ISR Institute for Software Research
University of California, Irvine

Scenarios, Business Rules, and Matching

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

April 2006

ISR Technical Report # UCI-ISR-06-5

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

Scenarios, Business Rules, and Matching

Thomas A. Alspaugh

Institute for Software Research
University of California, Irvine
Department of Informatics
Donald Bren School of Information and Computer Sciences
Irvine, CA 92697-3425 USA
alspaugh@ics.uci.edu

ISR Technical Report #UCI-ISR-06-5
April 2006

Abstract

Scenarios (or use cases) and business rules each have strengths and offer mutually reinforcing views of a software system. Scenarios describe uses of a system in terms of situations, interactions, and events unfolding over time. Business rules describe fundamental constraints on a system's transactions. However, analysis of scenarios and business rules together can be challenging. We present an approach for evaluating these two distinct views in combination. We operationalize business rules as scenarios that must match the world in every appropriate context, or as negative scenarios that must not match. Scenarios are patterns that can match occurrences in the world; a system whose behavior meets its requirements results in occurrences that its scenarios match successfully. We mediate the interaction of business rules and scenarios through *co-matching events* appearing in two or more scenarios. A co-matched event is not judged to have successfully matched unless all the events that co-match it do. A case study of business rules for automated teller machines illustrates our approach.

Scenarios, Business Rules, and Matching

Thomas A. Alspaugh

Institute for Software Research
University of California, Irvine
Department of Informatics
Donald Bren School of Information and Computer Sciences
Irvine, CA 92697-3425 USA
alspaugh@ics.uci.edu

ISR Technical Report #UCI-ISR-06-5
April 2006

Abstract

Scenarios (or use cases) and business rules each have strengths and offer mutually reinforcing views of a software system. Scenarios describe uses of a system in terms of situations, interactions, and events unfolding over time. Business rules describe fundamental constraints on a system's transactions. However, analysis of scenarios and business rules together can be challenging. We present an approach for evaluating these two distinct views in combination. We operationalize business rules as scenarios that must match the world in every appropriate context, or as negative scenarios that must not match. Scenarios are patterns that can match occurrences in the world; a system whose behavior meets its requirements results in occurrences that its scenarios match successfully. We mediate the interaction of business rules and scenarios through *co-matching events* appearing in two or more scenarios. A co-matched event is not judged to have successfully matched unless all the events that co-match it do. A case study of business rules for automated teller machines illustrates our approach.

1. Introduction

Both scenarios and business rules are widely used to describe the required behavior of systems. It can be challenging to determine whether a group of scenarios follows a set of business rules, however. Identifying the rules that apply to specific scenarios, and analyzing each scenario for compliance to them, are tasks that typically require human expertise and insight. For systems described by commercial-sized collections of scenarios and business rules, this can be a daunting challenge. Our approach uses a combination of scenario expressiveness and analysis techniques to address this challenge, with the potential for automated support.

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 [2]. 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 [6, 14]. They are used to simulate and explore a system's use [6] or a design's utility [22].

Business rules describe fundamental constraints and policies for the transactions conducted by an organization and its systems. They provide a global perspective on the operations of the organization, in contrast to the local, partial descriptions provided by scenarios. Because they are global, they can be used to check that scenarios are consistent with the principles and procedures of the organization as a whole. The primary audience for business rules is the business-oriented members of the organization or similar organizations, rather than technically-oriented developers. Business rules are typically written informally in natural language, or alternatively as structured natural language corresponding to a more formal underlying structure in a logic. They are not system requirements themselves, but are at a higher level and give rise to system requirements [11, 18].

Scenarios and business rules provide complementary and mutually reinforcing kinds of knowledge. Both scenarios and business rules are expressed in prose forms; both stakeholders and developers can understand and work with them directly. Scenarios are specific behaviors desired for the system, illustrating particular uses in specified contexts. A scenario shows how the system will act in achieving a desired result while satisfying business rules. Business rules are more general policies or procedures that apply in many or all contexts. The business rules express system properties that are often distributed across many different scenarios. A rule expresses in one place a policy that every relevant scenario must adhere to. Business rules can provide an understanding of how a particular scenario fits into the overall system behavior. Scenarios provide specific examples or counterexamples for a business rule, helping communicate the meaning of the rule and aiding stakeholders and developers in coming to a common understanding. In addition, the use of scenarios to explicate business rules brings those rules into the same domain as the scenarios that must satisfy them. Separating these two types of concerns simplifies the expression of each one, but finding a common ground enables evaluation and validation of both.

Because business rules express things in a markedly different way than scenarios, it can be challenging to identify conflicts between the business rules and the scenarios that are to adhere to them. Indeed, it is not straightforward to locate the scenarios that might satisfy or contradict a particular business rule. The current trend in commercial products is toward the generation of code from business rules, either as a means of integrating services provided by components produced separately, or for inclusion in a system to monitor and verify that business rules are followed. However, stakeholders and developers still need to analyze whether a proposed system may be expected to follow business rules, and this analysis is needed before resources have been expended to implement the system.

We describe an approach for integrating and evaluating the combination of business rules and scenarios. We operationalize business rules with scenarios describing specific ways in which each business rule may succeed or fail. They are either scenarios that must match the world in every appropriate context (else the system has failed), or negative scenarios that must not match. These scenarios may describe behaviors that are in some sense test cases for success or failure of a business rule, or (depending on the rule) may arguably apply to every possible success (or failure). Scenarios are patterns that can match occurrences in the world, and a system whose behavior meets its requirements results in occurrences that its scenarios match successfully. We use our language ScenarioML for expressing scenarios in a form close to natural language but offering significant additional expressive power; its novel facilities for parameterization of events, variables for newly identified entities, anaphora, and co-matching events offer a significant foundation for evaluating scenarios in terms of the occurrences they match in the world. We mediate the interaction of business rules and scenarios through *co-matching*, in which one scenario gives additional constraining context to an event in another. By expressing scenarios in a language with semantics defined in terms of matching, we can model interactions between scenarios and business rules by examining how their semantics compare. This allows us to begin to evaluate and draw inferences about them, for example by identifying scenarios that do not follow a specific business rule, and business rules that do not constrain any scenario and thus have no effect, all based on comparison of what each scenario can match. We believe our approach can offer significant benefits for early-lifecycle evaluation and analysis.

The remainder of this paper is organized as follows. Section 2 defines scenario terminology used in the paper. Section 3 discusses business rules and gives some examples. Section 4 explores how scenarios match against the world. Section 5 presents our scenario language ScenarioML which we use here in exploring scenario matching and the interaction of business rules and scenarios. Section 6 illustrates our approach using examples from a case study of ATM scenarios and business rules. Section 7 presents related work, and Section 8 summarizes the lessons learned and future work.

2. Scenario terminology

A *scenario* is a sequence of events, and associated information such as the necessary context.

An *event* describes something that can happen over an interval of time: an action performed by some agent, a communication from one entity to another, a change in the world's state, an interval during which a condition holds, or in general anything of interest that may happen.

An *occurrence* is something that actually happens in the world (or could), and that may be described by an event in a scenario.

A *simple event* is one not subdivided into briefer component events. A *compound event* is composed of subevents and specific temporal relations between them. A compound event begins when its earliest subevent begins, and ends when its latest subevent ends.

Corresponding to a compound event, a *constellation of occurrences* is a set of occurrences and their relations.

Scenarios and events *match the world* when they describe occurrences that have taken place. A simple event can match a single occurrence. A compound event can match a constellation of occurrences, if each subevent matches a distinct occurrence and the relation between every two matched occurrences is consistent with the relation between the events matching them.

A *negative scenario* is a scenario for which the sense of matching success is negated: a negative scenario successfully matches the world whenever its events fail to match.

Two scenarios are *equivalent* if they match the same constellation. A scenario is a *generalization* of another if it matches every constellation the other does, plus some more.

3. Business rules

Business rules represent policies, procedures, and constraints that describe an enterprise's way of doing business. A business rule may address a process for achieving an objective, the way the business views the world and structures its records, or an obligation imposed by outside regulation or custom. Conceptually, business rules lie between high-level objectives and system requirements; they are seen as arising from the objectives of the enterprise and leading to system requirements [11, 18]. Below are some example business rules.

- "A bank requires a supervisor's signature before cashing a check over $5000." [18]
- "A meeting must have a meeting initiator." [11]
- "Whenever the stock is below the reorder point, only good customers will have their order immediately processed." [26]
- "Good customers of a product are defined as those who have bought at least twice the average sales per customer over the last 12 months." [26]
- "Put the orders of bad payers on a waiting list, until they pay the amounts due." [26]

A business rule can be operationalized positively by a scenario that describes how the rule is followed, or negatively by one describing how it fails to be followed. An informal scenario positively operationalizing the first example business rule is

(1) A customer presents a check for more than $5000 to a teller to be cashed;
(2) The teller obtains a supervisor's signature for the check;
(3) The teller cashes the check and gives the amount to the customer.

The business rule is integrated with other scenarios for the bank's operations by having (1) and (3) in the business rule scenario co-match with any appearance of events (1) and (3), respectively, in another scenario. Thus this scenario becomes a necessary context for any transaction involving cashing a check for more than $5000. If there is any situation in which this necessary context prevents another scenario from matching, then that is a situation in which the business rule fails to be followed. We use potential co-matched events to identify other scenarios or combination of scenarios for which the business rule should apply, then analyze them to find any contexts in which the business rule fails, as will be described in Section 6.

An informal scenario negatively operationalizing the second example business rule is

Precondition: Meeting M has no meeting initiator.
(1) The meeting scheduler system schedules M.

This scenario uses a precondition to restrict the contexts in which the scenario can match: only for a meeting M that has no meeting initiator. Because it negatively operationalizes the business rule, any context in which it matches is one in which the business rule fails to be followed. As with the positive business rule scenario, we look for potential co-matched events to identify other scenarios or combinations of scenarios for which the business rule should apply, then analyze them further to search for ways it could fail.

4. Scenarios and matching

We believe that the meaning of a scenario is the set of all possible occurrences in the world that the scenario could match. We cannot deal with this set directly (it is generally infinite), but we can use it to understand scenarios better and make the most effective use of them. A simple event matches individual occurrences in the world; two simple events whose description is the same match the same occurrences, and are equivalent. A compound event matches constellations of occurrences; a compound event consisting of a sequence of events matches constellations of the occurrences matched by the events in the sequence, where the sequence of the occurrences in each constellation is the same as the sequence of the corresponding events matching them. Two event sequences containing equivalent events in the same sequence match the same constellations, and are equivalent also. A scenario consisting of an event sequence matches the constellations that the sequence matches, constrained by the scenario's precondition if present. An episode (or inclusion of a use case), an event that stands for another scenario, matches if the referred-to scenario matches. This description covers the vast majority of scenarios (and use cases).

However, a closer examination shows that more is going on. Most scenarios and events are implicitly parameterized: "A customer inserts an ATM card into the ATM" is implicitly parameterized by at least the choice of customer, ATM card, ATM, and time. Some of these may be parameters of the scenario, which were bound to entities in the world before the scenario began. Others, probably all four in this case, are variables that were newly bound to entities when the event was matched to an occurrence. The concrete scenario

> Alan Windham inserts his wife's ATM card into the FCU (Friendly Credit Union) ATM at the corner of Fourth and Main two seconds after a backhoe cut the main communication line to the central server

while it may or may not match the world, is clearly related to the original.

Similarly, most scenarios and many events are implicitly quantified on one or more implicit variables and/or parameters: "An individual who can read either English or Spanish inserts any one of the thousands of FCU ATM cards into any one of the ATMs running our software" is one reasonable interpretation of the original event. The quantification ranges are necessarily conjectural, as it is rare for a scenario to say anything about them directly. Later events in the same scenario use anaphora to refer back to implicit variables and parameters in earlier events, as in "The ATM returns the ATM card and displays a welcome screen." In this case it is certain that the same ATM and card mentioned earlier are meant. However in a scenario involving two of the same thing (such as a transfer from one account to another) the situation is more complex.

Closer consideration also shows that a strict sequence of events is not always intended. Adjacently-listed events may be intended each to begin when the preceding one ends (as in "The ATM beeps while the card is in the ejected position" followed by "The customer takes the card"); with some time in between, as is often the case; overlapping (as in some ATMs that print a receipt while asking if another transaction is desired); or in other relationships.

Event schemas such as alternation and iteration are also seen in scenarios and related notations, and can be defined in terms of matching. We distinguish schemas from other events because

- each schema represents a set of events and matches the union of what those events match,
- an alternation represents the set of its alternative events, and
- an iteration represents a set of sequences of the iterated event.

Whereas a scenario containing no schema is still essentially the classic script or linear sequence of events that is its most common definition, a scenario containing a schema (or a *schematic scenario*) is equivalent to a set of scripts. We find event schemas are essential in expressing behavior with a manageably small number of scenarios. Schemas also support reasoning reuse because each schema's pattern provides its own kind of consistent modularity and relations for its subevents.

Taking account of parameterization, quantification, anaphora, relations between events, and schematic events enables us to compare and reuse events more effectively. With this knowledge we have a better grasp on what occurrences or constellations an event can match, and how this may vary when the same event appears in different contexts.

We can make further use of this knowledge in using *co-matching*. Any event is constrained to match in the context that its scenario provides. The event "The teller obtains a supervisor's signature for the check" does not occur in isolation. It happens after a customer presents a check for more than $5000, and before the teller pays out the check. If an event is part of several scenarios, then it can occur in any of the contexts those scenarios provide for it, and can match in all the constellations any of them match. Co-matching is a way not of adding a new possible context, but of constraining contexts previously available. If event e in scenario S becomes co-matched by event c in scenario C, e's context is constrained. Before it could match in

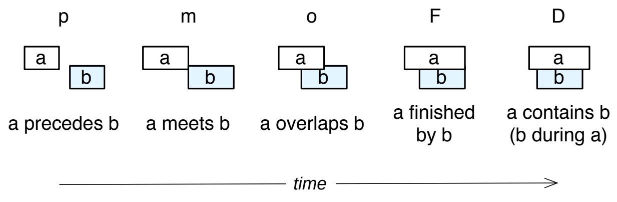

Figure 1. The basic positive Allen relations

any constellation S matched. Now it can match only in a constellation S matches that shares at least one occurrence with a constellation C matches. The shared occurrence is the one matched both by e in S and by c in C. By adding a scenario describing how a supervisor verifies a check (signature, endorsement, sufficient balance, etc.) and ending with "The teller obtains a supervisor's signature for the check" co-matched with the same event in the original scenario, we constrain the contexts in which large checks can be cashed. We accomplish this without complicating the original scenario, and have a new scenario that is more likely to be reusable.

5. ScenarioML

ScenarioML is an XML-based language for expressing scenarios in natural language with additional supporting information. Its goal is to support ordinary scenario use while allowing enhanced expressiveness and automated operations. The enhanced expressiveness includes the parameterization, quantification, anaphora, relations between events, and co-matching discussed in the preceding section. Examples of ScenarioML scenarios appear below in Section 6. During the two-year evolution of the language to its present form, automation has been prototyped for comparison of events for equivalence and for generalization, matching scenarios against occurrence streams, transformations between equivalent forms or to specializations, and translation to presentation forms such as HTML and $\LaTeX$ [4].

We use ScenarioML rather than plain text because it supports automated analysis and manipulation that plain text does not. We work with scenarios rather than Message Sequence Charts (MSCs) and similar formalisms because we believe that not all events should be modeled as messages from one agent to another, especially in pre-design phases, and that the substantial benefits of scenarios in this context are worth pursuing.

Simple events in ScenarioML are text, with optional markup to identify parameterization, quantification, and anaphora. There is support in ScenarioML for defining entity types and subtypes for parameters, variables, and quantification ranges. Compound events are constructed using Allen's interval algebra relations between the subevents. These relations are the thirteen possible disjoint qualitative relationships between two concrete time intervals, and all combinations thereof; they support a variety of operations including temporal inference [3]. We focus on the *positive* relations, the 2^5 combinations of the five basic relations for which the first-listed interval begins before the second one. The five positive basic relations are illustrated in Figure 1. ScenarioML classifies compound events into three cases, both to simplify the expression of the most common and simplest compound events, and to make use of much more efficient algorithms wherever possible.

- event chains (ordinary sequences), in which successive events are related by positive Allen relations;
- partial orders of events whose covering relations are positive Allen relations; and
- general event graphs with unrestricted Allen relations.

Event chains are simplest and by far the most commonly encountered compound events, and event partial orders are more complex but rare. Comparison and inference with these is quite manageable. The general satisfaction problem for Allen's interval algebra is NP-complete; but we have yet to encounter a scenario that requires a general event graph, much less one of substantial size, so the intractability of the general case is not of practical concern.

ScenarioML provides event schemas for alternation and iteration.

Any scenario in ScenarioML may be used as an episode, of another scenario. If the scenario has parameters then it may be given arguments in its appearance as an episode. To better support reuse of individual events, ScenarioML allows definition of parameterized event types that take an event (simple, compound, or schema) as a template. Instances of the event type then appear as individual events, with arguments if the type was parameterized.

The base task for matching ScenarioML events and scenarios against the world is to classify occurrences by the simple events that match them. In the most straightforward case the occurrences are themselves strings, and a simple event matches an occurrence if their strings match. This is the case in our use of ScenarioML in computed social worlds [5]. In general, the implementation of occurrence classification must be done on at least a domain-by-domain basis. Once simple events are matched, compound events and event schemas are matched recursively as discussed above in Section 4.

Comparison of scenarios and events follows a similar recursive pattern. Except in a small number of special cases, an event of each kind can only be equivalent to another event of the same kind: two simple events, two event chains, two episodes, etc. Exceptions occur for chains of one event, partial orders of one event, alternations of one event, etc. (each is equivalent to that event); linear partial orders, equivalent to the corresponding event chain; alternations of all unrollings of a finite iteration, equivalent to the iteration; and so forth. The situation is more complex but analogous when comparing events for generalization-specialization; due to space limitations here, we will simply note that parameterization becomes a significant issue.

6. Case study: ATM business rules

We validated our approach on a collection of business rules and scenarios for ATMs, from which we selected the examples presented in this section. The business rules were obtained from the author's credit union and from a contact at a bank; the scenarios were recorded and abstracted from notes of interactions with a variety of ATMs in the United States and Europe, focusing on ATMs at two credit unions in the U.S. and using the remainder as comparisons and contrasts for insight. Each business rule was used to derive one or more scenarios, either positive scenarios that matched ways the business rule could be satisfied, or (more commonly) negative scenarios that matched ways the rule could fail to be satisfied. Where possible, we produced scenarios for which we could argue that the scenarios matched all ways the rule could be satisfied (or fail to be). Where this did not appear possible with ScenarioML as it stands now, we noted the rules for further research. We found no business rules in this study for which operationalization seemed impossible in principle. We analyzed the original scenarios plus the business rule scenarios to identify equivalence classes of events, and reworded equivalent events as necessary so events in each class were textually identical. We made use of event parameterization wherever possible to efficiently produce larger sets of equivalent events that differed only by which account was involved, for example. Finally, we derived an event type definition for each equivalence class, and replaced each equivalent event by a reference to that type. Note that for clarity, most events shown in the figures of this paper have not had this final replacement of identical events done. Because this case study was an initial study investigating the practicality of our approach, resources were not spent to produce automated support beforehand. Instead, transformations and comparisons were done manually, with care that the manual process followed the projected automated process as closely as possible, and did not produce results that required human insight. The event classification process can be assisted by software tools such as SMaRT [21], and is a necessary preliminary for conversion of scenarios to MSCs or other formalized notations for further analysis in that form.

The four business rules presented below as examples interact with a number of scenarios, including the ATM "Withdrawal" scenario sketched in Figure 2. The scenario has three parameters, the particular ATM involved, the person interacting with it, and the account to which the ATM card is tied (lines 2–4). The body of the scenario is a sequence of events similar to those encountered at many ATMs. Many of the events are parameterized to refer to the specific ATM, customer, or account of this scenario (for example, line 7 refers to the ATM parameter). The first two business rule scenarios contain events equivalent to the simple event named '`debit`' at lines 27–31. In the figures the equivalent events have been made textually identical (compare Figure 3; the event at lines 12–21, which is textually identical with it except for the particular references and the `coMatched` element, which does not participate in the comparison). The other event appearing in more than one is the typed event at line 24 (line 9 of Figure 3). This event matches the duration of the transaction, and temporally contains the event that debits the amount and the event that dispenses that amount of cash.

6.1. "Fee for over five savings transfers"

The first example is of a business rule whose goal is to discourage excessive transfers out of savings accounts:

> Each withdrawal from savings over five transactions/month will be charge a fee of $1.00 each.

At the credit union in question, every transaction that withdraws from a savings account is counted, whether from an ATM or by another method.

```
<scenario name="Withdrawal">
<parameter name="ATM" type="ATMtype"/>
<parameter name="cust" type="custType"/>
<parameter name="acct" type="acctType"/>
<eventChain relation="pm">
  <simpleEvent>
    <ref to="ATM">The ATM</ref> displays a list of
    <ref to="acct">the account</ref>'s subaccounts
    and "Please select the 'from' subaccount".
  </simpleEvent>
  <simpleEvent>
    <ref to="cust">The customer</ref> selects
    <variable name="fr">a subaccount</variable>.
  </simpleEvent>
  <simpleEvent>
    <ref to="ATM">The ATM</ref> displays "Please
    enter withdrawal dollar ammount".</simpleEvent>
  <simpleEvent>
    <ref to="cust">The customer</ref> enters
    <variable name="amt" which="any"
      type="DollarAmtType">the amount</variable>.
  </simpleEvent>
  <eventChain relation="D">
    <typedEvent type="Transaction"><ref to="ATM"/>
      <ref to="acct"/></typedEvent>
    <eventChain relation="m">
      <simpleEvent name="debit">
        <ref to="ATM">The ATM</ref> debits
        <ref to="amt">that amount</ref> from
        <ref to="fr">the selected subaccount</ref>.
      </simpleEvent>
      <simpleEvent>
        <ref to="ATM">The ATM</ref> dispenses
        <ref to="amt">that amount</ref>.
      </simpleEvent>
    </eventChain>
  </eventChain>
</eventChain>
</scenario>
```

Figure 2. Withdrawal scenario

We operationalized this business rule with the scenario FeeForOverFive of Figure 3. This scenario describes how the business rule can fail to be met; thus, it is a negative scenario (line 1), since matching of it in the ordinary sense corresponds to failure. Its event at lines 12-21 must co-match with the Withdrawal scenario's 'debit' event; for the events to match the same occurrence, the 'debit' event's reference to variable 'fr' must produce the same value as FeeForOverFive's parameter 'fr'. Finally, FeeForOverFive has a negative event at lines 22-25. This event succeeds if it does not match. In total, from the inside out, this scenario attempts to match transactions out of the same savings subaccount during a single month, but not transactions for which a fee was paid. The scenario as a whole attempts to match six such fee-less transactions. Because it is a negative scenario, if it matches then it fails; and it fails when the business rule has not been observed.

Having operationalized the business rule as a scenario, we can now analyze its interactions with the other scenarios in several ways.

- We can look for scenarios with unsuspected connections to the business rule. Having divided the simple events into equivalence classes, it is possible to identify other scenarios containing events of the same classes as the business rule scenario, and examine these scenarios individually and in combination in search of potential co-matching with the business rule.

- We can see what specific actions the business rule requires from each related scenario, in terms of the kind of events the scenario is expressed with.

- We can see what specific actions the business rule forbids, in terms of the events each scenario is expressed with.

```
<scenario name="FeeForOverFive" negative="true">
<parameter name="fr" type="subaccountType"/>
<eventChain relation="D">
 <typedEvent type="CalendarMonth">
   Any calendar month.</typedEvent>
 <iteration>
   <iterationsCounted min="6" max="unbounded"/>
   <eventChain relation="D">
     <typedEvent type="Transaction"><ref to="ATM"/>
        <ref to="acct"/></typedEvent>
     <eventChain relation="pm">
       <simpleEvent>
         <coMatched><ref to="Withdrawal.debit">
            <argument parameter="fr" ref="fr"/>
           </ref></coMatched>
         <variable which="any" type="ATMtype"
           name="ATM">An ATM</variable> debits
         <variable which="any" type="DollarAmtT">some
         amount</variable> from
         <ref to="fr">the subaccount</ref>.
       </simpleEvent>
       <simpleEvent negative="true">
         <ref to="ATM">The ATM</ref> debits the fee
         from <ref to="fr">the subaccount</ref>.
       </simpleEvent>
     </eventChain>
   </eventChain>
 </iteration>
</eventChain>
</scenario>
```

Figure 3. Scenario `FeeForOverFive`

- We can evolve the scenarios toward better satisfaction of the business rules, by changing the scenarios to add events the business rule demands and elide events the business rule forbids, in particular contexts.
- Because all these analyses are in terms of scenario matching, tool support based on matching can be implemented.

These benefits are provided when any business rule is operationalized as a scenario using our approach.

6.2. "Limit six savings transfers"

A similar business rule, attributed by the credit union to federal regulations, limits transfers from savings to six each month:

> You are currently allowed six (6) transfers per month from savings, as set by Federal Regulation D. The limit is reset on the first of every month.

On the face of it, this business rule seemed questionable when compared with the previous one. We following our process to operationalize it and identify related scenarios. It was operationalized as a negative scenario, matching only in contexts in which the business rule was not followed. Figure 4 shows the scenario. We compared its set of matched constellations with that of `FeeForOverFive`'s operationalization, giving savings subaccounts to both as arguments. The sets were not comparable, as each one matched contexts the other did not ("FeeForOverFive" matched a seventh savings transfer out, while "LimitSix" matched six savings transfers out without any fee paid). Our approach compared scenarios, but only for equivalence or generalization, and this was neither. This part of our approach was not helpful for this business case.

However, since the two business rule scenarios shared two of their events, and our approach did look for shared events, our approach did bring the two business rules to our attention. for further examination as possibly conflicting.

None of the ATM scenarios we elicited supported this business rule, as we had not observed that there was such a limit. Further tests with an ATM seemed to show that the rule was no longer active.

```
<scenario name="LimitSix" negative="true">
<parameter name="fr" type="subaccountType"/>
<eventChain relation="D">
  <typedEvent type="CalendarMonth">
    Any calendar month.</typedEvent>
  <iteration>
    <iterationsCounted min="7" max="unbounded"/>
    <simpleEvent>
      <variable which="any" type="ATMtype"
        name="ATM">An ATM</variable> debits
      <variable which="any" type="DollarAmtT">some
      amount</variable> from
      <ref to="fr">the subaccount</ref>.
    </simpleEvent>
  </iteration>
</eventChain>
</scenario>
```

Figure 4. Scenario `LimitSix`

```
<scenario name="StandaloneFastCash" negative="true">
<eventChain relation="D">
  <simpleEvent name="Standalone">
    <variable name="ATM" which="any">An
    ATM</variable> has no connection with
    the central server.</simpleEvent>
  <eventChain relation="p">
    <simpleEvent name="Cash1">
      A customer withdraws cash from
      <ref to="ATM">the ATM</ref> using
      <variable name="ATM-card" which="any">an
      ATM card</variable>.</simpleEvent>
    <simpleEvent name="Cash2">
      A customer withdraws cash from
      <ref to="ATM">the ATM</ref> using
      <variable name="ATM-card" which="any">the
      same ATM card</variable>.</simpleEvent>
  </eventChain>
</eventChain>
</scenario>
```

Figure 5. Scenario `StandaloneFastCash`

6.3. "Standalone mode fast cash"

Several credit union business rules apply only to ATMs in *standalone mode*, when the ATM does not have communications with the central server. An example is

> Each ATM in standalone mode will only allow one withdrawal per account, and only of the 'fast cash' amount.

We operationalized this rule with the scenario sketched in Figure 5.

The scenario `StandaloneFastCash` is a negative scenario, one that fails to match the world if the systems described are functioning properly. It describes a counterexample for the BR-StandaloneFastCash business rule, in which the ATM is in standalone mode (event 'Standalone'), during (`relation="D"`) which interval the same ATM card is used first for one withdrawal (event 'Cash1') and then for a second (event 'Cash2') from the same ATM. This scenario matches every way the business rule can fail to be followed.

The next step in our approach was to identify the scenarios that this business rule could co-match with. In our case study, there were no scenarios specific to stand-alone mode, as we had never observed it or any sign of its existence during elicitation. Therefore the normal-case scenario for withdrawing cash matched twice in succession along with cash-withdrawal events in the business rule's negative scenario, indicating a violation of the business rule. The chosen solution was to create

```
<!-- Authorization for ATM transactions -->
<simpleEvent>
A customer inserts an ATM card for
an accoutn into the ATM. </simpleEvent>
<simpleEvent>
The customer enters the PIN for the ATM card
into the ATM. </simpleEvent>
<simpleEvent>
The ATM verifies that the PIN matches the ATM card.
</simpleEvent>

<!-- Authorization for teller transactions -->
<simpleEvent>
The customer tells or shows the teller the account number.
</simpleEvent>
<simpleEvent>
The customer shows his or her drivers license to the teller.
</simpleEvent>
<simpleEvent>
The teller verifies that the name on the license
matches an owner of the account. </simpleEvent>

<!-- Authorization for mail transactions -->
<simpleEvent>
The customer writes a letter giving his or her name
and account number and requesting a specific transaction.
</simpleEvent>
<simpleEvent>
The customer signs the letter.
</simpleEvent>
<simpleEvent>
The teller verifies that the name is of an owner of the account,
and that the signature matches the one on file.
</simpleEvent>
```

Figure 6. Authorization in several contexts

new scenarios specific to stand-alone mode, and edit the current scenarios to exclude them from matching in stand-alone mode. This made direct use of the knowledge we had gained by operationalizing the business rule.

6.4. Crosscutting authorization rule

The final business rule presented here was, strictly speaking, not part of the case study, as it was suggested by a colleague rather than obtained from a credit union. A little reflection and investigation show that a credit union always requires a customer to show authorization for the requested transactions, no matter how those transactions are handled: at an ATM, a teller window, by physical mail, etc.

The suggested business rule was

> A customer's authorization must be confirmed before he or she can request a transaction.

This is a reasonable business rule, and one that is presumably a generalization of all the specific business rules requiring authorization of an ATM customer, a teller customer, etc.

The operationalization of this business rule was beyond the present capabilities of ScenarioML, primarily because each specific sub-rule's scenario had a list of parameters, and each list was different. We considered several approaches but concluded none was sufficient. Figure 6 shows three specific operationalizations of the hypothesized business rules, for ATMs, teller windows, and mail transactions. We considered what a scenario language would need in order to express these scenarios, and concluded that while it was conceivable, it might not repay the effort.

7. Related work

Although business rules have been widely discussed in the trade press and are becoming more prominent in the software industry, there has been relatively little research on them in the requirements community. Leite et al. examine the connections between business rules and the requirements baseline, using business policies as a useful viewpoint from which to understand and analyze requirements evolution [11]. Rosca et al. propose a metamodel and methodology for explicitly modeling business rules. The metamodel and methodology support and guide work with business rules, including analysis of a set of rules and their degree of conformity with an enterprise's policies [18]. Wan-Kadir et al. explore the relation between business rule evolution and evolution of related software systems [26].

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/interuption are described by Dardenne et al. [8], the Message Sequence Charts specification [10], Maiden [13], and many others. Basic scenarios have been extended with specific sets of temporal relations between events, including important work by Maiden [13] and by Breitman et al. [7] Episodes under one name or other appear to have been nearly universal in scenario and use case languages, including notably Potts et al. [17] and the OMG [16]. Negative scenarios, anti-scenarios, and misuse cases have been explored and shown effective by Alexander [1], Hope et al. [9], Sindre et al. [20], and Uchitel et al. [25]. The structure of ScenarioML is built squarely on this work; perhaps the most distinctive features of ScenarioML are its use of parameterization, variables for newly bound entities, quantification for restricting variable bindings, and anaphora, its emphasis on matching the world as the basis for the meaning of events and scenarios, and co-matching so that one scenario may be used to constrain the context of another.

Perhaps the most successful approaches to date for analyzing collections of scenarios have been those that first translate the scenarios into a more formal structure such as labeled transition systems. Uchitel et al. have achieved striking and basic results in this way [24], [12]. Our work is an early investigation into the possibilities of analyzing scenarios on their own terms.

Finally, researchers have investigated the fundamentals of combining different forms of specification, notably a long line of work on Viewpoints [15], [19] and model merging [23] Our work is much less general and specifically focused on one specific pair of forms.

8. Lessons learned and future work

In this paper we discussed an approach for evaluating business rules and scenarios together, by operationalizing the business rules as scenarios, mediating their interactions with other scenarios theough co-matching events, and examining their combination through a consideration of the occurrences they match. We make use of ScenarioML's flexible support for parameterization, variables and their quantification, and anaphora to increase the range of events that can co-match. Finally, we make use of negative scenarios to increase the expressive power of scenarios and match a broader yet focused set of occurrences, the better to cover the cases that follow a business rule and conversely to cover the counterexamples that fail to follow it.

Results from our first case study, presented in this paper, indicate that this approach for analyzing scenarios and business rules in combination is more promising than we had expected. While it is clear that a practical application of the approach will require software support, at the same time we see that there are opportunities to take advantage of prototype automation for scenario matching tasks, and the use of ScenarioML highlighted the possibilities of support of other tasks.

During the course of the case study, we learned that matching scenarios manually is more challenging that is first apparent. We found that our intuition about precisely which scenarios would interact how was incomplete or inaccurate in many cases. Manually working through the matching process was tedious, as expected, but also surprisingly error-prone. The equivalent of a "scenario match calculator" would be of great utility for this approach; among other tasks, it might evaluate two co-matching scenarios to determine if the intersection of the sets of occurrences each matches is empty, or equal to the set of one or the other scenario.

Finally, we were pleasantly surprised at the expressibility of scenarios in ScenarioML in practice, both for matching a broad swath of constellations and for matching quite focused sets. We did note it has limitations as well. However, its possibilities for extending the ease and power of ordinary scenario practice may be of considerable interest.

Our future work includes the continuation of a case study on ScenarioML as a language for expressing software requirements in the form of scenarios. We will continue to apply scenarios and ScenarioML in addressing areas outside requirements per se, such as software testing, especially using goals and plans to do more effective specification-bsed testing; the connection between requirements, software architecture, and design; and computed societies and the areas connected to it. As ScenarioML has become more stable and mature, it is more practical to pursue software support for its use. We are particularly interested in the possibilities of using Allen's interval algebra with event chains and partial orders, and in investigating the extent to which calculations in it for these restricted but useful cases can be built on in working with scenarios. We are presently doing work for Unisys Corp. to investigate the connection and synergy between scenarios and business rules, of which the present research is a first small step.

References

[1] I. F. Alexander. Initial industrial experience of misuse cases in trade-off analysis. In *10th IEEE Joint International Conference on Requirements Engineering (RE'02)*, pages 61–70, Sept. 2002.

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

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

[4] T. A. Alspaugh. Temporally expressive scenarios in ScenarioML. ISR Technical Report UCI-ISR-05-06, Institute for Software Research, University of California, Irvine, May 2005.

[5] E. Baumer, B. Tomlinson, M. L. Yau, and T. A. Alspaugh. Normative echoes: use and manipulation of player generated content by communities of NPCs. 2006. To appear: *AIIDE'06*.

[6] 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*, Dec. 1992.

[7] K. K. Breitman, J. C. S. d. P. Leite, and D. M. Berry. Supporting scenario evolution. *Requirements Engineering Journal*, 10(2), May 2005.

[8] A. Dardenne, A. v. Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, Apr. 1993.

[9] P. Hope, G. McGraw, and A. I. Antón. Misuse and abuse cases: Getting past the positive. *IEEE Security and Privacy*, 2(3):90–92, 2004.

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

[11] J. C. S. d. P. Leite and M. C. Leonardi. Business rules as organizational policies. In *Ninth International Workshop on Software Specification and Design (IWSSD'98)*, pages 68–76, Apr. 1998.

[12] E. Letier, J. Kramer, J. Magee, and S. Uchitel. Monitoring and control in scenario-based requirements analysis. In *27th International Conference on Software Engineering (ICSE '05)*, pages 382–391, May 2005.

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

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

[15] B. Nuseibeh, A. Finkelstein, and J. Kramer. Fine-grain process modelling. In *Proceedings of the Seventh International Workshop on Software Specification and Design*, pages 42–46, Dec. 1993.

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

[17] C. Potts, K. Takahashi, and A. I. Antón. Inquiry–based requirements analysis. *IEEE Software*, 11(2):21–32, Mar. 1994.

[18] D. Rosca, S. Greenspan, and C. Wild. Enterprise modeling and decision-support for automating the business rules lifecycle. *Automated Software Engg.*, 9(4):361–404, 2002.

[19] M. Sabetzadeh and S. M. Easterbrook. An algebraic framework for merging incomplete and inconsistent views. In *13th IEEE International Requirements Engineering Conference (RE'05)*, pages 306–318, 2005.

[20] G. Sindre and L. Opdahl. Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44, 2005.

[21] W. Stufflebeam, A. I. Antón, and T. A. Alspaugh. Smart — scenario management and requirements tool. In *11th IEEE Joint International Conference on Requirements Engineering (RE'03)*, page 351, Sept. 2003.

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

[23] S. Uchitel and M. Chechik. Merging partial behavioural models. In *SIGSOFT-2004/FSE-12: ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 43–52, 2004.

[24] S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE Trans. Softw. Eng.*, 29(2):99–115, 2003.

[25] S. Uchitel, J. Kramer, and J. Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Trans. Softw. Eng. Methodol.*, 13(1):37–85, 2004.

[26] W. M. N. Wan-Kadir and P. Loucopoulos. Relating evolving business rules to software design. *J. Syst. Archit.*, 50(7):367–382, 2004.