998.

[Gor02] Pieter Van Gorp. The meeting scheduler system. Universiteit Antwerpen, 2002.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[HLK95] C. Heitmeyer, B. Labaw, and D. Kiskis. Consistency checking of SCR-style requirements specifications. In *RE'95: Second IEEE International Symposium on Requirements Engineering*, pages 56–63, March 1995.

[HN96] David Harel and Amnon Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, October 1996.

[ITU99a] Specification and description language (MSC). ITU-T Recommendation Z.100, International Telecommunications Union, November 1999.

[ITU99b] Message Sequence Chart (MSC). ITU-T Recommendation Z.120, International Telecommunications Union, November 1999.

[JBC98a] Matthias Jarke, X. Tung Bui, and John M. Carroll. Dagstuhl workshop on scenario management. Dagstuhl Seminar Report 199, Schloss Dagstuhl International Conference and Research Center for Computer Science, February 1998.

[JBC98b] Matthias Jarke, X. Tung Bui, and John M. Carroll. Scenario management: An interdisciplinary approach. *Requirements Engineering Journal*, 3(3–4):155–173, 1998.

[JCJÖ92] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. ACM Press, 1992.

[Kru83] Joseph B. Kruskal. An overview of sequence comparison. In *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 1–44. Addison-Wesley, 1983.

[LDL98] Axel van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing conflicts in goal-directed requirements engineering. *IEEE Transactions on Software Engineering*, 24(11):908–925, November 1998.

[LR94] David Lauzon and Thomas Rose. Task-oriented and similarity-based retrieval. In *Proceedings of the 9th Knowledge-Based Software Engineering Conference*, pages 98–107, September 1994.

[LW98] Axel van Lamsweerde and Laurent Willemet. Inferring declarative requirements specifications from operational scenarios. *IEEE Transactions on Software Engineering*, 24(12):1089–1114, December 1998.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[ML97] P. Massonet and A. van Lamsweerde. Analogical reuse of requirements frameworks. In *Proceedings: 3rd IEEE International Symposium on Requirements Engineering*, pages 26–39. IEEE Computer Society Press, 1997.

[MMMR98] N. A. M. Maiden, S. Minocha, K. Manning, and M. Ryan. CREWS-SAVRE: Systematic scenario generation and use. In *Proceedings: 3rd International Conference on Requirements Engineering*, pages 148–155, 1998.

[MR97] S. Mauw and M. A. Reniers. High-level message sequence charts. In *Proceedings of the Eighth SDL Forum (SDL'97)*, pages 291–306, 1997.

[MS96] N. A. M. Maiden and A. G. Sutcliff. Analogical retrieval in reuse-oriented requirements engineering. *Software Engineering Journal*, 11(5):281–292, September 1996.

[NKF94] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, October 1994.

[NoDRC+01] Johan Natt och Dag, Björn Regnell, Pär Carlshamre, Michael Andersson, and Joachim Karlsson. Evaluating automated support for requirements similarity analysis in market-driven development. In *Seventh International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'01)*, June 2001.

[OMG99] OMG Unified Modeling Language Specification (version 1.3). Document ad/99-06-08, Object Management Group, Framingham, MA, June 1999.

[Pet77] James L. Peterson. Petri nets. *ACM Computing Surveys*, 9(3):223–252, September 1977.

[PKKS00] S. Park, H. Kim, Y. Ko, and J. Seo. Implementation of an efficient requirements-analysis supporting system using similarity measure techniques. *Information and Software Technology*, 42(6):429–438, 2000.

[Pot93] Colin Potts. Software engineering research revisited. *IEEE Software*, 10(5):19–28, September 1993.

[Pot01] Colin Potts. Class slides for cs3300. Georgia Institute of Technology, 2001. http://www.cc.gatech.edu/classes/AY2001/cs3300_fall/scenarios.htm.

[PTA94] Colin Potts, Kenji Takahashi, and Annie I. Antón. Inquiry–based requirements analysis. *IEEE Software*, 11(2):21–32, March 1994.

[RBA98] Colette Rolland and Camille Ben Achour. Guiding the construction of textual use case specifications. *Data and Knowledge Engineering Journal*, 25(1–2):125–160, March 1998. Also published as CREWS technical report 98-01.

[REJ98] *Requirements Engineering Journal*, 3(3–4), 1998. Special Issue: Interdisciplinary Uses of Scenarios.

[RGG96] Ekkart Rudolph, Peter Graubmann, and Jens Grabowski. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, June 1996.

[RGK99] C. Rolland, K. Grosz, and R. Kla. Experience with goal-scenario coupling in requirements engineering. In *RE'99: Fourth IEEE International Symposium on Requirements Engineering*, pages 74–83, 1999.

[RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Object Technology Series. Addison Wesley Longman, Reading, Mass., 1999.

[RM93] K. Ryan and B. Mathews. Matching conceptual graphs as an aid to requirements re-use. In *RE'93: IEEE International Symposium on Requirements Engineering*, pages 112–120, 1993.

[RSBA98] Colette Rolland, Carine Souveyet, and Camille Ben Achour. Guiding goal modeling using scenarios. *IEEE Transactions on Software Engineering*, 24(12):1055–1071, December 1998.

[Sha01] Mary Shaw. The coming-of-age of software architecture research. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)*, pages 657–664a, May12–19 2001.

[Sim87] Peter Simons. *Parts : A study in ontology.* Clarendon Press, Oxford, 1987.

[Sim99] Anthony J. H. Simons. Use cases considered harmful. In *29th Conference on Technology of Object-Oriented Languages and Systems*, pages 194–203, 1999.

[SM92] Sally Shlaer and Stephen J. Mellor. *Object Lifecycles: Modeling the World in States.* Yourdon Press, 1992.

[SMMM98] Alistair G. Sutcliffe, Neil A. M. Maiden, Shailey Minocha, and Darrel Manuel. Supporting scenario-based requirements engineering. *IEEE Transactions on Software Engineering*, 24(12):1072–1088, December 1998. Special Issue: Scenario Management.

[SS00] S. Sendall and A. Strohmeier. From use cases to system operation specifications. In *Third International Conference on the Unified Modeling Language UML'2000*, pages 1–15, 2000.

[TSE98] *IEEE Transactions on Software Engineering*, 24(12), December 1998. Special Issue: Scenario Management.

[Tve77] Amos Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, July 1977.

[Wie98] Roel Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Computing Surveys*, 30(4):459–527, December 1998.

[WPJH98] Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke, and Peter Haumer. Scenarios in system development: Current practice. *IEEE Software*, 15(2):34–45, March/April 1998.

[Yin94] Robert K. Yin. *Case study research : design and methods*. SAGE Publications, Thousand Oaks, California, 1994.

Index