ISR Institute for Software Research
University of California, Irvine

Relationships Between Scenarios

Thomas A. Alspaugh
University of California, Irvine
alspaugh@ics.uci.edu

May 2006

ISR Technical Report # UCI-ISR-06-7

Institute for Software Research
ICS2 110
University of California, Irvine
Irvine, CA 92697-3455
www.isr.uci.edu

www.isr.uci.edu/tech-reports.html

Relationships Between Scenarios

Thomas A. Alspaugh

Institute for Software Research
Department of Informatics
University of California, Irvine
alspaugh@ics.uci.edu

Abstract. Scenarios are widely used in requirements analysis and other activities, but their informality is a challenge for reasoning about them and providing significant tool support. This research describes an approach for identifying aspects of scenarios that people use consistently, structuring them, and using this structure to support work with scenarios. Our approach clarifies how scenarios can be related (for example by specialization) and how they can be used to give each other context and constraints, and provides a foundation for more extensive automated support. Automated support for scenario manipulation and analysis lets human expertise be concentrated on the tasks that need it most. Our approach is implemented by an XML language and a Java package for it. We describe how they have been used in goal-driven specification-based testing, computed social worlds of autonomous animated agents, and analysis of business rules and scenarios.

Relationships Between Scenarios

Thomas A. Alspaugh

Institute for Software Research
Department of Informatics
University of California, Irvine
alspaugh@ics.uci.edu

Abstract. Scenarios are widely used in requirements analysis and other activities, but their informality is a challenge for reasoning about them and providing significant tool support. This research describes an approach for identifying aspects of scenarios that people use consistently, structuring them, and using this structure to support work with scenarios. Our approach clarifies how scenarios can be related (for example by specialization) and how they can be used to give each other context and constraints, and provides a foundation for more extensive automated support. Automated support for scenario manipulation and analysis lets human expertise be concentrated on the tasks that need it most. Our approach is implemented by an XML language and a Java package for it. We describe how they have been used in goal-driven specification-based testing, computed social worlds of autonomous animated agents, and analysis of business rules and scenarios.

1 Introduction

Scenarios (and use cases containing them) describe uses of a system in terms of situations, interactions between agents, and events unfolding over time. They are widely used in a number of ways during the development process, and by a variety of participants [AM04,BFJZ92]. Stakeholders (customers, users, and others affected by a system) use them to communicate what is wanted, and developers (designers, programmers, testers, etc.) use them to confirm their understanding [BFJZ92,MMMR98,Rob04]. They may be the primary form in which requirements are recorded [Coc00,Hau04,JCJÖ92], a preliminary form from which specialists produce more refined forms such as goals and requirements or behavior models [LW98,UCKM04], a guide and scaffolding in a process by which other artifacts such as goals are developed [RSBA98]. They are used to simulate and explore a system's use [BFJZ92] or a design's utility [Sut03], to present a test or validation [BBL+04], and to derive tests from requirements [ARSZ05].

In all these uses, the informal natural language form of scenarios is a key advantage. Their narrative form takes advantage of people's natural abilities for constructing and understanding stories. However, their informality limits the extent to which automation and tool support can be effective, and makes it challenging to define relations between them. This paper presents an investigation into the ways and degree to which the underlying structure and meaning

of scenarios can be brought to the surface and made use of, without interfering with the advantages of informal scenarios. The investigation has taken place in the context of the definition and implementation of ScenarioML, a language for structuring and marking up the text of scenarios. A fundamental requirement on this language is that it must always be possible to automatically produce an informal natural language text form from any marked up scenario, in order to preserve the key advantage of using scenarios in the first place. ScenarioML satisfies this requirement while also supporting the structuring and marking up that allows automation and tool support.

In the process we have defined and implemented identification of key relationships among scenarios: scenario equivalence and specialization, and implication of one scenario by another. These relationships are defined in terms of our assertion that *the meaning of a scenario is the (usually infinite) set of occurrences it describes in the world.* We have also identified potentially powerful generalizations of ordinary scenario usage, including using one scenario to state the context for others, and independently constraining the behavior one scenario describes by specifying its relationship to a second scenario. Scenarios expressed in this form can be read by a program, analyzed and manipulated internally, and output in the same or another form or used to drive another process. Scenario-specific editors and stronger scenario management tools are the most obvious use of this language. Additional automated uses of scenarios have been explored.

This research focuses on scenarios rather than use cases for several reasons. A use case groups together the scenarios that achieve a particular goal in alternative ways, plus the scenarios that describe associated exceptional cases and actions. The possibilities we have identified reside at the level of an individual scenario, not at the higher level of organization of a use case. Secondly, use cases have a definition and relationships that are widely accepted, if not necessarily strictly defined, so that generalizations and additions are less likely to be of interest. Finally, since use cases consist of scenarios, results from research in scenarios can be incorporated into use cases through that path. We focus on scenarios rather than message sequence charts (MSCs) and similar modeling constructs because those are especially appropriate for architecture- and design-level specification and have received much attention in that context. MSCs and similar constructs are less appropriate for higher-level requirements specification, where the actors are often human beings and the range of possible actions and interactions is broad and less well defined. Scenarios at this level have not been investigated to the extent that MSCs and other design-level formalisms have been, and can benefit from more attention.

The remainder of the paper is organized as follows. Section 2 summarizes related work. Section 3 motivates and presents the core of the ScenarioML language. Section 4 discusses how ScenarioML events are compared and their relations determined. Section 5 discusses its implementation and tool support, and briefly presents applications and validation to date. We outline our future work in Section 6, and summarize and present conclusions in Section 7.

2 Related Work

There has been a substantial amount of work on scenario and use case languages, models, and semantics. Many authors have proposed event schemas in one form or another. Various forms of alternation, iteration, and exception/interruption are described by Dardenne *et al.* [DLF93], the Message Sequence Charts specification [ITU99], Maiden [Mai98], and many others. Basic scenarios have been extended with specific sets of temporal relations between events, including important work by Maiden [Mai98] and by Breitman *et al.* [BLB05]. Episodes under one name or other appear to have been nearly universal in scenario and use case languages, including notably Potts *et al.* [PTA94] and the OMG [OMG03]. The structure of ScenarioML is built squarely on this work. The features of ScenarioML that distinguish it most strongly from this related work are its extensive and consistent use of parameterization, variables for newly created or identified entities, quantification for restricting variable bindings, markup for expressing anaphora unambiguously, co-matching so that one scenario may be used to constrain the context of another, and the emphasis on matching the world as the basis for the meaning of events and scenarios and for relationships between them. The present work focuses on this last item.

A number of XML languages for use cases exist, notably OMG's interchange language for all parts of UML [OMG05]. In contrast, ScenarioML is not designed to match a particular notation but is designed to embody the aspects of scenarios that people use and interpret consistently, with the goal of supporting automation based on this consistency.

Nentwich *et al.*'s xlinkit is a general-purpose framework for expressing and checking consistency constraints in XML documents [NEFE03]. This framework is much more general than ScenarioML and aimed at documents of all sorts, not specifically scenarios. ScenarioML supports the checking of analogous consistency constraints within scenarios, for example checking that the scenario an episode references has in fact been defined.

3 The ScenarioML Language

Scenarios are inherently informal, but nevertheless have certain aspects that are treated consistently by people. ScenarioML provides formal structure and support for a number of such aspects that we have identified from our own experience and in observations and discussions with authors and users of scenarios:

- *Same wording means same thing.* If the same wording is used in two places to mean two different things, this is (in general) considered a mistake. Different wording either means something different, or at least raises the question in a reader's mind of what different meaning might have been intended.
- *Implicit parameterization of scenarios and events*, with actors, agents, systems, etc. being the implicit parameters. A "customer" mentioned in a scenario is very likely to be in essence a parameter that may refer to a different individual in every occurrence of the scenario. Furthermore, within a scenario

the same individual is meant by every reference to that customer, either as repetitions of "the customer" or anaphorically with other phrasing ("he").
- *Implicit time parameterization.* The scenario may take place once at some unspecified time, more than once, or possibly never (e.g. an antiscenario).
- *Division of scenarios into events.* Each event can at least conceptually be identified in the world as a separate occurrence from the events around it. Individual events can occur independently of others in the scenario; it is at least conceivable that one event in a scenario may occur, and the next-listed event may not.
- *Temporal relationships among events.* The relationships are most commonly a simple sequence, but other relationships are encountered ("While the customer is entering his PIN ...") and any consistent relationship is possible.
- *Simple events that are defined by the text that describes them.*
- *An entire scenario as a single event of another scenario*, termed here an episode of that scenario. In use case terms, one "includes" another.
- *Event schemas* that may be equivalent to a repeated event (an iteration), a choice drawn from two or more alternative events (an alternation), or one event possibly superseded by another (an interruption or an exception).
- *An implicit model of the world* (an ontology), containing at a minimum the entities mentioned in the scenario. These entities are often categorized into types ("An ATM card" which refers to an instance but implicitly characterizes a set of similar instances), and may have relationships to entities of the same or other types ("the PIN that corresponds to the customer's ATM card"). Some entities exist outside the time span of the scenarios, and can be identified by name ("the ATM at the main branch on Hillsborough St."); others are created as the result of an event in a scenario, and are identified by a reference that reaches back to that event, directly or indirectly. The world described may be 'the' world, or a hypothetical world containing (for example) planned systems that do not yet exist.

The syntax of ScenarioML supports a scenario author who wishes to make explicit any or all of these aspects that people treat consistently.

Figure 1 shows an excerpt from an industrial scenario for the Mirth medical informatics middleware system, both in XML and in a 'stakeholder-friendly' informal natural language text form derived automatically from the XML. This scenario exhibits several of the features discussed above. The scenario is parameterized explicitly (with parameters *kindOfModule* and *user*). Each parameter is identified with a type (*KindsOfModules* and *Users*, respectively) defined in an ontology not shown in the figure. The scenario is divided into top-level events that in this case occur in a sequence. Event 4 is a schema equivalent to a choice drawn from several alternatives (due to space limitations, only alternative A is visible in the figure). Alternative A is another schema equivalent to a repeated event; the repeated event is given the 'number' ∗because it may occur more than once. The text of simple events and other text components contains words and phrases that have been identified (manually) as references to entities and types defined in the ontology and to the scenario's parameters. This 'stakeholder-friendly' text form is HTML presented by a web browser, and the references are

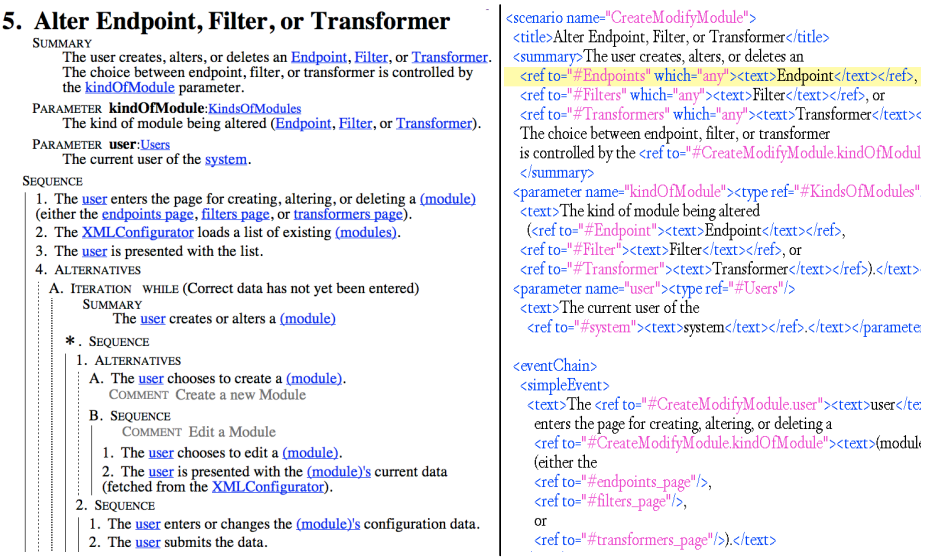

Fig. 1. ScenarioML scenario excerpt, in stakeholder-readable and XML forms

expressed in it as hyperlinks to their referents. In the XML the references are expressed with elements mixed with the text as markup. Each such `ref` element names its referent with a `to=` attribute. Space does not permit discussion of the entire ScenarioML language (presently 49 element types and 20 attribute types); here we will focus on events only.

The author and his research team have become accustomed to writing scenarios directly in ScenarioML using a text editor with XML support, validation against the ScenarioML schema, and occasionally checking the derived informal text form to verify our work; with practice we have become relatively efficient. However, we cannot and do not expect people who write informal text scenarios to choose to write scenarios in ScenarioML directly. We are in the process of developing a ScenarioML scenario editor in the form of an Eclipse plugin, supported by an IBM Eclipse Innovation Award. The plugin provides a visual editor interface analogous to the form of an informal text scenario, with support for structuring scenarios, making textually-expressed parameterization explicit, linking words and phrases in text events to entities in an ontology, and other tasks that are needed in order to create, edit, and work with ScenarioML scenarios. We plan validation studies with the editor to show the extent to which this approach can be made both palatable and effective for the range of kinds of people that work with scenarios.

Figure 2 shows the elements of ScenarioML discussed in the remainder of this paper and their relationships. There are four classes of events: simple events defined by a text description, episodes that use another scenario as an event, compound events composed of other events in a temporal pattern, and event

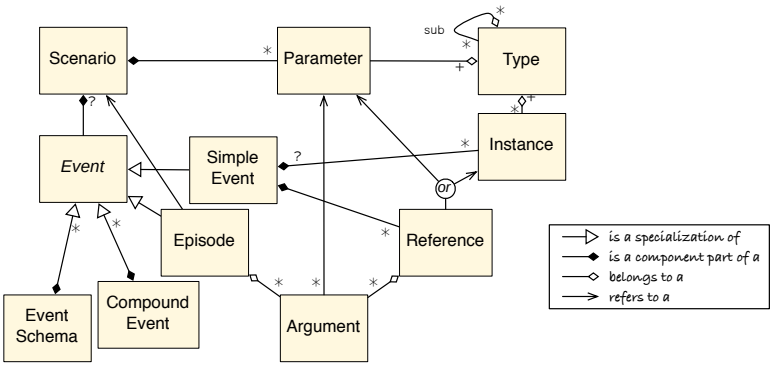

Fig. 2. Metamodel of events and related constructs in ScenarioML

schemas. Compound events and event schemas are themselves events, but also have other events as their component parts.

3.1 Simple events

A simple event consists of text describing something happening in the world. The text is fundamentally something to be understood by a human reader; we do not attempt to interpret it by natural language processing, and assume only that two simple events whose text is the same are equivalent, and that two simple events whose text is different (once references are resolved) are not.

3.2 Episodes

An episode reuses one scenario as an event of another. It is equivalent to and can be replaced by the events of the scenario it references, with the scenario's parameters text-replaced by the corresponding arguments of the episode.

Scenarios reused in this way often appear as episodes in more than one scenario. The text of the reused scenario is typically not equally appropriate for all its uses as an episode, leaving the reader to guess precisely how to interpret it in a particular context. We address this issue through explicit parameterization of scenarios and arguments for episodes. This is not a perfect solution (text replacement of parameters by arguments can result in capitalization, plurality, and other oddities) but it has worked well in a large majority of cases and more clearly expresses the scenario author's intent.

3.3 Compound events

A compound event comprises two or more component events with temporal constraints between them. The compound event begins when its earliest component event begins, and ends when its latest component event ends. We use Allen's

interval algebra to specify qualitative temporal constraints between the events [All83]. Allen identified the thirteen relations that can hold between two concrete time intervals, illustrated in Figure 3. A concrete interval is one bound to a

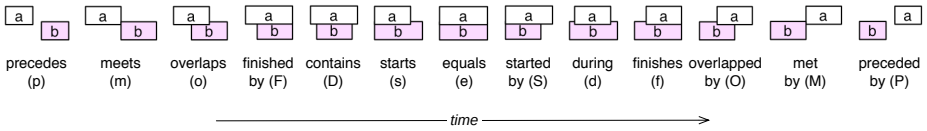

Fig. 3. Allen's interval relations

particular interval on the line of time, such as "16:02:14 through 16:02:18 GMT" or "the first time ATM card 283749 was in ATM 182" The thirteen relations are distinct in that only one holds between any two intervals, exhaustive in that any two concrete intervals are related by one of the thirteen, and qualitative in that they do not specify any numerical lengths of time, but only relative position in time. For example, June 14 *precedes* June 16 (p); on any one day, morning *meets* afternoon (m); and summer *contains* July (D). For abstract time intervals such as those corresponding to a scenario event whose concrete time is not known, the possible relations are given as a set of individual relations. For example, if event a must begin before event b, but may end either before b begins, when b begins, or after b begins but before it ends, then the relation would be (pmo). For groups of three or more intervals, it is possible to specify relations for some of the pairs of intervals and infer the remaining relations using an algorithm (although in the general case the algorithm is NP-complete). More detail is given in our technical report [Als05].

We divide compound events into three groups, based on the complexity of their temporal interrelationship and (correspondingly) the complexity of temporal inference on them. Simplest are *chains* of events related by combinations of the five positive Allen relations (listed in Figure 4). Using positive relations

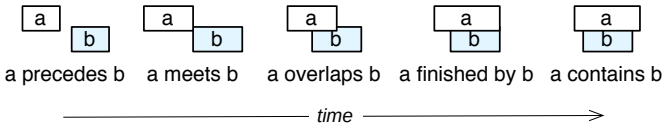

Fig. 4. Positive Allen relations

constrains the chains to be listed with events in order by beginning, so that each chain can only be written in one canonical way and there are no cycles. In practice, almost all scenarios are written as chains. Next are *partial orders* (dags) of events related by positive Allen relations. Again, using positive relations constrains the dags so that there are no cycles, and in addition allows each dag to

have a canonical form (if a sort order for the children of each node is specified). Figure 5 shows a compound event that is a dag. Inference for both chains and dags is tractable, and we believe that they suffice to write virtually all scenarios that are of practical interest. Last and most complex are the unrestricted *graphs* of events, which allow arbitrary relations among events and are not only computationally intractable in the general case but are also very difficult for a person to write or think about, and so are unlikely to occur in actual use.

3.4 Event schemas

The event schemas we will consider here are *alternation* among several alternative events and *iteration* of an event some number of times, either in a numeric range of repetitions, once for each of a list of entities, or while a given condition in the world is true. We term these 'schemas' as distinct from compound events because each schema in effect causes its scenario to be equivalent to a set of scenarios containing no schema:

- an alternation makes its scenario equivalent to a set of scenarios each containing a different one of the alternatives;
- an iteration makes its scenario equivalent to a set of scenarios each containing a a different possible unrolling of the iteration as a chain.

ScenarioML also has schemas for interruption of one event by another at some point, and for exceptions, where one event may preempt the remainder of another at some point, but these will not be considered in this paper.

3.5 Matching events to the world

As mentioned in the Introduction, we say that the meaning of a scenario (or an event) is the set of occurrences it describes in the world. In this subsection we discuss how this can be implemented, by matching specific events to occurrences in the world.

A simple event is a text description of a single occurrence. The simple event matches the occurrences it describes, and none other. Since we leave the interpretation of text to human readers, matching of an event to an occurrence must be done either by a person or by a software oracle written for that event.

We can generalize for simple events containing references. If the simple event contains markup for references, the occurrence must involve the appropriate entities referred to.

- If the reference is to a specific named entity, the occurrence must involve it.
- If the reference is an anaphoric one to the entity involved in the occurrence matched by another event, then this event's occurrence must correspond.
- If the reference is to a new entity, the entity must be one previously unknown.

A compound event comprises a group of component events with specific temporal relations. Therefore, a compound event can match an occurrence that comprises a group of component occurrences if the component events can be mapped onto the occurrences so that

– each component event matches its corresponding occurrence, and
– the relation between each pair of occurrences is contained in the relation between the corresponding events.

Figure 5 illustrates the matching of a compound event's subevents to occurrences in the world. Each subevent is mapped to a unique occurrence that it matches. For example, occurrence a is matched by event A. In this figure, we

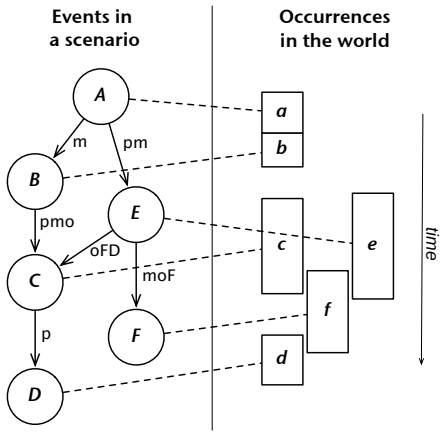

Fig. 5. Mapping events to occurrences

don't know whether A matches a because it is a simple event that describes a, or an episode, compound event, or event schema that recursively matches the component suboccurrences (not shown) of a.

The compound event then matches if occurrence relations are contained in corresponding event relations. Figure 6 illustrates this. Event A is mapped to occurrence a and E to e, so in order for the compound event to match, the relation between a and e, which is (p), must be contained in the relation between A and E (and similarly for the other corresponding relationships). The relation between A and e is (pm) which does in fact contain (p). Examination of the figure shows that every occurrence relation is contained in the corresponding event relation, so this compound event matches.

Because the compound event is a dag, with only positive relationships in the partial order, it is not necessary either to infer the remaining relationships among the subevents, or determine the remaining concrete relationships between the suboccurrences; once we have established that the partial order relationships correspond, we know the others will correspond as well. This is an example of the computational simplification that separating chains and dags out of the general case affords us.

Event schemas are matched against occurrences in the expected ways. An alternation schema matches an occurrence if any of its subevents matches the

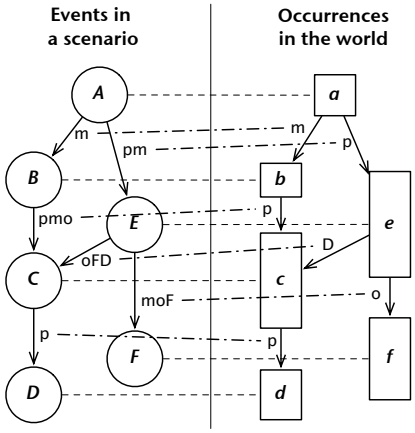

Fig. 6. Mapping corresponding relations

occurrence. An iteration schema matches a (compound) occurrence if the occurrence is a chain matched by any of the iteration's unrollings.

Finally, an episode matches an occurrence if the scenario the episode refers to, with arguments substituted for parameters, matches the occurrence.

4 Comparing Events

We have shown how each kind of event can be matched recursively against occurrences in the world. In this section, we will explore how this can be leveraged to compare two events. Since the meaning of an event is the set of occurrences it describes in the world, we shall compare two events by comparing their sets of occurrences. Three of the relations that result are illustrated in Figure 7.

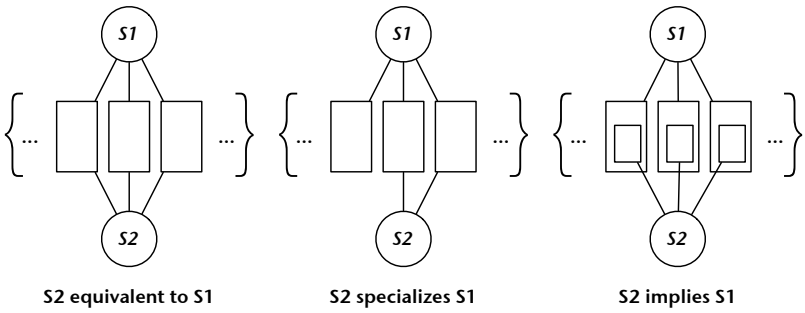

Fig. 7. Three relations between events

- Two events are *equivalent*, if their sets are equal.
- One event is a *specialization* of another, if its set is a subset of the other's.
- One event *implies* another if every occurrence of the other is contained in an occurrence of the first (this can only occur with compound events).

Specialization and implication produce partial orders of events, with every implication partial order a subset of some specialization partial order.

4.1 Simple events

Two simple events without references are equivalent if they are textually equal.

Two simple events with references are equivalent if their texts are equal except for references in corresponding locations in their text, and if corresponding references are equivalent; two references are equivalent it they refer to the same instance or range over the same sets of instances.

The relation of specialization between two simple events containing references is complex and not discussed here due to limitations of space.

4.2 Compound events

Two compound events are equivalent if (i) there is a bijection between their subevents such that corresponding subevents are equivalent, and (ii) corresponding pairs of subevents have the same temporal relation in each of the compound events. But they may also be equivalent if one or both contain compound subevents, in which case the two compound events may need to be reduced to compound events with only simple events as subevents in order to determine equivalence. Here the classification into event chains, dags, and graphs is useful: chains of chains reduce to chains, and chains of dags or dags of chains or dags reduce to dags, while compound events involving graphs reduce to graphs. Because chains and dags have canonical forms, their comparisons are computationally straightforward. Comparison of graphs is computationally intractable; fortunately, event graphs appear to be extremely uncommon in actual scenarios.

Specialization of two compound events is analogous, with an injection between subevents whose temporal relations in the first are subsets of those in the second. The computational approach and issues are also analogous.

A compound event A *implies* another compound event B if there is a bijection between their subevents such that

- each subevent of A specializes the corresponding subevent of B, and
- the relation between every pair of B's subevents is contained in the relation between the corresponding pair of A's subevents.

The computational approach and issues are analogous.

The only compound events that are comparable to simple events are the degenerate ones that contain a single simple event; in that case, the compound and simple event are compared in terms of the singleton simple subevent.

4.3 Schemas

Alternation schemas are comparable if one's alternative subevents map to the the other's: equivalent if corresponding subevents are equivalent, and a specialization (implication) if corresponding subevents are specialized (implied) in the same direction.

Iteration schemas have more complex possibilities for comparison. In the simplest cases, two iteration schemas are comparable if their iterated events are comparable. They are equivalent if their subevents are equivalent, the schemas' temporal relations are equal, and the sets of iterations are the same; one specializes (implies) the other if its subevent specializes (implies) the other's, its temporal relation is a subset (superset) of the other's, and its set of iterations is a subset (superset) of the other's. However, there are a range of additional possibilities based on relations between the unrollings, as for example when one's subevent is a chain of repetitions of the other's, which this paper will not address.

An alternation and an iteration can be related only if the alternation's alternatives are related to the iteration's unrollings

An alternation is specialized by each event that is one of its alternatives, and is implied by the same events. Only a degenerate alternation of one alternative can be equivalent to a compound or simple event. Similarly, an iteration that may match a single repetition is specialized by an event that is a specialization of its subevent, and is implied by the same event. Only a degenerate iteration of exactly one repetition can be equivalent to a compound or simple event.

4.4 Episodes

Finally, two episodes are compared by comparing the events of their respective scenarios, with parameters replaced by arguments.

5 Implementation, Validation, and Application

The syntactic form of ScenarioML is specified by a grammar and (somewhat less strictly) an XML schema. Its semantics are implemented in a Java package most of which is generated by a Perl script from the ScenarioML grammar. The package provides methods for examining, comparing, and operating on the scenarios embodied by the objects, and for constructing objects from ScenarioML input and marshalling the resulting objects back into ScenarioML or into other output forms. HTML output is implemented (Figure 1), and LaTeX output has been prototyped for feasibility. The package implements a conservative comparison of events for equivalence, specialization, and implication, in that if it identifies two events as related, a person would judge the events have that relation, but the converse is not always true. We are continuing to explore the extent to which automated comparison may be extended.

ScenarioML has been validated by using it to write scenarios for a variety of systems and contexts, including the Mirth medical informatics middleware system [ASW+06], a Traffic Information System (TIS) in a private plane [ATB06],

banking business rules, Web site personalization, and others. The Mirth scenarios were the basis for an empirical validation that showed a range of industrial stakeholders preferred to read and work from ScenarioML scenarios over use cases, and that two comparable participants re-expressing use cases in ScenarioML and in sequence diagrams, respectively, uncovered more requirements problems by using ScenarioML [ASW+06]. Interestingly, it took comparable amounts of time to write the scenarios in ScenarioML with an XML editor and to write comparable sequence diagrams with a specialized sequence diagram tool.

The concepts of ScenarioML have been used as the foundation of an approach for specification-based testing using plans and goals [ARSZ05,WAZR06]. This approach annotates a system's source code with system goals, then precompiles them into goal and event emitters. The plans, which are essentially scenarios whose events are goal satisfactions, are automatically translated into rules for an expert system that recognizes when each plan has been followed successfully. Manually produced oracles are used to verify goal satisfaction. The result is that the running program is tested for conformance to plans and achievement of expected goals.

Research has shown that scenarios augmented with rich media are more effective for walkthroughs, analysis, and elicitation, but the cost of videographing users in the context of using a system can be prohibitive. We have used ScenarioML to automatically produce an animated social visualization of scenarios. In this project, actors and entities in scenarios are annotated with a small increment of extra information, from which social worlds animation software produces an animation of the scenario's events on the fly. An initial pilot study showed promise that it helped users understand scenarios and find problems in them [ATB06].

Baumer *et al.* use ScenarioML scenarios to drive computed social worlds of autonomous animated agents [BTYA06]. The social norms of the computed society are expressed as scenarios; individual agents 'watch' for occurrences matched by the first events of scenarios, and contribute to the interaction by speaking later events. User-spoken inputs are incorporated into their social norms, replaying sound files automatically recorded from the input as events in an interaction.

6 Future Work

The implementation of support for operations and analysis on scenarios and the definition of the relationships between scenarios opens possibilities for using scenarios in combination in new ways. We envision using a scenario to specify the expected context of other scenarios that it calls as episodes, and the use of one scenario as a constraint on another. These uses of scenarios open new ways of modularizing and combining scenarios, with a number of possible uses.

One such use is the integration of scenarios more fully into software architecture specifications. We believe that software architecture needs to integrate behavioral specifications into architectural component descriptions, and use architectural connectors as operators that combine these specifications recursively.

Some work has been done with (for example) Statecharts and other design-level notations. However, we feel that requirements-level scenarios offer possibilities of stronger connection to requirements and better interaction with stakeholders. This area holds promise and we plan to pursue it using ScenarioML.

Scenarios are often used in design processes, but usually as a peripheral element that supports other design artifacts. We believe that one reason for this is the difficulty of working with scenarios without effective tool support. A goal of our scenario tool support is to produce tools with which design can be done with scenarios, rather than merely using scenarios.

7 Conclusions

We have described an approach for adding structure to scenarios in a way that reflects the assumptions people ordinarily make about the meaning of text scenarios. This approach has been embodied in an XML language, ScenarioML, and a Java package implementing it. In the process of working through the approach and implementing its features, some interesting relations between scenarios have been given clear definitions. The approach offers a foundation for automated support of editing, analyzing, and managing scenarios. It has been used successfully in several projects and further use is planned and appears promising.

Acknowledgments

The author gratefully acknowledges the comments and suggestions made by the anonymous reviewers of an earlier version of this paper.

References

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[Als05] Thomas A. Alspaugh. Software support for calculations in Allen's Interval Algebra. ISR Technical Report UCI-ISR-05-02, Institute for Software Research, University of California, Irvine, February 2005.

[AM04] Ian F. Alexander and Neil Maiden, editors. *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*. John Wiley & Sons, Ltd., 2004.

[ARSZ05] Thomas A. Alspaugh, Debra J. Richardson, Thomas A. Standish, and Hadar Ziv. Scenario-driven specification-based testing against goals and requirements. In *11th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'05)*, pages 201–216, June 2005.

[ASW+06] Thomas A. Alspaugh, Susan Elliott Sim, Kristina Winbladh, Mamadou Diallo, Hadar Ziv, and Debra J. Richardson. The importance of clarity in usable requirements specification formats. Technical Report UCI-ISR-06-14, Institute for Software Research, University of California, Irvine, September 2006.

[ATB06] Thomas A. Alspaugh, Bill Tomlinson, and Eric Baumer. Using social agents to visualize software scenarios. In *ACM Symposium on Software Visualization (SOFTVIS'06)*, pages 87–94, September 2006.

[BBL+04] F. Basanieri, A. Bertolino, G. Lombardi, G. Nucera, E. Marchetti, and A. Ribolini. Cow_Suite: A UML-based tool for test-suite planning and derivation. *ERCIM News*, 58:30–32, July 2004.

[BFJZ92] K. Benner, M. S. Feather, W. L. Johnson, and L. Zorman. Utilizing scenarios in the software development process. In *IFIP Working Group 8.1 Working Conference on Information Systems Development Processes*, December 1992.

[BLB05] Karin Koogan Breitman, Julio Cesar Sampaio do Prado Leite, and Daniel M. Berry. Supporting scenario evolution. *Requirements Engineering Journal*, 10(2), May 2005.

[BTYA06] Eric Baumer, Bill Tomlinson, Man Lok Yau, and Thomas A. Alspaugh. Normative echoes: use and manipulation of player generated content by communities of NPCs. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE-06)*, June 2006.

[Coc00] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman, 2000.

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

[Hau04] Peter Haumer. Use case-based software development. In Ian F. Alexander and Neil Maiden, editors, *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*, pages 237–264. John Wiley & Sons, Ltd., 2004.

[ITU99] Message Sequence Chart (MSC). ITU-T Recommendation Z.120, International Telecommunications Union, November 1999.

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

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

[Mai98] N. A. M. Maiden. CREWS-SAVRE: Scenarios for acquiring and validating requirements. *Automated Software Engineering*, 5(4):419–446, October 1998.

[MMMR98] N. A. M. Maiden, S. Minocha, K. Manning, and M. Ryan. CREWS-SAVRE: Systematic scenario generation and use. In *Third International Conference on Requirements Engineering (ICRE'98)*, pages 148–155, 1998.

[NEFE03] Christian Nentwich, Wolfgang Emmerich, Anthony Finkelstein, and Ernst Ellmer. Flexible consistency checking. *ACM Trans. Softw. Eng. Methodol.*, 12(1):28–63, 2003.

[OMG03] OMG Unified Modeling Language specification (version 1.5). Document formal/03-03-01, Object Management Group, Framingham, MA, March 2003.

[OMG05] MOF 2.0/XMI mapping specification, v2.1. Document formal/05-09-01, Object Management Group, Framingham, MA, September 2005.

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

[Rob04] Suzanne Robertson. Scenarios in requirements discovery. In Ian F. Alexander and Neil Maiden, editors, *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*, pages 39–59. John Wiley & Sons, Ltd., 2004.

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

[Sut03] Alistair Sutcliffe. Scenario-based requirements engineering. In *11th IEEE Joint International Conference on Requirements Engineering (RE'03)*, pages 320–329, September 2003.

[UCKM04] Sebastian Uchitel, Robert Chatley, Jeff Kramer, and Jeff Magee. System architecture: the context for scenario-based model synthesis. In *SIGSOFT-2004/FSE-12: ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 33–42, 2004.

[WAZR06] Kristina Winbladh, Thomas A. Alspaugh, Hadar Ziv, and Debra J. Richardson. An automated approach for goal-driven, specification-based testing. In *21st International Conference on Automated Software Engineering (ASE 2006)*, pages 289–292, September 2006.