An Integrated Scenario Management Strategy

Thomas A. Alspaugh, Annie I. Antón, Tiffany Barnes and Bradford W. Mott

{taalspau,aianton,tmbarnes,bwmott}@eos.ncsu.edu

College of Engineering
North Carolina State University
Raleigh, NC 27695-7534, USA

Abstract

Scenarios have proven effective for eliciting, describing and validating software requirements; however, scenario management continues to be a significant challenge to practitioners. One reason for this difficulty is that the number of possible relations among scenarios grows exponentially with the number of scenarios. If these relations are formalized, they can be more easily identified and supported. To provide this support, we extend the benefits of project-wide glossaries with two complementary approaches. The first approach employs shared scenario elements to identify and maintain common episodes among scenarios. The resulting episodes impose consistency across related scenarios and provide a way to visualize their interdependencies. The second approach quantifies similarity between scenarios. The resulting similarity measures serve as heuristics for finding duplicate scenarios, scenarios needing further elaboration, and scenarios which have not yet been identified, yielding valuable information about how well the scenarios provide coverage of the requirements. These two approaches, integrated with a scenario database, project glossaries, configuration management, and coverage analysis, form the basis of a useful and effective strategy for scenario management and evolution.

1 Introduction

Developers use scenarios to validate their understanding of customers' work practices, organizational goals and system requirements. Scenarios are receiving widespread attention in various intellectual communities such as object-oriented development [8, 13, 16, 24, 25], human-computer interaction [11], and strategic planning [10]. In the requirements engineering community, scenarios have proven effective for discovering [4, 18, 22, 23], elaborating [16, 18], refining [27], and validating [1, 7, 15] requirements. A recent study of scenario usage in industrial projects, [29], highlights the increasing need for scenario management strategies.

This paper proposes an integrated strategy for scenario management that formalizes similar scenario structures and attributes to guide and facilitate the requirements analysis process. We combine our methods with the Goal-Based Requirements Analysis Method (GBRAM) [2] to form a more powerful goal-based scenario management tool. This tool will extend the functionality of the Goal-Based Requirements Analysis Tool (GBRAT) [3] to include our strategies for glossary creation, episode management, similarity measures, and coverage analysis. While this work has not been formally validated, we believe the strategy is relevant, effective, and feasible.

In the remainder of this section, we address some of the challenges and problems inherent in scenario management and introduce our terminology. Section 2 describes our scenario management strategy. In Section 3 we illustrate how the strategy is applied within the context of the meeting scheduler problem. Finally, in Section 4 we discuss our plans for future work and tool support.

1.1 Background and Related Work

An effective scenario management strategy must address several practical issues, which we discuss below in the context of some related work.

Similarity measures for requirements reuse

Similarity measures have been used extensively for requirements reuse; for example, *semantic similarity* has been used with conceptual graphs [26], analogical reasoning on a pre-existing case base [21], and on generic domain models [20]. A semantic measure is appropriate for these applications because they reapply the semantic structure of pre-existing requirements to new contexts. For our scenario management strategy, we believe *syntactic identity* is more appropriate. Our strategy reuses entire scenarios or attributes, rather than adapting them to new contexts; syntactic comparisons are easier to understand and eliminate the need to build a semantic structure.

Goals as an organizing structure for scenarios

Goal hierarchies are useful for representing relationships between goals and subgoals and for reasoning about goal relationships [4, 12]. The goal topography in the GBRAM [2] uses a goal hierarchy to structure and orga-

nize requirements information (i.e., scenarios, goal obstacles, constraints and auxiliary notes). Goal topographies help analysts find information and sort goals into naturally different functional requirements. In GBRAM, scenarios serve to document issues, identify new goals, and elaborate requirements. Preliminary evidence [2, 3] suggests that it is feasible to visualize goals and their related scenarios via a goal hierarchy. Maiden *et al.* have also proposed organizing scenarios using goals and goal obstacles [19]. While goals and goal hierarchies are promising, their use is not sufficient as a scenario management strategy. This observation, coupled with previous experiences [5], have led us to further specify the requirements for scenario management tool support.

Support for scenario evolution

Requirements continually evolve, increasing the need to track open issues. A recent goal-based analysis [5] highlighted the evolutionary nature of goals, scenarios, and requirements. The scenario evolution framework of [9], suggests two types of changes that affect scenarios during development: "inter-scenario" evolution, where changes involve a set of scenarios, and "intra-scenario" evolution, where changes concern a single scenario. We present a viable strategy to manage the relationships in both inter- and intra-scenario evolution. Our strategy is distinct from that of Breitman and Leite [9]; our strategy extends scenario relationships to include goals, and supports the automatic identification of related scenarios.

Process guidance

In 15 projects studied by Weidenhaupt *et al.*, most project developers lacked appropriate project guidance [29]. The CREWS-SAVRE tool [19] addresses this need for guidance by providing a wizard to enable and support the systematic walkthrough of generated scenarios. It proposes generic requirements statements for inclusion in a requirements specification. Similarly, we will provide process guidance using GBRAM [2] and our episode and coverage heuristics to guide goal and scenario analysis. The GBRAM heuristics guide goal identification, classification, refinement, and elaboration by considering possible obstacles and constructing scenarios to uncover hidden goals and requirements. The inclusion of such heuristics in a tool will enable analysts and stakeholders to play a more active role in constructing scenarios. The proposed episode management, similarity measure, and coverage analysis strategies extend this active role in scenario development by offering heuristics for scenario identification and elaboration.

1.2 Terminology

We define the following key terms:

- A *scenario* is a linear sequence of events, with associated attributes. Note that common scenario representations can be abstracted to this form.
- An *event* consists of an actor and an action.
- A *subsequence* is a sequence of one or more events that forms all or part of a scenario's sequence.
- An *episode* is a named subsequence that is usually shared among several scenarios. It may have other attributes, like a scenario, and in fact, it may be a scenario; but in its role as an episode everything other than its sequence of events is ignored.
- The *attributes* of a scenario may include: system goals, viewpoint, pre- and post-conditions, purpose, concreteness level, author, requirements, events, actors and actions, as well as episodes.

2 Scenario Management Proposal

This section provides an overview of the five main components of our scenario management strategy: tool support, glossaries, episode management, similarity measures, and coverage estimation.

2.1 The Case for a Scenario Management Tool

Tool support for scenario management is important for both scenario developers and project managers [29]. Scenario developers need a tool for quick and easy entry of scenario data, with automatic checking for redundancy and glossary term creation, requiring minimal time to examine other scenarios for consistency. Project managers need a tool to allow variability in redundancy checking, so tradeoffs can be made between consistency and ease of use on an individual project basis. Both developers and managers need a tool that can identify similar scenarios and examine redundancy, consistency, and coverage.

The major components of our proposed scenario management tool are: 1) a scenario database, 2) individual glossaries for each scenario attribute, 3) a configuration management system for time-stamping and documenting changes in the scenario and glossary databases, 4) episode identification and management, 5) similarity measures, and 6) system coverage analysis. Our strategies extract attribute lists from any common scenario representation to perform episode management, measure scenario similarities, and investigate system coverage. The glossaries, built in parallel with scenario creation, provide the foundation for consistent attribute descriptions and comparisons, simplifying both system terminology and management heuristics. The configuration management system supports both scenario evolution and coverage analysis. Together with the scenario database, glossaries, and configuration management system, our strategies for episode management, similarity measures, and coverage analysis form the basis for a powerful scenario management tool.

2.2 Support for Consistency: Glossaries

The benefits associated with consistency checking in requirements specification are highlighted in [15]. Automated support for consistency checking frees analysts from a time-consuming and error-prone process. In four of the studies discussed in [29], project glossaries helped stakeholders establish a common understanding of scenario terms. In an industrial case study where 88 scenarios were used to define requirements, stakeholders also noted the need to define general and domain-specific terms [14].

In addition to providing a repository of terms, a project glossary can help search for and prevent redundancies, which become a problem as the number of scenarios in a system increases. However, building glossaries can be burdensome and time-consuming. To reduce this burden, our tool will form attribute glossaries in parallel with the scenario database, allowing analysts to browse the glossaries, check for consistency and redundancy, and add terms during the analysis process.

Different scenario attributes require different levels of consistency and redundancy checking, according to the specific system and characteristics of the attribute itself. For example, actors are easy to list, but actions may be very scenario or event-specific, so that checking for redundancy during data entry is time-consuming and usually unproductive. Instead, we can perform redundancy checks on an "as-needed" basis, such as during the search for episode matches, as discussed below.

2.3 Episode Management

Episodes, sequences of events shared by two or more scenarios, play a significant role in effective scenario management [23]. To manage episodes, a system must be able to distinguish events and recognize those that are identical. Two events are identical if they share the same actor and action. The set of actors for a system is well-defined; hence, a tool can readily recognize identical actors in two or more events. However, the set of actions may not be so well-defined; if so, the intervention of an analyst is needed to recognize identical actions.

We envision several ways to support the identification of equivalent actions. In each case presented below, the tool presents pairs of potentially identical actions, the analyst determines whether they are identical or distinct, and the tool records the decision for future use.

1. *The program presents only the pairs that the analyst asks for.* The analyst chooses which pairs to examine, but for a thorough comparison, many pairs must be examined.
2. *The program presents potentially identical pairs whose textual descriptions are similar.* The analyst may have to examine many pairs, but fewer than in the first method.
3. *The program presents only potentially identical pairs of actions that, if identical, would make two events identical.* The textual descriptions of the actions must be similar, and the actions must appear in two events sharing the same actor. The analyst examines fewer pairs and many actions will not need to be examined.
4. *The program presents only potentially identical pairs of actions that, if identical, might make two subsequences of n or more events identical.* In addition to the three previous conditions, the events that the actions appear in must be part of two shared subsequences of length $n \geq 2$ that are potentially identical. n may be adjusted to reduce the number of presented pairs as desired.

Once the episodes are identified and recorded in the scenario database, several helpful features are possible:

- Visible marking of episodes when the events of a scenario are displayed.
- Identification of other scenarios sharing each episode, with easy navigation from the present scenario to each of its relatives.
- Two views of scenarios that share an episode: individually, or simultaneously with one shown as a variant of the other.
- A warning when an event in an episode is edited.
- Presentation of a choice when editing events in an episode: "change all the scenarios that share this episode," or "detach this scenario from the others and change it, leaving the others the same."

The value of these features increases with the size of a project; as the numbers of both scenarios and developers increase, it becomes more difficult to track scenario redundancy and dependencies. Episodes explicitly represent these relationships, allowing analysts to monitor and control inter- and intra-scenario evolution. In addition, managing decisions about scenario consistency and dependency can prevent and reduce errors in the scenario database. This early prevention of errors can save money and time in the requirements analysis process: the earlier errors are caught, the easier and more inexpensive they are to fix. Thus, our episode management strategy improves requirements analysis by partially automating event comparisons, error prevention, and dependency tracking, resulting in a faster, more reliable scenario development process.

2.4 Similarity Measures

A *similarity measure* is a function that produces a number expressing the degree of similarity between two scenarios. 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.

We consider only the syntactic representation of the scenario; that is, our measure takes into account 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 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.

The similarity measure has several important properties. First, its values are scaled to lie between 0 and 1, so that similarity measures can be compared across the system. 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 the attribute glossaries 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) of attribute values, and examines these sets for overlap. 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 sum of the number of common attribute values in each attribute list, divided by the sum of the sizes of each attribute list, (see [28] for other ratio models):

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

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.

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. 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 [17]. It can also help identify accidental duplication in

a large set of scenarios. Using the similarity measures, our proposed tool would 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.

2.5 Estimating Coverage

We predict that any similarity measure, such as those presented in this paper, demonstrating Tversky's properties of matching, monotonicity, and independence [28], can be used to estimate the coverage of a group of scenarios.

As the group of scenarios expands to more nearly cover the system's requirements, we expect to reach a balance between similar and distinct scenarios, reflecting both the diversity and elaboration that characterize a fully developed requirements specification. This balance would be reflected by a fairly low similarity measure for the entire system, arising from both groups of highly similar scenarios demonstrating the exploration of particular system aspects, and groups of distinct scenarios demonstrating the exploration of different requirements areas. When the similarity measure seems to have stabilized at a balanced point, the scenarios may be reaching sufficient coverage of the requirements and/or goals.

Throughout the development process, we monitor system coverage using a histogram of similarity measure values for scenario pairs versus the number of pairs with those values. A high number of high-similarity pairs indicates that some areas of the system requirements have been well-explored, but that other areas remain uncovered. In this case, there is not yet enough diversity in the scenarios for requirements coverage. The tool would then indicate that scenarios need to be developed in new system areas.

Conversely, if the histogram reflects large numbers of highly distinct pairs of scenarios, there is a rich diversity in areas explored, but a shallowness of depth in their exploration. In this case, the tool would indicate that existing areas of the system need further elaboration. Hence, we predict that systems with insufficient coverage will have a histogram heavily weighted towards high numbers of pairs with high similarity, or high numbers of pairs with low similarity. In contrast, a histogram with a balanced distribution of similarity values, reflecting both diversity and in-depth exploration, would represent more complete coverage.

These conjectures about system coverage, if correct, would provide a powerful way to monitor system evolution. A scenario management tool using our measures could display similarity measure histograms and evolutionary changes in similarity for the entire system, particular attributes, and for different system components. Thus, in addition to their usefulness as searching tools, the similarity measures defined in this paper can be used to examine overall system coverage, providing guidance in scenario elaboration and elicitation.

3 Example

As an illustration, we apply our strategy to the meeting scheduler problem [23]. The purpose of the meeting scheduler is to facilitate the organization of meetings. Specifically, for each meeting request it schedules a meeting time and place such that most participants can attend. Each meeting has an *initiator*, who calls for the meeting, as well as *ordinary*, *active*, and *important* participants.

When scheduling a meeting, the initiator proposes time constraints for the meeting and the participants respond with times during which they are and are not available to meet (indicated as *preference* and *exclusion* sets, respectively). Based on the exclusion and preference sets, the scheduler attempts to schedule the meeting. Sometimes it succeeds and sometimes, due to conflicts, it fails.

A scenario-based analysis of the meeting scheduler [23] yielded a number of scenarios, four of which are reproduced in Figures 1 through 4 with minor annotations, (e.g. the addition of scenario purpose).

Scenario 1

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 preferences
7.	Scheduler	Send appropriate mail messages to participants
8.	Ordinary participant	Respond with exclusion and preference sets
9.	Active participant	Respond with exclusion and preference sets as well as equipment requirements
10.	Scheduler	Request required equipment
11.	Important participant	Respond with exclusion and preference sets as well as location preference
12.	Scheduler	Schedule meeting on the basis of responses, policies, and room availability
13.	Scheduler	Send confirmation message to all participants and meeting initiator

Figure 1: Meeting Scheduler S_1 - no conflicts

Episode management

Effective scenario management requires identifying and maintaining episodes in a system. Table 1 lists the potential episodes discovered as shared subsequences in Scenarios 1 through 4. Our proposed tool can display these subsequences graphically, and interact with analysts to determine which ones to mark as episodes.

Scenario 2

Name: Schedule Meeting with Slow Responder (ordinary participant)
Purpose: Requirements elaboration
Developer: Annie Anton
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 preferences
7.	Scheduler	Send appropriate mail messages to participants
8.	Active participant	Respond with exclusion and preference sets as well as equipment requirements
9.	Scheduler	Request required equipment
10.	Important participant	Respond with exclusion and preference sets as well as 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 meeting initiator

Figure 2: S_2 - ordinary participant is a slow responder

Scenario 3

Name: Schedule Meeting with Slow Responder (important participant)
Purpose: Goal obstacle analysis
Developer: Kenji Takahashi
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 preferences
7.	Scheduler	Send appropriate mail messages to participants
8.	Ordinary participant	Respond with exclusion and preference sets
9.	Active participant	Respond with exclusion and preference sets as well as equipment requirements
10.	Scheduler	Request required equipment
11.	Scheduler	Recognizes that timeout has expired and reminds late participant
12.	Scheduler	Recognizes that drop-dead date has passed
13.	Scheduler	Notifies initiator that important participant has not responded
14.	Initiator	Decides to cancel meeting
15.	Scheduler	Release reserved equipment and room
16.	Scheduler	Send cancellation message to all participants

Figure 3: S_3 - important participant is a slow responder

Consider an example situation: an analyst creates a "Meeting Initiation" episode for events 1-7 in S_1, S_2, and S_3. Since events 14 in S_3 and 1 in S_4 share the same actor and similar actions that could compose part of a 3-event episode, the tool presents this pair of actions to the analyst for comparison. The analyst chooses to identify these

Scenario 4

Name: Cancel meeting
Purpose: Requirements elaboration
Developer: Annie Anton
Goals: ACHIEVE Meeting Canceled

Events:

No.	Actor	Action
1.	Initiator	Request meeting cancellation
2.	Scheduler	Release reserved equipment and room
3.	Scheduler	Send cancellation message to all participants

Figure 4: S_4 - meeting is canceled

Scenario Groups	Potential Episodes
1,2	(1-7, 1-7) (9-11, 8-10) (12-13, 13-14)
1,3	(1-7, 1-7) (8-10, 8-10)
2,3	(1-7, 1-7) (8-9, 9-10) (11-12, 11-12)
3,4	(14-16, 1-3)
1,2,3	(1-7, 1-7, 1-7) (9-10, 8-9, 9-10)

Table 1: Potential episodes

Scenario 3

Name: Schedule Meeting with Slow Responder (important participant)
Purpose: Goal obstacle analysis
Developer: Kenji Takahashi
Goals: ACHIEVE Meeting scheduled

Events:

No.	Actor	Action
1. Episode: Meeting Initiation		
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 preferences
7.	Scheduler	Send appropriate mail messages to participants
2.	Ordinary participant	Respond with exclusion and preference sets
3.	Active participant	Respond with exclusion and preference sets as well as equipment requirements
4.	Scheduler	Request required equipment
5.	Scheduler	Recognizes that timeout has expired and reminds late participant
6.	Scheduler	Recognizes that drop-dead date has passed
7.	Scheduler	Notifies initiator that important participant has not responded
8. Episode: Meeting Cancellation		
1.	Initiator	Request meeting cancellation
2.	Scheduler	Release reserved equipment and room
3.	Scheduler	Send cancellation message to all participants

Figure 5: S_3 - with identified episodes

two events and create a "Meeting Cancellation" episode for events 14-16 in S_3 and 1-3 in S_4. Using these episodes, the new structure of S_3 is shown in Figure 5.

Once episodes are identified, the tool tracks them and

provides visible marking of episodes, warnings when editing events of episodes, and alternative views of scenarios that share episodes. For example, the tool can use branching diagrams to display common episodes, as in Figure 6. Here, S_1 and S_2 share the "Meeting Initiation" episode and then branch. Using such diagrams, analysts can better visualize the episodic relationships among scenarios.

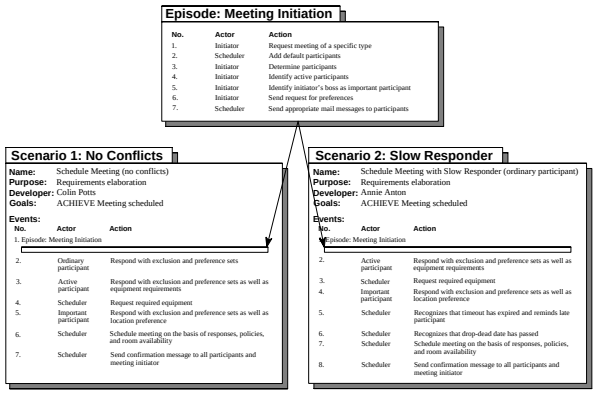

Figure 6: S_1 and S_2 shown with branching

Similarity measures

In this section, we demonstrate the use of the similarity measures. Applying the similarity measure $S(S_1, S_2)$ to the *purpose*, *developer*, *goal*, and *actor* attributes of the scenarios in Figures 1 through 4 results in the following measures:

$$S(S_1, S_2) = 0.800 \qquad S(S_1, S_3) = 0.667$$
$$S(S_1, S_4) = 0.462 \qquad S(S_2, S_3) = 0.571$$
$$S(S_2, S_4) = 0.667 \qquad S(S_3, S_4) = 0.333$$

By using these measures a tool can easily see that S_1 and S_2 are very similar while S_3 and S_4 are not. This coincides nicely with an analyst's view of the scenarios since S_1 and S_2 are in fact very similar while S_3 and S_4 are not.

The weighted similarity measure, $SW(S_1, S_2)$, can be used to emphasize particular attributes over others by assigning them relatively high weights. For example, to emphasize episodic structure when comparing scenarios, we weight episodes with 1 and all other attributes with 0.25, yielding:

$$SW(S_1, S_2) = 0.870 \qquad SW(S_1, S_3) = 0.667$$
$$SW(S_1, S_4) = 0.286 \qquad SW(S_2, S_3) = 0.615$$
$$SW(S_2, S_4) = 0.400 \qquad SW(S_3, S_4) = 0.500$$

These measures reflect the amount of sharing between each pair of scenarios. Note that S_1 and S_2 have a high measure of overlap, while S_1 and S_4 show almost no overlap, which corresponds to our intuitive evaluation of these two pairs.

The extension $S^*(S_1, S_2, ..., S_n)$ computes the average similarity for all the pairs of scenarios in a group. For example:

$$S^*(S_1, S_2, S_3) = 0.679 \qquad S^*(S_1, S_3, S_4) = 0.487$$

Clearly, the group containing S_1, S_2 and S_3 is more similar than the group containing S_1, S_3 and S_4. In this case, the measure serves as a heuristic for grouping scenarios based on their similarity without requiring analysts to read through each scenario.

The general similarity measure, $SW^*(S_1, S_2, ..., S_n)$, provides a weighted group comparison. For example, to measure the overall amount of episode usage, we weight episodes with 1 and all other attributes with 0, yielding:

$$SW^*(S_1, S_2, S_3, S_4) = 0.500$$

indicating that on the average two scenarios in this group share half their episodes.

These examples demonstrate some simple applications of the measures and suggest how the resulting values begin to provide valuable information about scenario coverage.

4 Discussion and Future Work

Current approaches to scenario management provide frameworks within which to discuss scenarios [29]. The CREWS framework [29] classifies scenarios according to four facets: Purpose, Lifecycle, Contents and Form. Our integrated scenario management strategy indirectly supports each of these aspects of scenarios. More recently, a representational scenario framework has been proposed [6]. Our strategy is not constrained or limited to any one representation alternative. Instead, it was devised to be applicable to extensible scenario attributes while allowing the use of common scenario representations.

This paper goes beyond previous work by providing a concrete, implementable strategy to manage the large and at times unwieldy amounts of information associated with scenarios. The proposed strategy allows analysts to vary the level of redundancy and consistency checking required for scenario evolution. The GBRAM includes heuristics for the identification of redundant and synonymous goals and for their elimination and reconciliation, respectively. However, the strategies discussed in this paper extend these types of heuristics to other scenario attributes, such as episodes. When applied to various attributes, these heuristics facilitate the visualization of scenarios and their interrelationships. Coupled with the heuristics available in GBRAM [2], the similarity measures are a rich information source regarding requirements coverage. The weighted similarity measures, in particular, support the meaningful tracking of change in similarity throughout scenario evolution. This allows us to provide constructive process guid-

ance (e.g. stopping criteria for scenario construction) to requirements practitioners.

Although our preliminary studies suggest this integrated strategy is feasible and effective, our investigations have entailed manual calculations and observations. We realize there are possible risks associated with depending on a tool to, for example, find similarities and common episodes. Another issue to be addressed is the potential for other similarities to exist that the tool may not identify. How do we identify these and what do they look like? Our immediate plans involve further investigation of these issues. We plan to apply the strategy to several large sets of scenarios available from case study repositories, and build extensions to the strategy as we investigate and address such risks. A prototype to support the proposed algorithms and strategies is currently being developed. We seek to answer questions about how effective these strategies are in providing process guidance, supporting scenario evolution, and using goals to both structure scenarios and aid in ensuring coverage of system requirements.

Acknowledgements

The authors wish to thank Don Bitzer, Axel van Lamsweerde, and the anonymous reviewers for their invaluable comments.

References

[1] J. Anderson and B. Durney. Using Scenarios in Deficiency-driven Requirements Engineering. In *IEEE Intl. Symposium On Requirements Engineering*, pages 134–41, Jan. 1993.

[2] A. Antón. *Goal Identification and Refinement in the Specification of Software-Based Information Systems*. PhD thesis, Georgia Insitute of Technology, Atlanta, GA, USA, 1997.

[3] A. Antón, E. Liang, and R. Rodenstein. A Web-Based Requirements Analysis Tool. In *5th Work. on Enabl. Technol.: Infrastruct. for Collab. Ent.*, pages 238–243, June 1996.

[4] A. Antón, W. McCracken, and C. Potts. Goal Decomposition and Scenario Analysis in Business Process Reengineering. In *Adv. Info. Sys. Eng.: 6th Intl. Conf.*, pages 94–104, June 1994.

[5] A. Antón and C. Potts. The Use of Goals to Surface Requirements for Evolving Systems. In *20th Intl. Conf. on Software Engineering (ICSE98)*, pages 157–166, Apr. 1998.

[6] A. Antón and C. Potts. A Representational Framework for Scenarios of System Use. *Requirements Engineering Journal*, to appear, 1999.

[7] K. Benner, M. Feather, W. Johnson, and L. Zorman. Utilizing Scenarios in the Software Development Process. In *IFIP WG 8.1 Work. Conf. on Info. Sys. Dev. Proc.*, Dec. 1992.

[8] G. Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 1994.

[9] K. Breitman and J. Leite. A Framework for Scenario Evolution. In *Int. Conf. on Req. Eng.*, pages 214–21, Apr. 1998.

[10] T. Bui, G. Kersten, and P.-C. Ma. Supporting Negotiation with Scenario Management. In *Proc. of the 29th HICSS*, volume 3, pages 209–18, Jan. 1996.

[11] J. Carroll and M. Rosson. Getting Around the Task-Artifact Cycle: How to Make Claims and Design by Scenario. *ACM Trans. on Information Systems*, 10(2):181–212, Apr. 1992.

[12] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 20(1-2):3–50, Apr. 1993.

[13] M. Fowler. *UML Distilled: Applying the Standard Object Modeling Language*. Addison Wesley, 1997.

[14] P. Gough, F. Fodemski, S. Higgins, and S. Ray. Scenarios - An Industrial Case Study and Hypermedia Enhancements. In *2nd Intl. Symp. on Req'ts Eng.*, pages 10–17, Mar. 1995.

[15] C. Heitmeyer, B. Labaw, and D. Kiskis. Consistency Checking of SCR-Style Requirements Specifications. In *2nd Intl. Symp. on Req'ts Eng.*, pages 56–63, Mar. 1995.

[16] I. Jacobson. *Object-Oriented Software Engineering: A Use-Case Driven Approach*. Addison-Wesley, 1992.

[17] D. Lauzon and T. Rose. Task-oriented and similarity-based retrieval. In *9th Knowledge-Based Software Engineering Conf.*, pages 98–107, Sept. 1994.

[18] M. Lubars, C. Potts, and C. Richter. Developing Initial OOA Models. In *15th Int. Conf. S/W Eng.*, pages 255–264, 1993.

[19] N. Maiden, S. Minocha, K. Manning, and M. Ryan. CREWS-SAVRE: Systematic Scenario Generation and Use. In *Intl. Conf. on Req'ts Eng.*, pages 148–155, Apr. 1998.

[20] N. Maiden and A. Sutcliffe. Analogical retrieval in reuse-oriented requirements engineering. *Software Engineering Journal*, 11(5):281–292, Sept. 1996.

[21] P. Massonet and A. van Lamsweerde. Analogical reuse of requirements frameworks. In *Proc. of the Third IEEE Intl. Symp. on Requirements Engineering*, pages 26–37, Jan. 1997.

[22] C. Potts. Using Schematic Scenarios to Understand User Needs. In *Symp. on Des. Interactive Sys.: Processes, Practices, Methods, and Techniques*, pages 247–56, Aug. 1995.

[23] C. Potts, K. Takahashi, and A. Antón. Inquiry-Based Requirements Analysis. *IEEE Software*, 11(2):21–32, Mar. 1994.

[24] K. Rubin and A. Goldberg. Object Behavior Analysis. *Communications of the ACM*, 35(9):48–62, Sept. 1992.

[25] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, New York, NY, 1991.

[26] K. Ryan and B. Mathews. Matching conceptual graphs as an aid to requirements re-use. In *Proc. of the IEEE Intl. Symp. on Requirements Engineering*, pages 112–120, San Diego, California, Jan. 1993.

[27] K. Takahashi and Y. Yamamoto. An Analysis of Traceability in Requirements Documents. *IEICE Trans. on Information Systems*, E78-D(4):393–402, 1995.

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

[29] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer. Scenarios in System Development: Current Practice. *IEEE Software*, 15(2):34–45, Mar. 1998.