Submitted to IEEE Transactions on Software Engineering

Scenario Networks for Software Specification and Scenario Management

TR-2001-15

Thomas A. Alspaugh, *Student Member, IEEE,* and Annie I. Antón, *Member, IEEE*

Abstract— **Scenarios are widely used to specify the desired behavior of a system, but managing the large collection of scenarios that frequently result and making a scenario-based specification complete are challenging tasks. Scenario networks address these challenges while retaining the many advantages of scenarios during software specification activities. A scenario network is a collection of scenarios that has been integrated into a single entity by the specification of the sequential and concurrent relationships among its component scenarios. The addition of these relationships specifies the larger-scale behavior that is typically missing from a collection of scenarios, and ties scenarios together in a way that either indicates no gaps in the description are present, or makes gaps obvious. Scenario networks provide procedural guidance for scenario creation and support for scenario management. Gaps in the structure of a scenario network correspond to missing or incomplete scenarios, and the closing of these gaps result in the completion of the scenario collection. A scenario network organizes the collection of its scenarios, and its structure indicates several kinds of scenario relationships, including equivalence relations dividing them into equivalence classes. These relationships address some of the challenges associated with scenario management.**

Keywords— **Scenario networks, requirements engineering, scenario analysis, scenario management, software specification.**

I. Introduction

THE use of scenarios in software development has become increasingly common [1]. Scenarios are useful for describing required behavior that a system will have; expressing what part of a system is visible from a particular viewpoint; and specifying a test case the system should pass. A scenario is a sequence of events, each consisting of an actor (human or otherwise) who performs the event and the action that is performed [2]. Scenarios are typically expressed in prose and thus are accessible to any reader. Since they are often expressed in the language and terms of the system stakeholders, they can be understood by the stakeholders as well as by analysts and developers. Each scenario narrates a sequence of events, and this focus helps the scenario author in choosing what facts the scenario should cover, and the scenario reader in relating the facts the scenario presents.

While scenarios have proven beneficial [1], [3], [4], [5], [6], [7], their use also involves certain challenges [1], [2]. A scenario is inherently partial; each scenario offers a single view of a single part of a system's behavior. Even a large collection of scenarios may not provide a complete description of a system, and it is difficult to determine to what extent the collection is complete. Additionally, the relationships and dependencies between scenarios in a collection are not obvious. Finally, the scenarios in a collection are unlikely to be consistent with each other, especially after the intended behavior of the system has evolved during development. These challenges, inherent in scenario-driven requirements engineering, have been a central motivation for our work. This work has two objectives: (1) to ensure requirements coverage via the specification of a complete and consistent set of scenarios, and (2) to develop strategies to support scenario evolution and management.

The kind of specification information that is typically missing from a collection of scenarios consists of information that is on a larger scale than a single scenario, such as temporal, causal, or dependency relationships between scenarios. The temporal relationships between scenarios are an important part of the intended behavior of the system. Causal relationships between scenarios express that the occurrence of one scenario can cause or prevent the occurrence of another scenario, whether immediately or at some later time. Dependency relationships between scenarios express that for a particular scenario, the events chosen for it, the scope of its effects, the form in which it is expressed, or other features of it depend on some aspect of another scenario. A number of other relationships between scenarios can be significant, such as interchangeability between scenarios or substitutability of one scenario for another. These relationships are not typically expressed in a collection of scenarios, or supported by current approaches to scenario-driven requirements engineering.

In this paper we discuss our approach for expressing this missing information using scenario networks. A *scenario network* is a collection of scenarios whose sequential and concurrent relationships have been made explicit. It can be used to provide a complete specification of a system's important behavior, including not only the behavior expressed in individual scenarios but also behavior that spans more than one scenario. The unification and integration of its constituent scenarios contributes to improving the quality of those scenarios; aids in determining whether the collection of scenarios is complete; and guides the process

T. A. Alspaugh (corresponding author) and A. I. Antón are at the Department of Computer Science, North Carolina State University, 1010 Main Campus Drive (EGRC 408), Raleigh, NC 27695-7534 U.S.A. Email: {taalspau,aianton}@eos.ncsu.edu.

of completing them.

Throughout the paper we use the Enhanced Messaging System (EMS) as an example [5]. 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); as well as recording, playing, and archiving of subscriber outgoing messages.

The remainder of this paper is organized as follows. Section 2 of this paper summarizes related work. Section 3 introduces scenario networks. Section 4 presents the relationships between scenarios that scenario networks support. Section 5 compares two approaches for constructing a scenario network. Section 6 discusses the application of scenario networks to software development. Section 7 summarizes our findings and plans for future work.

II. Related work

Our work on scenario networks builds upon research in a number of areas in requirements engineering. Each subsection below provides an introduction to one of these areas, summarizes the most pertinent related work, and discusses how scenario networks build upon that work.

A. Scenario management

We use the term *scenario management* to refer to the management and administration of a collection of scenarios. Scenario management addresses issues that arise as the number of scenarios for a system increases, such as organizing, listing, and classifying the scenarios for the system; tracing dependencies among scenarios, and between scenarios and other artifacts; determining and maintaining relationships among scenarios; finding the scenario that defines a particular behavior of the system; detecting and eliminating duplicate or near-duplicate scenarios; managing inconsistencies among scenarios; and determining whether a group of scenarios is complete. It is needed for any system with more scenarios than one person can keep track of in his or her head.

These issues have received little attention. 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 [1]. Traceability, maintenance of consistency with other artifacts, and scenario evolution and management are significant problems that have not been adequately addressed by research [1]. Jarke *et al.* also note that little research addresses the problems that arise in managing a large set of scenarios [8]. A special issue of the IEEE Transactions on Software Engineering [9] is devoted to scenario management, and a related issue of the Requirements Engineering Journal [10] is devoted to interdisciplinary uses of scenarios, but the articles in these two 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.* [8], which contains a section on "scenario management in the large"). More recently, Alspaugh *et al.* have proposed an approach to scenario management based on glossaries, episodes (scenario fragments that appear in several scenarios), and measures of similarity between scenarios. 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 [2].

Scenario networks provide an organizing structure for a system's scenarios, and the scenario relationships that a scenario network uncovers provide ways to classify the scenarios.

B. 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.

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 [1]. 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 [3]. 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 [4], [11]. 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 [12]. The process employs guidelines for effective expression of use cases in prose, and rules that ask for more information or generate a new use case element from what is already known. The guidelines and rules are based on linguistic patterns and structures [12]. The work on viewpoints discussed below provides process guidance for improving consistency among scenarios.

Process guidance is an important benefit of the use of scenario networks. A scenario network aids in identifying missing scenarios and behaviors by forcing analysts to consider how the scenarios can occur in sequences or concurrently. The consistency this enforces between scenarios is different than that provided by viewpoints.

C. Viewpoints

A viewpoint is a partial specification of a system from a particular perspective [13], [14]. This is usually the perspective of a single actor who interacts with the system. Viewpoints are used as a means of expressing 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 [13], [14]. 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 guidance for eliciting and elaborating the specification [14]. 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 [13].

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.

D. 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 producing object type state diagrams from them [15]. A use case map (UCM) integrates use cases to provide a whole-system specification [16], [17]. The use cases are expressed as causal sequences of responsibilities and denoted graphically as a graph with responsibilities attached. Concurrency is explicitly represented in a UCM. 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 [18]. 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) [19]. The conditions express when each operation can occur and the effect of each operation. 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 of a system, since it can describe both a principal sequence of actions and also possible variants for alternate orderings, exceptional cases, or error handling [20]. Only scenarios that are variants on each other are contained in a single use case, however. Viewpoints provide a means of integrating overlapping scenarios, as discussed above [13], [14].

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

E. 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 agents and episodes [12]. 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 [21].

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

F. 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 I.

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

TABLE I
COMMONLY USED SCENARIO RELATIONSHIPS

Jacobson *et al.* discussed the "uses" and "extends" relationships in their original work on use cases [22]. Simons and others note that the Jacobson and the Unified Modeling Language (UML) definitions and redefinitions of "uses" and "extends" have been problematic and are still not entirely satisfactory; 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 [23]. Breitman and Leite define and use relationships between scenarios (overlap, equivalence, and subset) to classify and guide the evolution of scenarios [24]. Alspaugh *et al.* discuss the importance of dependency relationships between scenarios and their preservation as the scenarios evolve [2].

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

G. Concurrency between scenarios

Concurrency in requirements specification has been addressed by, for example, the CoRE method [25] the modeling approach of Coleman *et al.* [26], and the Specification and Description Language (SDL) and Message Sequence Charts (MSC) [27], [28]. 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 [29]. The graphic notation of UCMs expresses the concurrency that is desired between use cases, and has been used to detect feature interactions [16]. There is a large body of established work on concurrency and concurrent systems [30], [31], [32], but it been little applied to concurrency between scenarios.

Scenario networks begin to address concurrency between scenarios and this continues to be an area of focus for future work.

H. 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 [33]. If an arc runs from a node c to another node d (whether a place or transition) then c is said to be an input to d, and d an output from c. The state of the 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 them 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 [33]. The structure of Petri nets does not map to the structure of the systems they describe or the problems they solve, however, so it can be difficult to trace a feature of a Petri net to a feature of a description of the system in another form.

High-Level Message Sequence Charts (HMSCs) are a formalism for describing systems in terms of Message Sequence Charts (MSCs) [28]. A Message Sequence Chart is a graphical representation of sequences of messages transmitted between instances (systems, components, processes, etc.) [28], [34]. 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 partially 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 [28], [35].

An extended finite state machine is a finite state machine (FSM) whose states have been augmented by variables [36]. 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 can be changed by the explicit state, by transitions between explicit states, or possibly from outside the extended FSM; the values of the variables may be used as guards for each transition. Extended FSMs are widely used to describe many sorts

of systems, and occur in several variants, including Statecharts [37], [38] and state diagrams in the Specification and Description Language (SDL) [27], [39]. Statecharts extend the basic idea of extended FSMs 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 extended FSM), 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.

III. Scenario networks

This section defines specific terminology and introduces scenario networks.

A. Terminology

We define the following key terms.

A *scenario* is a sequence of events, plus possibly some associated attributes such as pre- and postconditions [2]. Each event consists of an action and an actor that performs it. 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 interconnections between them that indicate the allowed scenario sequences and concurrency. The interconnections are restricted by the pre- and postconditions of each scenario. At least one of the scenarios is distinguished as initial, and at least one other as terminal.

A *multipath* is a possibly ramified path along the connections between the scenarios of 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.

An *initial scenario* of a network is one that can begin a multipath in the network, and can appear nowhere else.

A *terminal scenario* of a network is one that can end a branch of a multipath in the network, and can appear nowhere else.

The *precondition* of a scenario is a logical expression that must be true in order for the scenario to begin. The precondition of scenario S_A is denoted $\mathrm{Pre}(\mathrm{S_A})$.

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 $\mathrm{Post}(\mathrm{S_A})$.

The *primitive terms* of a network are a set of variables of Boolean, integer, or other types, in which the conditions of scenarios in the network 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 $\mathrm{Follow}(\mathrm{S_A})$.

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 $\mathrm{Precede}(\mathrm{S_A})$.

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 $\mathrm{Ramify}(\mathrm{S_A})$.

B. Motivation and example

Individual scenarios are frequently used to describe a single transaction or a single sequence of events accomplishing a particular purpose (as in Scenario S_{12} shown in Table II). A scenario describes part of a system's behavior, and a group of scenarios describes the entire behavior of a system. Ideally, every system behavior is expressed by one or more scenarios in the group. For illustration, we consider scenarios for the Enhanced Messaging System, a voice mail system discussed in our earlier case study [5].

S_{12}. Subscriber listens to the next message.	
1.	Subscriber s dials the *next message* command.
2.	EMS plays s's next message.
3.	If that message was 'new', its state becomes 'old'

TABLE II
EMS Scenario S_{12}

What is not expressed by a group of scenarios is the allowed temporal relationships among all the scenarios. There is no specification of either the allowable sequences of scenarios or concurrency between sequences of scenarios. Scenario networks provide a way to express this additional information. Any allowable behavior of the system corresponds to a (possibly ramified) path through the network,

beginning at an initial scenario and continuing until each branch of the path reaches a terminal scenario.

A detailed example of a sequential scenario network for a simplified EMS is presented in our earlier work [5]. Here we present an example from the complete EMS that demonstrates concurrency, using the scenarios listed in Table III.

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 III
EMS SCENARIOS USED IN THE EXAMPLE

In the context of the EMS, expected or desired behaviors are represented by a number of multipaths through the scenarios listed in Table III. Wherever a multipath diverges into two or more paths, it indicates concurrency between the scenario sequences on the parallel paths. Some allowed multipaths for the EMS are listed in Table IV.

The list of allowed multipaths continues without end, so rather than listing the multipaths we create a scenario network that expresses exactly the multipaths that are allowed.

We express scenario networks in one of two equivalent ways: in tabular form (see Table V), or as a diagram (see Figure 1). In each form, the scenarios in the network may be further restricted by pre- and postconditions. For the tabular 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 V provides this information for the example scenarios.

A second way to express a scenario network is by producing a diagram in which scenarios, represented by circles, are connected by arrows indicating sequence, and slashed arrows indicating where multipaths through the network may branch, as portrayed by the scenario network diagrams in Figures 1 and 2.

Figure 1 shows the entire scenario network diagram for the completed EMS. In the diagram, scenarios are represented by labelled circles, and transitions between them by arrows. Each gray circle indicates the Cartesian product of all arrows into it with all arrows out of it, and all arrows out with all arrows in; otherwise the diagram would contain a thicket of arrows. For example, at the gray 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}. For purposes of illustration in this paper, we focus our discussion on a portion of the scenario network shown in Figure 2.

The scenario network in Figure 2 supports all the allowed scenario multipaths listed in Table IV, and an infinite num-

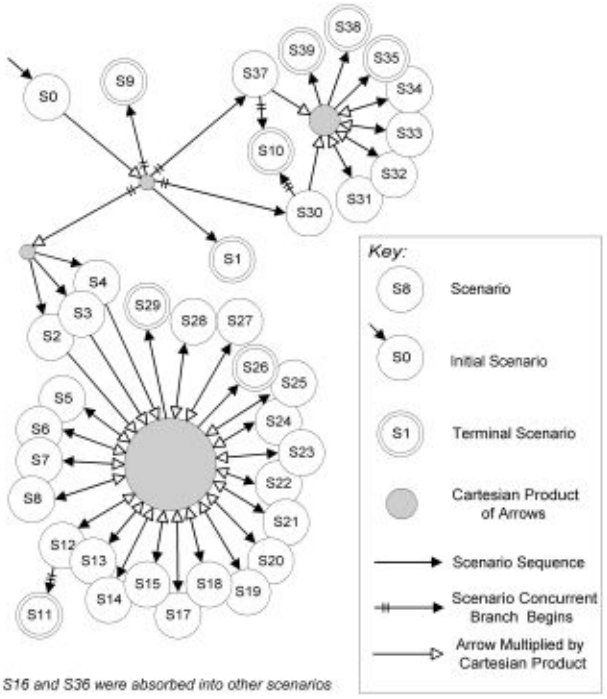


Fig. 1. Diagram of EMS scenario network

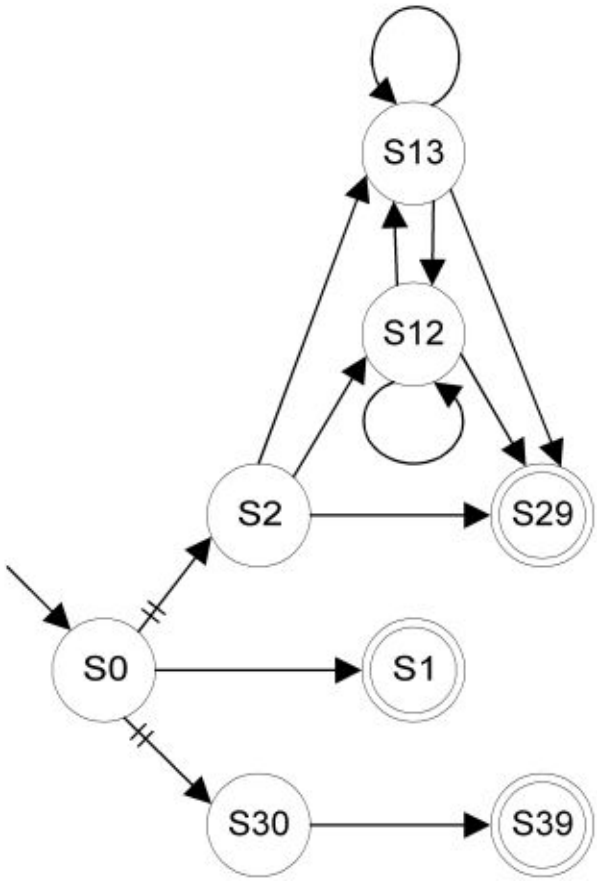


Fig. 2. Diagram for example scenario network

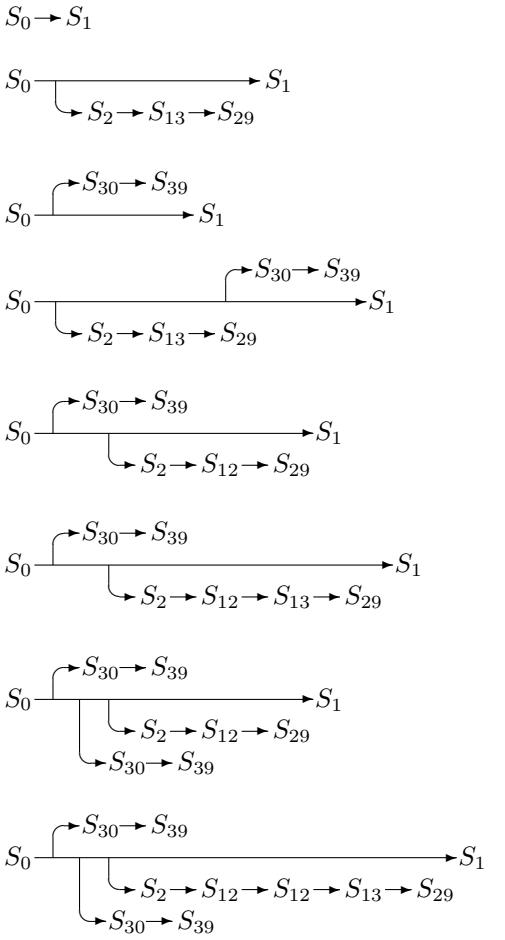
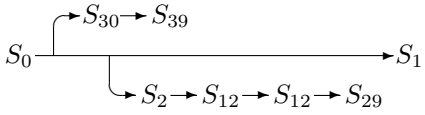


TABLE IV
SOME ALLOWED EMS MULTIPATHS

ber 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 assigning a precondition to each scenario. A scenario's precondition is required to be true in order for the scenario to begin. Each scenario is also assigned a postcondition which is satisfied at the end of the scenario. 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 example scenarios are given in Table VI.

We note that scenario networks may appear on the surface 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, so that 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 nondeterminism.

Initial scenarios	S_0	
Terminal scenarios	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 V
TABULAR FORM FOR EXAMPLE SCENARIO NETWORK

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 VI
PRE- AND POSTCONDITIONS OF EXAMPLE SCENARIOS

C. Scenario networks as system simulations

Scenario networks simulate a system by presenting sequences of scenarios that correspond to desired behaviors of the system. Each individual scenario describes part of the system's behavior in the usual way; 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 are taken to occur in the order the scenario specifies. For the purposes of the scenario network, each scenario is considered to occur atomically; the scenario network does not reflect any details of the interior of the scenario. This simplification is to allow us to separate concerns by concentrating on relationships between scenarios, and by con-

sidering each scenario only in terms of its conditions and its follow and ramification sets.

A scenario network begins a multipath with an initial scenario, then continues by following each scenario's follow and ramification sets. A scenario is allowed to begin only when it is a possible next scenario and its precondition is true; if both these are the case, the scenario can be triggered by its first event. The possible next scenarios may be visualized using one or more counters, initially a single counter on an initial scenario. The possible next scenarios are all the scenarios in the follow or ramify sets of a scenario that has a counter. If a scenario in a ramify set is triggered, a new counter is given to that scenario, indicating a new concurrent thread and beginning a new branch of the multipath. If a scenario in a follow set is triggered, that scenario is given its predecessor's counter and the existing branch of the multipath is extended. Only one of the scenarios in the predecessor's follow set can be triggered. When 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, we consider that their postconditions are applied in some particular sequence, without defining which of the possible sequences occurs. When each counter reaches a terminal scenario, it has no further effect since terminal nodes have empty follow and ramify sets. When all a scenario network's counters have reached terminal nodes, the multipath is complete.

We note that it is possible for a scenario network to hang, for its scenarios to set its 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 scenario network 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 fields of an object or mappings from the set of subscribers onto another set of values.

It is important to note that although a scenario network simulates a system, it does not itself represent the system being modelled or any other entity in the environment. Instead, it represents possible sequences of events, nothing more, and the actors of those events may be the system or components of the system, entities in the environment, people interacting with system or in the environment, or anything else that can cause an event to occur. All these actors are distinct from the scenarios that describe them, and from the scenario networks that are made up of the scenarios. The system (and all the other actors) are present only by virtue of the events in which they participate, and these events may be distributed across many scenarios.

IV. Relationships between scenarios

Several kinds of relationships between scenarios are evident from the scenario sequences for a scenario network. Each of these relationships can be expressed in terms of follow sets. In the definitions below, S_A, S_B, S_C, S_{sub}, and S_{super} represent arbitrary scenarios.

The elementary relation *can be followed by* ($\rightharpoonup$) expresses follow and ramification sets as a relation. $S_A \rightharpoonup S_B$ if S_B is either in S_A's follow set or ramification set.

$$S_A \rightharpoonup S_B \triangleq S_B \in \mathrm{Follow(S_A)} \cup \mathrm{Ramify(S_A)}$$

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

$$S_A \sqsupseteq_{\mathrm{fol}} S_B \triangleq \forall S_C.(S_C \rightharpoonup S_B) \Longrightarrow (S_C \rightharpoonup S_A)$$

$$S_A \sqsupseteq_{\mathrm{pre}} S_B \triangleq \forall S_C.(S_C \leftharpoonup S_B) \Longrightarrow (S_C \leftharpoonup S_A)$$

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 $\sqsupseteq_{\mathrm{seq}}$.

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

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

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

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

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

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

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

$\sqsupseteq_{\mathrm{fol}}$ and $\sqsupseteq_{\mathrm{pre}}$, $\sqsupseteq_{\mathrm{seq}}$, $\equiv_{\mathrm{fol}}$and $\equiv_{\mathrm{pre}}$, and $\equiv_{\mathrm{seq}}$ 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 track each other as changes occur.

One of the most interesting relationships arising in a scenario network is a substitution relationship analogous to the behavioral subtype relationship [40], in which S_{sub} is

a behavioral subtype of S_{super} (denoted $S_{sub} <: S_{super}$) iff S_{sub} has all the behavior of S_{super}, and possibly some additional behavior. Then S_{sub} can be substituted for any occurrence of S_{super} and still satisfy the behavior (here, the pre- and postconditions) expected from S_{super}. The "can be substituted for" relationship $\sqsupseteq_{\text{seq}}$ is a step in this direction. In analogy to object-oriented types and subtypes, we consider that a pre- and postcondition define a *scenario type*, and that two scenarios are of the same type if their preconditions and postconditions are equivalent. Then the appropriate reflexive and transitive *subtype relationship* among scenarios $S_{sub} <: S_{super}$ occurs whenever two conditions hold:

(1) S_{sub}'s precondition is no stronger than S_{super}'s (so that S_{sub} can begin whenever S_{super}) can)

$$\text{Pre}(\text{S}_{\text{super}}) \Longrightarrow \text{Pre}(\text{S}_{\text{sub}})$$

(2) S_{sub}'s postcondition is no weaker than S_{super}'s (so that S_{sub} fulfills all the conditions that S_{super} does)

$$\text{Post}(\text{S}_{\text{sub}}) \Longrightarrow \text{Post}(\text{S}_{\text{super}})$$

These two conditions call to mind the covariance and contravariance appearing in object-oriented subtyping [41]. Behavioral subtyping of scenarios by this definition has the desirable property that it may be automatically determined, for an appropriate language of conditions.

S_A *is equivalent to* S_B	$\equiv_{\text{seq}}$
S_A *can be substituted for* S_B	$\sqsupseteq_{\text{seq}}$
S_A *can begin whenever* S_B *can*	$\sqsupseteq_{\text{fol}}$
S_A *is a refinement of* S_B	$<:$ or $\sqsupseteq_{\text{seq}}$

TABLE VII
SOME SCENARIO RELATIONSHIPS AND POSSIBLE FORMALIZATIONS

V. CONSTRUCTING A SCENARIO NETWORK

In this section, we discuss two systematic approaches to construct a scenario network for a collection of scenarios: (1) annotating each scenario with pre- and postconditions, and (2) refining large-scale narratives covering many scenarios about the system's behavior into smaller units.

A. Constructing a scenario network from pre- and postconditions

The basis of the pre- and postcondition approach is to consider what result each scenario achieves, and what it requires in order to achieve that result. We express the objectives and needs of each scenario as logical pre- and postconditions in [5]. Then scenario S_B can occur immediately after scenario S_A if $(\text{Post}(\text{S}_\text{A}) \wedge \text{Pre}(\text{S}_\text{B}))$ is satisfiable, that is, if there is a set of values for the primitive terms of the network for which both the postcondition of S_A and the precondition of S_B are true. However, S_A and S_B are not necessarily in the same sequential branch; satisfiability only indicates that they can be adjacent in a linearization of some multipath. In order to make satisfiability indicate which scenarios can follow each other sequentially, we first partition the scenarios into subsets, based on which scenarios can appear in sequence without beginning a new concurrent branch. Then within each subset, satisfiability of $(\text{Post}(\text{S}_\text{A}) \wedge \text{Pre}(\text{S}_\text{B}))$ indicates that S_B is in the follow set of S_A, and equivalently that a sequential arrow connects S_A to S_B in the diagram of the scenario network. A table or a diagram of the scenario network can be produced from the conditions in combination with the partitioning and the concurrent branching among the subsets, and relationships between conditions express equivalence or ordering relationships between the scenarios with those conditions. However deriving a diagram and these relationships without automated support is a cumbersome process.

The EMS case study demonstrated that this was not an efficient method for constructing a scenario network. We found that in constructing the pre- and postconditions first, the knowledge the analyst already has about the desired sequencing among the scenarios is effective in debugging and verifying the conditions. But by the time the analyst's knowledge could be brought to bear on the problem in this way, a substantial effort had already been expended in creating the initial pre- and postconditions. We also found that in using this knowledge to debug the follow sets, the follow sets converged slowly towards acceptable results. We hypothesized that slow convergence resulted in part because the follow sets depend on the interactions between many pre- and postconditions, and this approach did not provide sufficient guidance in the question of which condition to address next. Too much time was spent adjusting first one condition then another, and then backtracking. In the remainder of this paper, we discuss the narrative refinement approach, which proved more effective.

B. Constructing a scenario network by narrative refinement

In the narrative refinement approach, analysts consider the envisioned system that is described by the collection of scenarios. The system's behavior is first described with a single high-level story, expressed in natural language. For example, in the EMS case study the single high-level story was "The EMS records messages that callers leave for particular subscribers; subscribers may listen to these messages and store them for a time if they wish." The stories are then successively decomposed into smaller, more detailed stories until each story contains enough detail so that all the desired system behaviors are described in some story. These smaller stories may be extracted from the higher-level stories in several ways: choosing a single actor and following his or her interactions with the system; following a single concurrent thread of behavior of the system and its environment; or identifying alternatives to a previously identified path of actions.

As the stories become progressively more detailed, they yield sequences of the specific scenarios covering actions that take place in that story. They indicate sequential, alternative, and concurrent relationships among their com-

ponent scenarios, from which a scenario network diagram can be constructed.

C. Parallel construction of scenarios and network

Other approaches are possible and constitute future work. The two approaches described above (derivation directly from conditions, and narrative refinement) assume that a collection of scenarios already exists before the scenario network is begun, but in practice this need not be the case. We have seen indications that an approach in which the scenarios and the scenario network are constructed in parallel can offer further advantages. In this situation, the scenario network provides process guidance throughout the creation of the scenarios, rather than only when the scenario collection is nearly complete. Individual scenarios can be produced by elaborating the stories to the desired level of detail, and dividing them into smaller stories and ultimately scenarios. The additional effort required for constructing the scenario network would be minimal with this approach.

VI. Applying scenario networks

Scenario networks offer many benefits to developers as they seek to specify a system's requirements. A scenario network records the sequences and concurrency that can occur among its scenarios, information that is not part of the individual scenarios and may not otherwise be recorded. A scenario network provides a clear visual representation of those sequences and concurrency, in the form of the scenario network diagram. The process of creating a scenario network helps the analyst improve the quality of the individual scenarios and (our case study showed) also helps the analyst uncover problems with the requirements and thus improve their quality as well (Section VI-A discusses this in greater detail). A completed scenario network acts as a system specification that includes everything the component scenarios specify and additionally specifies behavior that spans more than one scenario, thus providing a more effective operationalization of the system's requirements. All these benefits aid in the validation of the analysts' understanding of the system's behavior, the system specification (which is the scenario network itself), and the system requirements. In this section, we discuss in more detail how the application of scenario networks during requirements engineering activities offers two specific benefits: it offers process guidance to analysts, and provides a framework for scenario management as discussed in sections A and B below.

A. Process guidance

Scenario networks offer process guidance in several ways. A scenario network provides a useful roadmap for guiding walkthroughs of the scenarios. The benefits of walkthroughs are widely known [3], [4]. The process of constructing a scenario network provides important benefits besides the obvious one of producing the network. Constructing a scenario network by either of the systematic approaches described in Section V spurs a comprehensive sequence of walkthroughs; uncovers equivalence classes and subtypes among the scenarios and results in improved consistency in what the scenarios expect from and provide to one another, based on these relationships; and draws attention to gaps, overlaps, and shared episodes among the scenarios. We discuss these points in more detail below.

Scenario walkthroughs

A scenario network diagram provides a graphical aid that is well suited to walkthroughs. The directed-graph layout of the diagram suggests paths to follow and guides the choice of what paths to traverse or walk through next. A scenario network guides the walkthrough process by directing walkthroughs along a sufficiently exhaustive selection of the multipaths allowed by the scenario network. Although a scenario network does not indicate how large a selection would be sufficiently exhaustive, or exactly which multipaths should be in the selection, it does indicate what multipaths there are to select from. In the simplest sense, a scenario network acts as a reminder to visit each scenario along the path a walkthrough follows. A scenario network makes visually explicit the points at which two or more alternative paths exist, and provides a structure for following each alternative in succession in an organized fashion. Similarly, a scenario network makes visually explicit the points at which there are opportunities for additional concurrency in a walkthrough, including simultaneous instances of the same concurrent branch. A scenario network also guides the consideration of consistency among a set of walkthroughs that describe the desired behavior of a system, by providing a specification against which to verify that each walkthrough's sequences of scenarios and concurrency among sequences are contained in the scenario network's multipaths. Finally, the diagram provides a form in which certain kinds of information uncovered by the walkthrough can be conveniently recorded as the walkthrough proceeds. We note particularly that the primitive terms of a scenario network and the pre- and postconditions of scenarios in the network can record many kinds of behavior and relationships that span several scenarios. For example, the primitive terms $s.tot$ and $s.rem$ express several relationships between 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", and S_{30} "Caller calls a subscriber and leaves a message", including the relationship that S_{30} for a given subscriber s enables the occurrence of S_{12} for s, and the relationship that S_{13} for a given subscriber s can only occur if the number of occurrences of S_{30} for s is the same as the number of occurrences of S_{12} for s since the most recent occurrence of S_2 for s.

Guidance from systematic construction

The process of constructing a scenario network impels a continuing series of walkthroughs to check the part of the scenario network completed so far and to examine how remaining scenarios should be added to the network. Our studies indicate that an analyst naturally tends to check each change to his/her scenario network by walk-

ing through several characteristic narratives for multipaths that pass through the scenarios or connections that were changed. Similarly, an effective way to determine how to add a scenario to a network is to walk through some narratives that involve that scenario. For example, consider the process of adding the scenario S_{37} "Caller calls EMS directly and leaves a message" to the scenario network of Figure 2. Walking through the paths that could involve this scenario shows that S_{37} is quite similar (follow- and precede-equivalent) to S_{30} "Caller calls a subscriber and leaves a message." Thus S_{37} should be connected into the diagram in exactly the ways that S_{30} is. When we examine the full EMS diagram in Figure 1, we see that this is indeed the case. We also find that for an analyst, thinking about the system in terms of walkthroughs suggests additional new multipaths that should be walked through and specified. These walkthroughs offer opportunities to improve the quality of individual scenarios and the analyst's understanding of the system as recorded in the scenario network.

Another benefit that occurs as a scenario network is constructed is the uncovering of equivalence classes of scenarios, and the establishment of subtype orderings among the equivalence classes. Construction of a scenario network forces the analyst to consider which scenarios are equivalent and to what degree. This occurs as the analyst considers which scenarios can follow or precede a particular scenario, as part of the process of creating a scenario network diagram. The diagram expresses these relationships indirectly and graphically, and a correct diagram is one in which (among other things) each of these relationships is expressed correctly. An example of an equivalence class in the small scenario network of Figure 2 is the set $\{S_2, S_{12}, S_{13}\}$ (see Table VIII); each of these scenarios has the same follow set. The careful analysis that occurs during the construction of a scenario network and the resulting clarifications of and corrections to the scenarios also improve the quality and usefulness of the scenarios and requirements and results in a more complete understanding of how the system is to behave. For example, in the EMS case study, corrections to S_{12} "Subscriber listens to the next message" resulted in the discovery and correction of two requirements errors, an incorrect requirement that archived messages be presented using a separate set of commands, and another error in the requirement for the sequence in which all messages are presented [5].

The analyst's consideration of scenario equivalence during the construction of a scenario network results in specific attention to what each scenario assumes is available and what each scenario assumes is true, and to the results the actions of each scenario produce. Construction of a scenario network necessarily draws attention to gaps that correspond to missing scenarios, to overlaps between scenarios (of the actions they contain, or the results they produce, or in what they assume), and to unsuspected episodes shared by several scenarios or even to the containment of one scenario by another. For example, in the EMS case study, construction of the EMS scenario network identified unintended overlaps between S_{12}, "Subscriber listens to the next message", and five related scenarios, all of which originally advanced the "current message" to be the next message but only some of which should have, resulting occasionally in an advancement beyond the next message. Construction of the scenario network also identified an episode for recording a message that had been roughly duplicated among S_{30}, "Caller calls a subscriber and leaves a message", and six other scenarios.

Gaps in the scenario network usually correspond to missing or incomplete scenarios. The visual metaphor provided by the diagram makes the discovery of such unsuspected scenarios much easier than it would be otherwise. For example, construction of the EMS scenario network uncovered the missing scenario that became S_{13}, "Subscriber has no more messages to listen to." It was found because, as in Figure 2, S_{12}, "Subscriber listens to the next message", could be repeated (as indicated by the arrow from S_{12} to itself) but there was no different behavior specified to occur when S_{12} could no longer be repeated because there were no more messages to hear.

B. Scenario management

Scenario networks provide a rich framework for managing a collection of scenarios. Additionally, the equivalence relations between scenarios that a scenario network supports form a useful means for classifying and linking the scenarios in a collection.

A scenario network provides a temporal scheme for organizing scenarios; every scenario has a place in the network. A scenario's place in the scheme is based upon where it can occur, temporally, relative to the other scenarios. Temporal position is an important aspect of the scenario from the analyst's point of view. Thus it produces a better organization than one based on a more arbitrary aspect, such as the order in which the scenarios were created. Organizing scenarios according to the order in which they were identified fails to provide insights into meaningful relationships, such as sequencing and concurrency. In contrast, scenario networks highlight missing scenarios and duplication between scenarios in a way that analysts find natural and useful, lending itself to methodical validation via walkthroughs (discussed in Section VI-A).

A limitation of organizing scenarios with a scenario network is that this organization does not linearize easily; its natural topology is that of a graph. Thus a scenario network does not translate straightforwardly into a table of contents, for example.

An assortment of relations between scenarios arise during the construction of a scenario network or are indicated by the structure of a scenario network. The most immediately useful of these are equivalence relations, which indicate scenarios that are equivalent in some sense. Examples are the relation that groups together scenarios that can follow all of the same scenarios (illustrated by Table VIII); or scenarios that can precede all of the same scenarios; or scenarios that can both follow the same scenarios and precede the same scenarios. Formally, each of these equivalence

relations partitions a collection of scenarios into disjoint equivalence classes; the scenarios in each class are interchangeable in the sense of the relation, and every scenario is in exactly one class. Informally, each relation groups scenarios that are similar in a particular way. As an example, Table VIII presents the four equivalence classes for the "follow-equivalence" relation applied to the example scenario network of Figure 2 and Table V. In this example, S_{12}, S_{13}, and S_{29} form an equivalence class because each of them can follow only S_2, S_{12}, or S_{13}. Each of the eight scenarios appears in exactly one equivalence class.

Equivalence class		Can follow
{ S_0 }	:	$\emptyset$
{ S_1, S_2, S_{30} }	:	S_0
{ S_{12}, S_{13}, S_{29} }	:	S_2, S_{12}, S_{13}
{ S_{39} }	:	S_{30}

TABLE VIII
FOLLOW-EQUIVALENCE CLASSES OF EXAMPLE SCENARIO NETWORK

These equivalence classes of scenarios are useful in scenario management. Any change to one scenario in a class is likely to be needed for the other scenarios in the class, or at least is more likely to be needed than for a scenario unrelated to the changed one. The scenarios in a class are consistent in some way (or should be if they are not). Specifically, scenarios that are precede-equivalent produce the same or similar results, and scenarios that are follow-equivalent require the same or similar things in order to achieve their results. For example, looking at Table VIII we can see that a change in scenario S_{12} may also require corresponding changes in the scenarios S_{13} and S_{29} that are follow-equivalent to it, and that the three scenarios have the same or similar prerequisites. In fact, all three scenarios require that the subscriber in question already be authenticated to the EMS by his or her passcode, with S_{12} additionally requiring that this subscriber have more messages to hear and S_{13} additionally requiring that he or she have no more messages.

VII. Discussion and Future work

In this paper we have demonstrated that scenario networks provide a form in which to express important specification information that is typically missing from a collection of scenarios. The graphical form of a scenario network provides a compact and useful summary of the temporal relationships between the scenarios describing a system. The structure of a scenario network forms a basis for determining significant relationships between scenarios.

In this paper we have used scenario pre- and postconditions, as dynamic restrictions on each scenario's follow and ramification sets. Another approach which we are considering is to assign *full conditions* to each scenario, conditions that are detailed enough that they define the follow sets, rather than merely restrict them. Then for two scenarios S_A and S_B in the same sequential branch, S_B is in S_A's follow set if $(\mathrm{Post(S_A)} \wedge \mathrm{Pre(S_B)})$ is satisfiable, that is if both the postcondition of S_A and the precondition of S_B are true for some set of values of the primitive terms of the network. A full postcondition is a more complete statement of the effects of a scenario, and a full precondition more completely states its prerequisites. A set of scenarios with full conditions is a partial form (sequential only) of the diagram of a scenario network and its tabular form, and scenario relationships may be derived directly from the full conditions.

We have identified several areas of future work with regard to full conditions. Determining correct full conditions for a set of scenarios has proven to be a cumbersome task, and verifying and debugging them has been challenging. Decidability and efficiency are also issues. The language in which the full conditions are expressed must be carefully restricted so that, for example, satisfiability of pairs of conditions is at least decidable, and preferably reasonably efficient. In trials, we were able to express all conditions in either propositional logic or first-order logic with quantification restricted to only universal quantification over finite sets (e.g. equivalent to the conjunction of a finite number of propositional logic formulae), but in practice this may not always suffice. We considered the use of Presburger formulas as an alternative [42]. Full conditions imply the sequential relationships of a scenario network, by means of satisfiability, but we have not found a satisfactory way to infer ramification of new concurrent branches from full conditions. We are considering the extent to which ramification is tied to the creation of new instances of a network's primitive terms, which appears to hold for some systems we have considered. A second approach is to consider the scope of primitive terms across the scenarios. We have given all primitive terms global scope so far, but an alternative we are considering is to tie scope to ramified branches for some primitive terms. Full conditions also imply all the relationships in Section IV, by the redefinition of *can be followed by* ($\rightharpoonup$) in terms of satisfiability of conditions, either across concurrency or restricted to be within a concurrent sequence as described above. Defining these relationships in terms of conditions may produce more flexible and interesting relationships, or relationships more directly tied to behavior rather than only to sequence, or other benefits. Some relationships that can be defined in terms of full conditions are listed in Table IX. And the *behavioral subtyping* relationship ($<:$) defined in Section IV requires full conditions in order for its definition to be satisfactory. Finally, from the point of view of an analyst working on a scenario network, the additional complexity added to conditions to express a sequential relationship between several scenarios seems disproportionately burdensome compared to the simplicity of drawing an arrow from one scenario to another and using conditions only to restrict the arrows dynamically. Until this practical issue is resolved effectively, full conditions can only be of theoretical interest.

In the work presented here, we have chosen to represent concurrency in a way that emphasizes the similarities between the relationship between a scenario and a "tail"

S_A *can be concurrent with* S_B
S_A *conflicts with* S_B
S_A *allows* S_B
S_A *prevents* S_B

TABLE IX
SOME SCENARIO RELATIONSHIPS FOR FULL CONDITIONS

of others that follow it in a single sequence, and the relationship between a scenario and a new concurrent thread of scenarios that begins after it, and have made this basic to our definitions of the equivalence and ordering relationships that arise in a scenario network. More generally, we have chosen to concentrate on scenario networks all of whose nodes are single scenarios, rather than extending to nodes that are themselves scenario networks. In each of these two cases the alternative of composition of scenario networks remains as future work. Parallel composition of two or more scenario networks is another representation of concurrency, more in line with the way concurrency is represented in Statecharts and other modelling formalisms and in programming language semantics, and which may produce benefits in other kinds of analyses of scenarios in networks. Hierarchical composition of scenario networks, in which a node of a scenario network can be either an individual scenario or another scenario network, is of practical interest as a means of attacking the challenge of scalability of scenario networks. Both kinds of composition are part of our future work.

We have restricted the scenarios in a scenario network to have a single entry point and a single exit point, or more accurately to have the single precondition and single postcondition that are all a single entry and exit require. An extension of this is to allow scenarios with more than one exit point, each with its own postcondition. This extension is analogous to use cases, which can include variant sequences of events. Multiple exit points for each scenario would allow us to move some complexity out of the connections of the network and into scenarios, and support a more natural expression of closely related event sequences, especially appropriate for scenarios that can encounter undesired events causing a chain of events different from the normal one. This extension is linked to hierarchical composition of scenario networks, because a scenario network already is allowed to have more than one terminal scenario, each of which (if the network were masquerading as a scenario of a higher-level scenario network) could have a distinct postcondition.

In summary, scenario networks integrate a collection of scenarios into a single entity that specifies the larger-than-a-scenario behavior that is typically missing from a collection of scenarios. The integration makes gaps in the collection of scenarios obvious, or indicates no gaps are present. Scenario networks provide a basis for several new relationships between scenarios, including equivalence and subtype relations based on the order in which scenarios can occur. Process guidance for scenario creation is an important benefit of the use of scenario networks. Scenario networks support scenario management by providing an organizing structure for a system's scenarios, and scenario relationships that indicate dependencies among scenarios.

REFERENCES

[1] Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke, and Peter Haumer, "Scenarios in system development: Current practice," *IEEE Software*, vol. 15, no. 2, pp. 34–45, Mar./Apr. 1998.

[2] 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*, June 1999, pp. 142–149.

[3] Colin Potts, Kenji Takahashi, and Annie I. Antón, "Inquiry–based requirements analysis," *IEEE Software*, vol. 11, no. 2, pp. 21–32, Mar. 1994.

[4] 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*, 1998, pp. 148–155.

[5] 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.

[6] Annie I. Antón, Michael McCracken, and Colin Potts, "Goal decomposition and scenario analysis in business process reengineering," in *Proceedings of the 6th International Conference on Advanced Information Systems Engineering (CAiSE'94)*, 1994, pp. 94–104.

[7] 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)*, Apr. 1998, pp. 157–166.

[8] Matthias Jarke, X. Tung Bui, and John M. Carroll, "Scenario management: An interdisciplinary approach," *Requirements Engineering Journal*, vol. 3, no. 3–4, pp. 155–173, 1998.

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

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

[11] Alistair G. Sutcliffe, Neil A. M. Maiden, Shailey Minocha, and Darrel Manuel, "Supporting scenario-based requirements engineering," *IEEE Transactions on Software Engineering*, vol. 24, no. 12, pp. 1072–1088, Dec. 1998, Special Issue: Scenario Management.

[12] Colette Rolland and Camille Ben Achour, "Guiding the construction of textual use case specifications," *Data and Knowledge Engineering Journal*, vol. 25, no. 1–2, pp. 125–160, Mar. 1998, Also published as CREWS technical report 98-01.

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

[14] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein, "A framework for expressing the relationships between multiple views in requirements specification," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 760–773, Oct. 1994.

[15] 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*, 1997, pp. 54–64.

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

[17] R. J. A. Buhr, "Use case maps as architectural entities for complex systems," *IEEE Transactions on Software Engineering*, vol. 24, no. 12, pp. 1131–1155, Dec. 1998.

[18] 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*, 1999, pp. 44–53.

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

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

[21] Colette Rolland, Carine Souveyet, and Camille Ben Achour, "Guiding goal modeling using scenarios," *IEEE Transactions on Software Engineering*, vol. 24, no. 12, pp. 1055–1071, Dec. 1998.

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

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

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

[25] 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*, Gaithersburg, MD, 1994, pp. 3–8, National Institute of Standards and Technology.

[26] 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*, May 1990, pp. 104–108.

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

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

[29] Jules Desharnais, Marc Frappier, Ridha Khédri, and Ali Mili, "Integration of sequential scenarios," *IEEE Transactions on Software Engineering*, vol. 24, no. 9, pp. 695–708, Sept. 1998.

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

[31] 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*, vol. 15, no. 1, pp. 36–72, Jan. 1993.

[32] 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*, vol. 8, no. 2, pp. 244–263, Apr. 1986.

[33] James L. Peterson, "Petri nets," *ACM Computing Surveys*, vol. 9, no. 3, pp. 223–252, Sept. 1977.

[34] Ekkart Rudolph, Peter Graubmann, and Jens Grabowski, "Tutorial on Message Sequence Charts," *Computer Networks and ISDN Systems*, vol. 28, no. 12, pp. 1629–1641, June 1996.

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

[36] Roel Wieringa, "A survey of structured and object-oriented software specification methods and techniques," *ACM Computing Surveys*, vol. 30, no. 4, pp. 459–527, Dec. 1998.

[37] David Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, June 1987.

[38] David Harel and Amnon Naamad, "The STATEMATE semantics of statecharts," *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 4, pp. 293–333, Oct. 1996.

[39] Rolv Bræk, "SDL basics," *Computer Networks and ISDN Systems*, vol. 28, no. 12, pp. 1585–1602, June 1996.

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

[41] Martín Abadi and Luca Cardelli, *A theory of objects*, Springer-Verlag, New York, 1996.

[42] William Pugh, "The Omega test: a fast and practical integer programming algorithm for dependence analysis," in *Proceedings Supercomputing'91*, Nov. 1991, pp. 4–13, Reproduced in Comm. of the ACM, vol. 35, No. 8, August 1992, pp. 102-114. Retitled: A Practical Algorithm for Exact Array Dependence Analysis.