Supporting Scenario Evolution*

Karin Koogan Breitman† Julio Cesar Sampaio do Prado Leite† Daniel M. Berry§
†Departamento de Informática §Computer Science Department
Pontifícia Universidade Católica University of Waterloo
Rio de Janeiro, RJ 22453, Brazil Waterloo, Ontario N2L 3G1, Canada
karin@les.inf.puc-rio.br julio@inf.puc-rio.br dberry@uwaterloo.ca

ABSTRACT

Scenarios have been shown to be very helpful in identifying and communicating about requirements for computer-based systems (CBSs). However, they appear not to be applicable to the rest of the CBS development process. Making scenarios more useful for the entire software development lifecycle requires integration of scenarios to other representations used during CBS development. This integration is achieved with tracing technology. Having integrated scenarios into the entire software development lifecycle creates the necessity to maintain scenarios through the inevitable changes that they and other documents undergo and to subject them to configuration management. We have prototyped automated support for full-lifecycle scenario management and have applied it to some non-trivial systems.

1 INTRODUCTION

In recent years, a growing number of researchers have gravitated to the use of scenarios as a means to improve communication among the stakeholders of a computer-based system (CBS) under construction. Scenarios have been shown to be very helpful in identifying and communicating about requirements for CBSs [12, 51, 38, 13, 49].

1.1 Scenarios

In a nutshell, a scenario is an informal description of a situation [63] in a CBS's environment and of a way that the CBS can be used. Sutcliffe amplified this idea with by saying that a scenario is a collection of "facts describing an existing system and its environment including the behavior of agents and sufficient context information to allow discovery and validation of system requirements" [55].

This paper's definition of "scenario" tries to capture a much bigger picture than in other treatments of either scenarios or use cases. We capture also important information about the environment, the stakeholders, and the relationships between these even before the requirements of the CBS are elicited.

Every bit of information needed to describe a situation or desired action of the CBS is in the situation's scenario specification. Each scenario specification thus specifies a whole set of scenario instances or scenario *outcomes*.

* This paper is an amalgamation and extension of the conference papers numbered 8, 9, 10, and 11 in the references of this paper.

When this paper talks about relationships between scenarios, it is talking about relationships between the whole sets of outcomes and not between the outcomes. Thus, what this paper calls "scenario" is closer to what is often called "use case" in the literature. An instance of a scenario, what this paper calls a "scenario outcome" is often called a "scenario" in the literature.

The specification of a scenario is composed of several elements:

- title,
- context,
- resources,
- actors,
- goals,
- episodes, and
- exceptions,

each of which contains mainly natural language text. The *title* identifies the scenario. The *context* gives at least one of a physical location at which, a time during which, or a precondition upon which the scenario executes. The *resources* are the physical objects or information that must be available for the scenario to execute. The *actors* are the persons, organizations, devices, or other systems that have roles in the scenario. Each actor is a role, and several actors may reside in one person, organization, device, or system. The *goals* must be satisfied by the execution of the *episodes*. The episodes represent the main course of action of the scenario, but include also variations and alternatives. While executing episodes, *exceptions* may arise, signaling obstacles to achieving the goals. What episodes do is called *behavior* [39].

Episodes are related to steps of a scenario outcome as scenarios are related to scenario outcomes. That is, each episode is an abstraction of a step just as a scenario is an abstraction of a scenario outcome.

There are three "database"s that figure in the discussion. The first is the *Scenario Base (SB)* in which all versions of all scenarios needed for the development of one CBS are stored. The second is the *Requirements Baseline (RBL)* in which all artifacts, including the SB, that are needed for requirements engineering for the development of one CBS are stored. The third are the database systems that are used to implement the others. Since the first two kinds have their own specific names, the third kind are called just *databases*.

1.2 Problems with the Use of Scenarios

The inherent informality of scenarios allows them to be used without methodological or tool support. This fact can be both an advantage and a disadvantage. It is an advantage because no special notation is needed to express them, and users can read them without any special training. It is a disadvantage because systematic use of scenarios is made more difficult.

Scenarios, like other artifacts, persist throughout the lifecycle of a CBS [29, 27, 45, 12, 34, 51, 54, 38, 42, 2, 21, 60, 13]. Thus, it is only natural that they undergo change, that they evolve, during the CBS development process.

These changes are mostly the result of gains in understanding of the CBS to be built [62]. They are manifested as additions to, subtractions from, or changes to scenario contents. Thus, it is necessary to manage the evolution of scenarios.

In reality, scenario evolution is a very complex process that goes beyond the creation of new versions. Typically scenarios merge, divide, and compose with other scenarios, making overall understanding difficult and impeding the tracing* of their contents [31]. Weidenhaupt *et al* describe the difficulty of maintaining consistency among many scenario versions and the lack of support for change management [60]. Rolland *et al* observe that currently available scenario representations are deficient in providing for a process in which scenarios can be created and managed [50].

Thus, tracing is as important for scenarios as for other software artifacts. It is necessary to capture information on the decision making process or record authorship and responsibility for the scenarios. This information is crucial to maintain requirements traceability, both backward, to the sources of the requirements, and forward, as the development process takes its course.

1.3 Our Approach to Solve the Problems

To understand how to use scenarios effectively and to understand scenario evolution in practice, we have conducted a number of empirical case studies that shed light on the complexities of scenario evolution. Some preliminary findings, mostly regarding scenario relationships, have already been reported [10, 9, 11]. The contribution of these earlier works is condensed in the scenario evolution taxonomy that describes all observed relationships between and operations on scenarios.

These studies have allowed us to produce the Scenario Evolution Model (SEM), a model of scenario-based CBS development that explains the dynamics of CBS development in general, while providing a multiple tier approach to scenario evolution, based on a set of operations and relationships. Based on this model, we see an organization of scenarios, an architectural framework, the Scenario Evolution Framework (SEF), that allows a systematic, customizable approach to scenario evolution. The SEF is based on domain knowledge embodied in the SEM and takes into consideration other CBS management issues, such as version and configuration management and tracing. It can be used in any open-ended, general CBS development environment and can be tailored, through the use of hot spots, to specific user needs.

We have taken the SEF as requirements and have built a prototype of a tool, the SET (Scenario Evolution Tool), that meets these requirements and provides automated support to scenario evolution. It is available for download from [53]. We describe our approach though examples derived from a case study. These examples were generated with the aid of the SET. We have validated the SET's effectiveness by using it to manage scenarios for a life-like, non-trivial application and for a real-life, non-trivial application.

---

* We use the correct term for the process, "tracing" [32], rather than the more popular, but incorrect term "traceability", which means only "the ability to trace".

1.4 Outline of the Rest of the Paper

Section 2 describes one case study from which the SEM was derived. Its subsections describe specific operations on scenarios that were carried out during the development of one particular CBS. Section 3 details the SEM and its implications for the SEF. Section 4 describes at length the SEF, which provides requirements for the SET. Section 5 introduces the SET, its structure, and its instantiation to deal with a particular SEF. Section 6 compares our work to other work, and Section 7 concludes the paper.

The contributions of this paper are the SEF and the prototype SET. To justify the SEF and the functionality of the SET, it is necessary to review lessons learned from case studies of scenario-based CBS development and to review a SEM derived from the lessons learned.

The original work was done in Portuguese, in which the word for scenario is "cenário". Consequently, in figures and in the explanatory text, scenarios are often called "C*n*". Also, when the contents of a scenario is not critical to understanding the concept being discussed, that scenario is called by a non-meaningful name or number, e.g. "C5" or "11".

Due to space limitations, many details are omitted in favor of focusing on the lessons learned. Fuller details are always available in Breitman's Ph.D. dissertation [7].

2 SCENARIO EVOLUTION CASE STUDY

This section presents the case study we use to build the scenario evolution taxonomy and framework. The case study was first presented during a requirements engineering workshop at Schloß Dagstuhl, Germany in June 1999 [20] and describes a CBS, called the Light Control System (LCS) whose job is to control the lighting of the Computer Science Department building at Universität Kaiserslautern. The complete description of the LCS can be found in: `http://rn.informatik.uni-kl.de/~recs/problem/`

The case study had us taking a set of scenarios for the problem through a total of eight releases, Rel 1, Rel 2, ..., and Rel 8. The scenarios are stored in a special database called the *Scenario Base* (SB). The first three releases were specifications, Spec I, Spec II, and Spec III. The next three releases were designs, Design I, Design II, and Design III. The last two releases were implementations, Impl I and Impl II. These names are alternate names for Rel 1, Rel 2, ..., and Rel 8. The process leading to Rel *n* starting from nothing, in the case n=1, or from Rel n–1, in all other cases, is called Transition *n*. Thus, for example, Transition 2 leads to Rel 2, also known as Spec II, as a modification of Rel 1, also known as Spec I.

Each artifact, including scenarios, undergoes changes. Each new version of an artifact gets a name which combines the artifact name and a version number. Thus, Version 3 of Scenario 4 is called Scenario 4.3. Not every artifact is changed in each release. Thus, the version of an artifact may be the same across several releases.

Subsection 2.1 gives definitions of relationships between scenarios and of operations on scenarios whose importance became apparent during the case studies. Subsection 2.2 gives a general statistical overview of the evolution of the scenarios of the case study. The remaining subsections consider the details of this evolution. Subsection 2.3

discusses evolution during the transition from one release to the next, Subsection 2.4 discusses evolution of a scenario across transitions, and Subsection 2.5 discusses the joint evolution of a scenario and its related artifacts. Thus, the case studies served as a vehicle for requirements engineering for the scenario evolution system described in the later sections of this paper.

In these subsections, we are trying to give the reader a feeling for the varieties of relationships and operations that come to play during scenario evolution and that must be supported by any scenario evolution system. For concreteness, we use data from the LCS case study. The same case study will feature in the later descriptions of the prototype tool. In an effort to give what must be a global view, we try to ignore the interior details of the scenarios involved. However, to explain why a relationship exists or why or how an operation is done, we sometimes need to expose a few details of the interiors of the scenarios involved. We hope that just the everyday understanding of words are sufficient to make the explanations clear. Readers with more curiosity about the details should consult Breitman's Ph.D. dissertation [7].

2.1 Scenario Relationships and Operations

In order to make this paper self contained, this subsection gives brief definitions of a variety of scenario relationships and operations. Fuller definitions and some examples are available elsewhere [8].

A scenario is connected to other scenarios in an intricate and complex network of relationships. In the course of the case studies, we have identified eight types of possible relationships, many based on a more fundamental relationship, *coincidence*, between scenarios and between corresponding scenario elements.

Two scenarios are said to present coincidence if, in the judgement of requirements engineer making the decision, there is a total or a sufficient partial match between corresponding elements of the scenarios' specifications. Two corresponding elements of two scenarios are said to present coincidence if, in the judgement of requirements engineer making the decision, there is a total or a sufficient partial match between the elements' contents.

1. A set of scenarios *complement* each other if they share a goal and present some coincidence in their contexts and resources. These scenarios should present also a lot of coincidence among their actors and episodes. Also, a set of scenarios that together serve a larger goal are considered complement each other.
2. A set of scenarios are *equivalent* to each other if they share a goal and present some coincidence in their contexts. The scenarios of the set are equivalent but not complementing if the scenarios of the set fail to present coincidence in their resources. Equivalent scenarios should present also a lot of coincidence among their actors and episodes.
3. Scenario *A* is *contained in* Scenario *B* if the context of *A* is fully contained within or is the same as the context of *B*. The two scenarios may present resource coincidence and should present a lot of coincidence among actors. At least one episode of *A* must be an episode of *B*.
4. Scenario *A* is a *precondition* of Scenario *B* if *A* appears in the context of *B*. Moreover, the two scenarios must present coincidence of at least one actor and they may present coincidence among episodes. The precondition

relationship is unique in that it allows some temporal aspect to be incorporated into a static relationship graph. It allows defining an execution sequence for scenarios and specifying that a specific scenario has to be completed before beginning another.

5. Scenario *A* *detours* to Scenario *B* if there is exceptional behavior in *A* that causes suspension of *A*, execution of *B*, and then resumption of *A*.
6. Scenario *B* is an *exception* for Scenario *A* if there is exceptional behavior in *A* that causes termination of *A* and execution of *B*.
7. Scenario *B* is *included* in Scenario *A* if *B* if during execution of *A*, execution of *A* is suspended, *B* is executed, and then execution of *A* is resumed, and if *B* is an independent scenario that contains behavior common to a number of other scenarios. The include relationship was proposed by Grady Booch and others in the context of use cases [5]. If the information in *B* were not isolated into a separate scenario, that information would have to be replicated in all including scenarios. Inclusion serves as a means to reduce overall redundancy in the design.
8. Scenario *A* is a *possible preceder* for Scenario *B* if *A* may or may not happen, but *if A* happens, it happens before *B* does. *A* is not a precondition of *B*, because *A* being a precondition of *B* implies that if *A* never happens, neither does *B*.

It is unfortunate that the properties of coincidence, complementation, and equivalence depend so much on human judgement and cannot be defined more objectively. However, the elements of a scenario specification are written in natural language, and these properties are judged in early stages of requirements engineering when many details are unknown or poorly understood or expressed. Therefore, there is no way other than by human judgement to measure coincidence, complementation, and equivalence.

The above defined relationships are static in the sense that each is described in terms of the internal component structures of the related scenarios. The fact that the relationships depend mostly only on the structure of scenarios facilitates automated assistance for the detection of relationships, a necessity for building a scenario evolution tool. As a matter of fact, the SET implements semi-automatic relationship detection based on similarity measures that are described by Breitman and Leite [8].

The scenario operations, on the other hand, describe the dynamics of scenario evolution. We make a distinction between an *interscenario* operation that involves two or more scenarios and an *intrascenario* operation that effects the contents of a single scenario. The operations reflect the complexities involved in the evolutionary process, making it clear that the scenarios do not evolve in an organized and linear fashion but, more likely, are merged and split and combined in a very haphazard way. Among the interscenario operations, the first group of operations, *Split*, *M-Split*, *Specialize*, and *Extend*, deal with information distribution among two or more scenarios.

1. Splitting separates one scenario into two or more independent scenarios. A split may be necessary during evolution to break a big block of information into smaller, more manageable pieces. Sometimes, design and coding decisions force an artificial division of the information in one scenario into several scenarios.
2. M-splitting is an extension of splitting that aims to isolate some basic behavior that is present in several

scenarios into a single, independent includable scenario in order to reduce redundancy.

3. Specialization of a scenario limits the scenario's possible outcomes or restricts the scenario to a more particular context.
4. Extension of a scenario extends the scenario's possible outcomes or extends the scenario to a larger context.

Clearly, each of specialization and extension is the other's opposite.

The second group of interscenario operations, *Fuse*, *Encapsulate*, and *Consolidate*, are opposites of *Split*.

1. Fusion of a set scenarios is done when the when the overlap between the scenarios of the set is too great. The result is a single scenario that combines the behavior of all the scenarios of the set.
2. Encapsulation is the result of the union of two scenarios that have a weak overlap into a single scenario that usually resembles one of the initial scenarios, with the other scenario taking a less important role.
3. Consolidation unites two scenarios when one is contained in the other. This operation is usually done to eliminate redundancy.

The third group of interscenario operations is *Exclude* and *Add*.

1. Exclusion is the removal of a scenario from the scenario base.
2. Addition is creation of a new scenario and its inclusion into the scenario base.

The intrascenario operations are *Include*, *Modify*, and *Remove*.

1. Inclusion adds new contents to an existing scenario.
2. Modification changes the contents of an existing scenario.
3. Exclusion removes contents of an existing scenario.

Any viable scheme for scenario evolution needs to document and otherwise deal with these scenario relationships and operations.

2.2 Statistics

Table 1 gives some important statistics about the eight releases. From this table, we observe that the number of scenarios in the SB at each release diminishes and stabilizes with advancing time. The initial set of scenarios was derived from the Language Extended Lexicon (LEL) [36] of the problem through application of a scenario-generation heuristic [40] that systematically generates a large number of scenarios containing much redundant information. We were able to refine this initial set into a smaller set as we gained greater understanding of the problem. From the Spec III release on, the number of scenarios in the SB at each release remained essentially constant. The initial set of changes were mostly interscenario and changed the number of scenarios in the SB, by excluding or adding scenarios or combining several. These mostly external changes gave way over time to changes that were mostly intrascenario and that did not change the number of scenarios in the SB. The last two columns of the table show the numbers of each type of operation carried out in each of the eight releases. For any row, the sum of these two column entries add up to the entry in the fourth column from the right.

Transition	Release	No. of Scenarios	No. of Relationships	No. of Operations Performed	Scenarios per Operation	No. of Inter-scenario Ops. Perf.	No. of Intra-scenario Ops. Perf.
Transition I	Spec I	23	21	23	1	23	0
Transition II	Spec II	17	16	9	1.9	7	2
Transition III	Spec III	10	14	5	2	3	2
Transition IV	Design I	10	14	10	1	0	10
Transition V	Design II	11	15	7	1.6	1	6
Transition VI	Design III	10	11	7	1.4	1	6
Transition VII	Impl I	7	4	5	1.4	4	1
Transition VIII	Impl II	7	5	7	1	0	7
Totals		96	100	73	—	39	34

Table 1: Statistics about Operations

Any viable scenario evolution scheme will have to be able to carry out all of these intra- and interscenario operations on the SB.

2.3 Evolution of Scenarios During a Transition to the Next Release

This subsection details the evolution of scenarios during one transition, namely Transition II, from one release to the next. For each operation performed in the transition, we examine what happens with each of the participating scenarios and try to determine parallels with the relationships held by the same scenarios before the operation was applied. We chose Transition II, from the release of Spec I to the release of Spec II, as the example for discussion, because this transition required both intra- and interscenario operators and is the most illustrative. Recall that release Spec I is the result of the application of scenario extraction heuristics to the LEL for the LCS application. Transition II was aimed at reducing the redundancy in, correcting, and reorganizing the information contained in the scenarios in the SB at the release of Spec I.

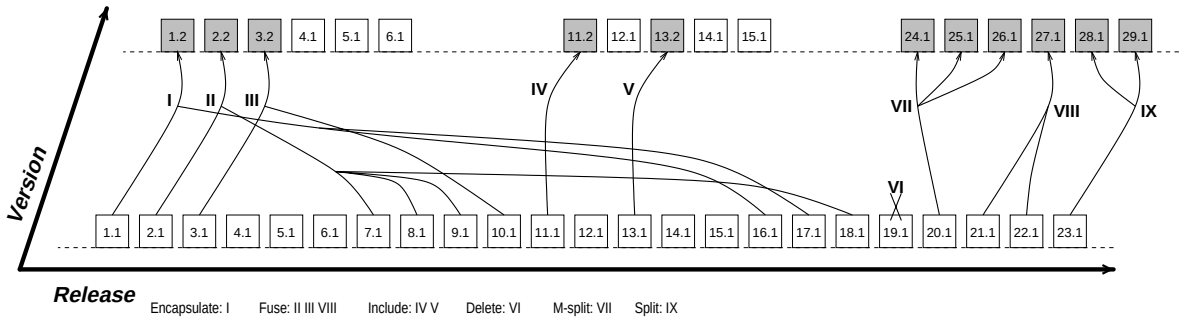

Figure 1: Operations Performed During Transition II

In Figure 1, the Spec II scenarios in clear boxes suffered no contents change during Transition II, while those in shaded boxes were the result of the application of some scenario creating or changing operation. A given scenario does not necessarily have to be changed in any one transition. Thus, a single version of one scenario may belong to

more than one release. Each of Scenarios 4.1, 5.1, 6.1, 12.1, 14.1, and 15.1 belongs to both releases Spec I and Spec II.

Of the nine operations illustrated in Figure 1, two are intrascenario, and the remaining seven are interscenario. The two intrascenario operations are *Include*s, and of the seven interscenario operations, one is *Encapsulate*, three are *Fuse*s, two are *Split*s, and one is *Delete*. Table 2 summarizes these operations, the scenarios involved and their results.

Operation Name	Identity Number	Involved Scenarios	Count of Involved Scenarios	Resulting Scenarios	Count of Resulting Scenarios	Type
Encapsulation	I	1.1, 16.1	2	1.2	1	Interscenario
Fusion	II	2.1, 7.1, 8.1, 9.1, 18.1	5	2.2	1	Interscenario
Fusion	III	3.1, 10.1 17.1,	3	3.2	1	Interscenario
Inclusion	IV	11.1	1	11.2	1	Intrascenario
Inclusion	V	13.1	1	13.2	1	Intrascenario
Removal	VI	19.1	1	—	0	Interscenario
M-Split	VII	20.1	1	24.1, 25.1, 26.1	3	Interscenario
Fusion	VIII	21.1, 22.1	2	27.1	1	Interscenario
Split	IX	23.1	1	28.1, 29.1	2	Interscenario

Table 2: Details of Transition II Operations

It is important to note the variety of the operations in Table 2. Some are intrascenario and do not change the number of scenarios in the SB. Some others are interscenario and work generally towards reducing the number of scenarios. Several scenarios disappeared during the transition as a result of contraction operations. The operations numbered I, II, III, VI, and VIII resulted in the removal of Scenarios 7.1, 8.1, 9.1, 10.1, 16.1, 17.1, 18.1, 19.1, 20.1. 21.1, 22.1, and 23.1. Note that only the labels of these scenarios have disappeared. Most of the information contained in most disappearing scenarios was moved into other existing scenarios or into brand new scenarios. Operation I incorporated the contents of Scenario 16.1 into Scenario 1.2, and Operation VII moved the contents of Scenario 20.1 into the new Scenarios 24.1, 25.1, and 26.1. Other operations resulted in the creations of new scenarios that did not exist in previous releases. Operation VIII fused the contents of Scenarios 21.1 and 22.1 into the new Scenario 27.1 and removed Scenarios 21.1 and 22.1 as no longer necessary because their contents are in the newly created Scenario 27.1. Operation IX split Scenario 23.1 to create Scenarios 28.1 and 29.1 and effectively removed the input scenario.

Let us consider the relationships among the scenarios in release Spec I. There is a connection between the relationships holding among the scenarios and the choice of operations performed on these scenarios. Figure 2 shows the relationships among the scenarios in Spec I, the input to Transformation I.

Operation I, *Encapsulate*, involves two equivalent scenarios. This kind of relationship is observed among scenarios that share the same goal at the same time and in the same context frames. In most cases, the same actors

are involved among the scenarios and sometimes even among the episodes. Encapsulation is normally applied to reduce redundancy in the SB. That the scenarios involved in the operation have equivalent contents is enough to justify applying the operation. For example Scenario 1.1, “malfunction occurs”, and Scenario 16.1, “malfunction”, are encapsulated into Scenario 1.2, “malfunction occurs”.

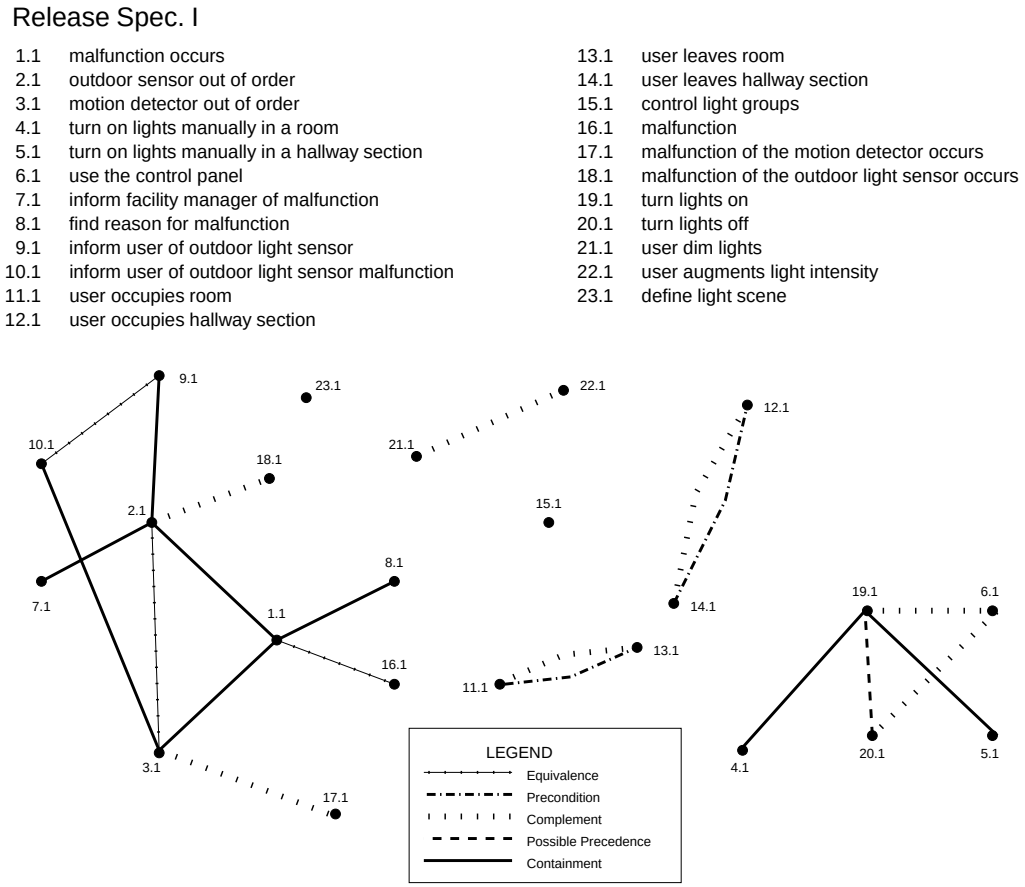

Figure 2: Relationships Among Scenarios in Spec I

Operation III is a *Fuse* of Scenarios 17.1 and 3.1 and 10.1. Scenarios 17.1 and 3.1 are complementary and Scenario 10.1 is contained in Scenario 3.1. The goal of a fusion is to unite scenarios that contain procedures that share data and and have disjoint but complementary behavior. In this case, one scenario detects a condition, the second contacts the responsible person, and the third deals with malfunction arising from the condition. Sometimes scenarios seem to negate each other, e.g., as do “reduce lighting” and “increase lighting”. However, if they act under disjoint contexts, and there is a single parameterized abstraction that captures both behaviors, e.g., “change lighting(direction)”, then they can be fused into a single scenario named after the parameterized abstraction. Such is the case in Operation VIII.

Any useful scenario evolution system will need to allow documenting relationships among scenarios to allow

users to see which operations to perform in a transition to the next release.

2.4 Evolution of a Scenario across Releases

This subsection examines the evolution of a single scenario through all releases of the CBS. Scenario 11, "user occupies room" (UOR), is the subject of the discussion, because it suffers both inter- and intrascenario evolution. Figure 3 shows this scenario's evolution. Some versions of this scenario appear in more than one release. For example, Version 2 appears in releases Spec II and Design I, and Version 3 appears in releases Design I and Design II.

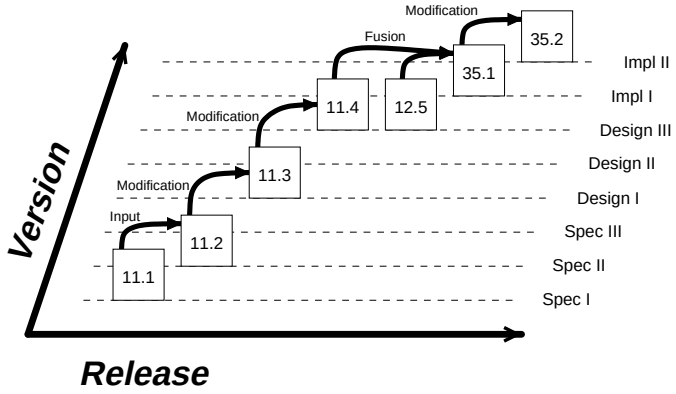

Figure 3: Evolution of Scenario 11 Through all Releases

The history of any scenario should be visible through the proposed SET. Figure 4 shows the information comprising the history of the scenario UOR. This information includes for each version of the scenario (1) data about the creation of the scenario version, i.e., the creation date, author, and person responsible, (2) the rationale behind the version, and (3) the contents of the version. (To save space in Figure 4, the figure indicates that the contents of Versions 2 and 3 are those shown in Figure 5.) This historical information should be accessible in any scenario evolution tool. The history of a scenario could be accessible via trace links from a simple list of the names of the versions of the scenario, as illustrated in the central window in the top of Figure 13.

Figure 4 and the top two windows on the left side of Figure 5 show the information and the contents of the first three versions of the UOR scenario. These windows are from the web site [53] for the case study. For each version of a scenario, the information about the creation of the version is available together with a list of the releases in which this version of the scenario participates. For example, the second version of Scenario 11 participates in the releases Spec I and Spec II. Comparison of the contents of the two versions shows that not only were the contents of the first version augmented, but they were also changed to produce the second version. In the first version, Episodes 3 and 4 contradict each other, because they suggest that after T1 minutes, both the default and chosen light scenes would be activated at the same time. The second version corrects this contradiction by giving different time limits for the establishment of each light scene.

Any scenario evolution system worth its salt must be able to trace through the history of versions of each

scenario in the SB.

Scenario: user occupies room
Version 1

First appeared in release: Spec.I	Date of creation: 9/11/99	Author: user 3
Appears in Release(s):	Spec.I	

Resulted from:

Operation: include	*Rationale:* The scenario was derived using heuristics from the Extended Lexicon of the Language	Person Responsible: user 3

Contents:

user occupies room	*Scenario contents as in Version 1 of Figure 5*

Scenario: user occupies room
Version 2

First appeared in release: Spec.II	Date of creation: 10/3/99	Author: user 2
Appears in Release(s):	Spec.II	Spec.III

Resulted from:

Operation: input	*Rationale:* We noticed some of the information in the scenario was incorrect. We made proper corrections	Person Responsible: user 4

Contents:

user occupies room	*Scenario contents as in Version 2 of Figure 5*

Scenario: user occupies room
Version 3

First appeared in release: Design.I	Date of creation: 11/23/99	Author: user 2
Appears in Release(s):	Design.I	

Can be traced to:

Artefact: CRC Cards
Detail: The functionality of this scenario is distributed in the following cards control panel, user, and room

Contents:

user occupies room	Goal: establish the procedure for occupied room Context: 4th floor of building 32, motion detector in order, user entered room Resource: value T1, outdoor light sensor last correct management value, Default light scene for this room, Chosen light scene value, Actors: user, Central Control, outdoor light sensor, Control panel, room Episodes: 1. user enters room 2. In case of malfunction, outdoor light sensor sends last correct measurement to room 3. user chooses light scene using the Control panel Exception: user input is not reasonable 4. IF room is reoccupied wihtin T1 minutes THEN activate last Chosen light scene stored in room 5. IF room is reoccupied after T2 minutes THEN activate Default light scene stored in room

Figure 4: History of Scenario "user occupies room" (UOR)

2.5 Evolution of Scenarios and Related Artifacts

We have previously introduced the Requirements Baseline (RBL) [38], a structure that captures and manages the evolution of all the artifacts generated during the lifecycle of one CBS. In this context, scenario evolution is only one of the many managed processes whose artifacts are kept in the RBL. Several other processes unfold in parallel. Among these is the evolution of the LEL, which is continually being updated in order to reflect changes to the CBS.

It is essential to trace the information contained in the scenarios, not only to other scenarios in the SB, but also to other software artifacts, including object models, LEL entries, and CRC (Class, Responsibilities, Collaborators) cards [61]. We distinguish between tracing to other scenarios and tracing to other software artifacts. The links of the former kind are called *homogeneous*, and the links of the latter kind are called *heterogeneous*.

We use Scenario 11, UOR, as an example of this facet of scenario evolution. Version 3 of this scenario, shown in Figure 4, appeared as a result of the updating operation performed for Release Design I. According to the "Can be traced to:" window, the functionality of this scenario is distributed over the CRC cards for the control panel, the user, and the room. There is a link from the scenario to each of the cards "control panel", "user", and "room". Figure 6 shows these CRC cards.

User	
Defines chosen light scene	room
Enters or leaves room or hallway section	control panel
set the value T1 of room	hallway section
set the default light scene of room	motion detector
set each ceiling light group of room	outdoor light sensor

Room	
keep the outdoor light sensor measurements	facility manager
store value T1	user
IF room is reoccupied within T1 minutes THEN activate last chosen light scene	control panel
IF room is reoccupied after T1 minutes THEN activate default light scene	
activate default light scene	
store chosen light scene	
store moment when person left the room	

Control Panel	
IF user input is not reasonable THEN system issues a warning	user facility manager

Figure 6: CRC Cards for Third Version of Scenario UOR

Recall that we built the scenarios starting from the LEL. Thus, some tracing comes automatically. The contents of the scenarios end up naturally linked to the relevant terms in the LEL. For other artifacts, the CRC cards for instance, tracing demands more effort, because the links have to be manually created.

In Figure 5, some of the terms or expressions in the scenario contents are underlined. An underlined term or expression has an entry in the LEL, and the underlined word or expression is a link to that entry. Figure 5 shows also the LEL entries for several terms appearing in different versions of scenario UOR. In the SB, the figure's arrow

between an underlined term and the bulb containing its LEL entry would be a tracing link.

As previously mentioned, the LEL is evolving constantly, as are the scenarios. Thus, the definitions of the terms also suffer modifications during CBS development. To illustrate this evolution, we show the evolution of the term "room" in Figure 5. We use Versions 1 and 3 of Scenario UOR and Version 1 of Scenario "Person occupies place" that resulted from the fusion of Scenario UOR with Scenario "user occupies hallway section" to show the evolution of the term "room", which is present in all of these scenarios. In this sequence of scenario versions, the definition of "room" evolves from being "a representation of physical space" to being the Java class that encapsulates the details of the room.

Any viable system for scenario evolution needs to be able to track the evolution of a variety of artifacts, not just scenarios.

2.6 Summary

This section has presented snippets from the case study used to validate and test the scenario evolution taxonomy. From the experience we gained during the elaboration of this case study, we refined the taxonomy and included aspects that go beyond the scenario evolution process. We learned that the static aspects of scenarios as products, represented through the relationships among the scenarios of a release, are intimately related to the evolution of the same scenarios and thus cannot be dissociated. Finally, the SEM, the SEF, and the SET should support all the relationships and operations that we have noted. Moreover, the same set of relationships and operations have sufficed for two other case studies we have performed. Therefore, we are confident that we have identified all the relationships and operations needed for carrying out scenario evolution. What we learned allowed designing the SEM that is the subject of the next section.

As we describe the SEM, SEF, and SET, we refer to figures showing screen snapshots from the prototype SET, applied to the case study of Section 2. The reader will notice that these snapshots show information, relationships, and operation results described throughout Section 2.

3 SCENARIO EVOLUTION MODEL

The previous section used parts of a case study to illustrate the complexities involved in scenario evolution. Our claim is that scenarios should be used to support all stages of CBS development. Thus, it is essential to understand how scenarios evolve. The experience gained in the case study allowed not only refining the scenario evolution taxonomy but also beginning to understand the mechanics of scenario evolution as a whole.

The result is the Scenario Evolution Model (SEM) depicted in Figure 7. The SEM is partitioned into three levels: *process*, *product*, and *instance*. The process level has the scenario transformation operations that transform also the Scenario Base (SB). The product level has several notations for schemas to represent scenario knowledge [54, 51, 37, 29, 45, 34]. The amount of detail represented by each notation is what determines the nature of the relationships held among scenarios expressed in that notation and how to detect them. The instance level has the actual data

in the formats dictated by the schemas at the product level.

Our understanding of the scenario evolution process is described as follows. At the process level, scenario evolution is a series of operation applications, each of which has an effect on one or more scenarios, and as a result, modifies the existing scenario configuration, to yield a new release. These operations were programmed using a single notation, at the product level. From the instances of the scenarios that contain information particular to a given domain, we deduce the relationships and dependencies among the scenarios of one release. These relationships and dependencies, in turn, determine the choice of operation that can be applied over the SB to yield a new release.

The variety of relationship types is intrinsically dependent on the representation chosen to describe the scenarios. The more detailed the representation, the more relationships are possible. Currently, a number of scenario representations using natural language are available in the literature [54, 51, 37, 29, 45, 34, 21, 50], each with a different amount of detail. It is reasonable to expect that a notation with more detail contains more information, thus allowing the inference of more relationships.

It is from the relationship network among scenarios of one release that we can determine the relevance and validity of the application of any given operation. For example, from the observation that *A* is a precondition to *B*, we decide that if *A* were removed from the SB, *B* would become inaccessible. Thus, *A* should not be removed.

Thus, it is possible to understand the scenario evolution process as a chain reaction among the elements present in the SEM. There are relationships at the instance level, determined by the component structure imposed by the chosen representation at the product level, and these relationships end up determining the applicability of the operations at the process level. Scenario evolution is thus a dynamic process, heavily based on relationships that hold among scenarios, which in turn are both determined and limited by the notation used at the product level.

The SEM thus supports the documentation of all relationships and the application of all operations mentioned in Section 2.

4 SCENARIO EVOLUTION FRAMEWORK

Scenario evolution requires generating and maintaining large volumes of highly connected written material that undergoes many changes. Consistency within a version of a scenario, among versions of scenarios, and among changes must be maintained. Thus, automated support of scenario evolution is essential. As with any other software artifact or collection thereof, version management, configuration management, and tracing are needed. This section builds a Scenario Evolution Framework (SEF) comprising a database for visualization, version management, configuration management, and tracing, from the SEM and the lessons learned from the case studies.

4.1 Scenario Visualization with the Scenario Base

Scenarios describe day-to-day situations that potentially have different outcomes. A practical question arises: “What is the best format in which to represent scenarios in order to understand complete scenarios and to validate their contents with clients?” The biggest problems with scenarios and requirements in general is ensuring their

completeness, in the sense of covering the problem. Ensuring completeness requires discovery of all missing scenarios and all exceptions to any scenario. Fortunately, our experience has been that the mere reading of scenarios tends to encourage the discovery of other scenarios and exceptions, in what is called the ripple effect. We therefore want to allow a visualization that encourages this ripple effect.

An effective way to encourage this ripple effect is to set up an easily traversed database that provides multiple views of the SB [18]. This database would typically contain the scenarios and related artifacts such as LEL entries, the SRS (Software Requirements Specification document), object models, and CRC cards. Under one such scheme, a release of the SB is a horizontal slice of the database, while one scenario's history is a vertical slice of the same database. Figure 8 shows the history of the Scenario "malfunction occurs" (MO) implemented in the proposed SET. Another possibility is to view the artifacts related to a single version of a single scenario. Figure 9 shows the LEL and the object model of the CBS related to Version 4 of Scenario MO.

Still another possibility is a variation of the history of a single scenario using the scenario operations as the basis of navigation. Figure 10 shows the history of scenario operations applied to Scenario MO. The figure illustrates every operation with which the scenario was involved during its life cycle and all other scenarios involved in these operations. The number of versions that were involved with Scenario MO is 6, which is a large number for this particular case study. When we consider that the number of links to LEL entries and other software artifacts per version of this scenario is around a hundred, it is clear that an automated, systematic scenario management process is essential.

4.2 Version Management

We believe that existing version management mechanisms support scenario evolution in a natural way. Starting from basic information elicited in the beginning of a CBS construction lifecycle, *natural evolution* refines this information as a result of increasing understanding of the CBS's macrosystem and its implications. This kind of evolution can be executed simply, using classical version control mechanisms that keep a history of changes.

Version control is defined as the process of coordinating and managing the development of evolving objects [26]. In scenario-based CBS development, as well as in other fields of endeavor, evolution is characterized by successive refinements of the involved objects. In many situations, it is useful to keep all intermediate versions of these objects, for future reference or for backtracking. The number of those intermediate versions may be very large, necessitating automated support for version control [57].

Section 2 describes some data drawn from one of several CBS developments that we conducted as case studies. We observed that during CBS development, scenarios suffer a great deal of modification. We observed also that scenario evolution is far from being a simple refinement process and cannot be managed by version control mechanisms alone. In reality, scenario evolution has a very strong transformational element. Rare is the scenario that remains unaltered during the process. Merging, splitting, consolidation, and addition of new scenarios are just a few of the possibilities for scenario transformation. Figure 11 tries to illustrate the complexities involved. The figure

shows two axes, version and release. They are completely independent axes, i.e., a scenario does not have to change version every time a new release is launched, and a scenario may undergo several versions before the next release. In the case study, there are a variety of different change histories. One extreme example is scenario C3 that appears first in Version 1 and remains unaltered during five releases. Another extreme example is Scenario C1, which gets a new version in each release. In general, a new version of at least one scenario is made at each application of some operation over a scenario or a group of scenarios. We have not only new versions of existing scenarios, but also creations of new scenarios, e.g., Scenario C5.V4 evolved into Scenario C5.V5 and generated a brand new Scenario C6.V1. It is also possible that changing one scenario contributes to changing other scenarios, e.g., changing Scenarios C2.V1, C4.V1, and C1.V4 resulted in Scenarios C1.V5 and C4.V2 being changed from previous versions. In general, as a result of the application of operations, new versions of scenarios are created or destroyed.

A scenario version may be based on one or more scenarios including its own previous versions. Thus, versioning maps naturally to a directed acyclic graph [17], rather than to a sequential graph or a tree. Superimposed on Figure 11 is an acyclic graph made up of black and white curved lines; the line is white when its background is black, and the line is black when its background is white. The left side graph shows the single Scenario C1.V4 both deriving a new version of itself, C1.V5, and, in conjunction with scenarios C2.V1 and C4.V1, deriving Scenario C4.V2. The right side graph shows the Scenario C5.V4 evolving into Scenario C5.V5 and originating the new Scenario C6.V1.

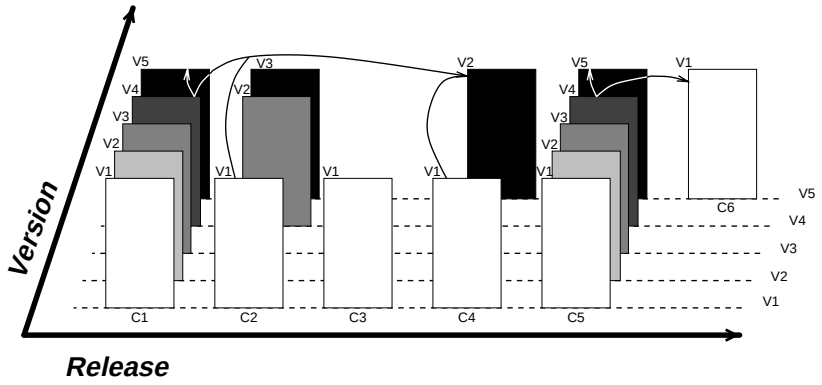

Figure 11: Example of Scenario Evolution

In order to trace the information encoded in a scenario, it is necessary to keep a record of the scenario's changes in a way that allows recreating the steps that led to any specific version. It is necessary to keep a history of the contents of versions in a way that allows reconstructing any specific version. Such is the function of a version control system [57].

4.2 Configuration Management

In a classical approach to configuration management [4], there should be for each version, information about the creation of the version. Typically, this information includes items such as the date and time of creation, the

author, and comments about rationale [43]. We adopted PUC-Rio's Requirements Baseline (RBL) [38] as the model for configuration management for use with the SEF, with the release taken as the unit of configuration management.

4.3 Traceability

Finally, the last requirement for SEF is for traceability of the information involved [19, 47, 48]. Traceability is defined as the ability to identify the origins of requirements [23]. These origins may be users, clients, other CBSs, documentation or any other sources of information directly related to the requirements. The task of maintaining traces is extremely complex and labor intensive.

So far we have discussed scenario evolution focusing on scenarios alone, i.e., versioning of scenarios and the launching of new releases of scenarios. However, independently of the method being used, the development of a CBS is characterized by the elaboration of a series of artifacts that describe its many components and stages. Among others and depending on the method chosen, there are object models, CRC cards, glossaries, data models, conceptual models, and entity–relationship models. Each of these artifacts reflects the rationale behind the stage of development in which it was conceived. As development unfolds, new scenarios are generated, and they must reflect the decisions captured in the other artifacts from which they were generated. Thus, it must be possible to trace from scenarios to these other artifacts and the other way around. In order to allow this tracing, it is essential to make those connections explicit and part of the SEF.

4.4 Summary

This section has described a SEF that includes a visualization database, version management, configuration management, and tracing, all of which are essential for scenario-based CBS development. The SEF was based on the SEM and the lessons learned from the case studies and provides the framework for carrying out all operations noted as necessary in Section 2.

5 SCENARIO EVOLUTION TOOL

We prototyped the SET (Scenario Evolution Tool) that implements the Scenario Evolution Framework (SEF) presented in Section 4. As shown in Figure 13, the SET has a three-layer architecture:

1. The bottom layer has the database, responsible for the storage of the scenario contents as well as information related to the evolution itself, i.e., rationale, operation log, and configuration. In addition, process information, such as date, hour, creation and links to other objects, is stored in the database, to ensure traceability.
2. The middle layer has the software responsible for processing the Scenario Evolution Model (SEM) information. Some of the software compiles queries entered by users into the format that is accepted by the database. Other software carries out the actual queries in the database. In addition, this layer implements the navigation capabilities that allow tracing among objects.
3. The top layer has the user-interface software. This layer enables all interactions with the SET and is the only layer to which users have access; the other two are hidden behind this layer.

The SET prototype was implemented using the relational database MS Access running on a Windows NT platform. This choice of database software is justified by (1) its general availability on this widely available platform, (2) its big installed base, and (3) its portability compared to other similar products. Remote access tests were conducted via the Internet, from both in Brazil and abroad, to demonstrate that the prototype's performance is adequate.

The software layer was implemented using Version 2.0 of Active Server Pages (ASP) technology, which provides mechanisms to access databases from remote locations using ODBC derivatives, and which makes it easy to create and manage dynamic pages. We preferred this language to similar ones, such as LUA and PERL, because we believed that compatibility problems would be reduced when working with software from a single vendor. Part of the ASP code is an SQL command that sends a query to the database according to the user's arguments, and the rest of the ASP code is concerned with translating the outcome of the query into HTML format.

The user interface is implemented through the user's own browser. The reasons for choosing this paradigm were: (1) the ease of use by users, (2) the large installed base, and (2) the possibility to make the prototype available to the rest of the community. Using an HTML interface greatly reduces the learning curve of the SET, since most users are already familiar with browser use and navigation. Implementation was simplified not only because all the required software already existed off the shelf, but also because this kind of interface provides templates for page creation. Figure 12 shows the result of a query using the SET. It shows the results of a specific operation that took place during the release of Spec I. Among the details exhibited are the person responsible for applying the operation and her rationale, information described as necessary in Subsection 2.4.

5.1 Example

This subsection shows an example of the use of the SET, so that later we can define more precisely the SET's internal structure and its relationships to the SEF. A typical interaction with the SET is illustrated in Figure 13.

In this interaction, we have just made a query to determine the tracing links from a specified scenario version to other artifacts in the SB. The process, seen from left to right, begins with the first interaction in the browser window. For economy reasons, we assume that the user has already chosen the "Trace" option from the home page and has chosen the option "traceability of any given scenario to other artifacts", as opposed to the option "traceability of a given scenario to other scenarios in the SB". The user is then presented a page titled "Choose a Scenario" that contains every scenario in the SB, and she chooses the scenario that she desires to examine.

The chosen scenario is processed by ASP code in the leftmost ASP page, which assembles a query in which all versions of this particular scenario are selected. The query is then sent to the SB, which in turn sends the result to the same ASP page. This ASP page converts the result, provided in the databases' table format, into HTML format, and the result is forwarded to the browser to be displayed to the user as the middle screen, titled "User Occupies Room". This screen displays the list of every version of the scenario chosen by the user. She chooses a scenario version about which to make a query. In order to locate all traces of the scenario version, it is necessary to make several queries to the SB. This query results in another ASP page, the middle one, and more accesses to the SB. The

result is the contents of the chosen scenario, its direct links to the LEL entries, and a list of links to other artifacts, such as object models and CRC cards. In addition, the history repository of the SB returns the creation data of the chosen scenario, i.e., the releases to which it belongs and the operations with which it is involved. This information is processed by the same ASP code and transformed into HTML format for display to the user, as the rightmost screen of Figure 13. (Figure 9 is the corresponding screen for a version of another scenario.) The screen contains several links, shown as words in gray, which, when pressed, are manipulated by another ASP page, the rightmost. Most of these links either are in the scenario contents and are links to LEL terms or are links to other software artifacts. Heterogeneous artifacts that were not generated by the SET are stored in the RBL. The pages that display them have to be dynamically created by some ASP code. This possibility is represented in Figure 13 by the "Results" arrow that comes out of the rightmost ASP code box in the middle layer. In reality, most queries result in pages with dynamic links that go to other artifacts and scenarios in the SB.

This example should give an idea of the way in which the SET works to provide the views and operations suggested throughout Section 2. We now detail the SET's structure, taking into consideration the components of the SEF discussed previously.

5.2 Structure

Structurally, the SET is divided into two parts. The first part is fixed and implements the objects of the SEF whose configurations do not change. The second part consists of the modules whose configurations can be changed by the user by entering her instantiation arguments to yield a specific instance of the SEF. The fixed part is implemented at the database level, in which the data storage is located and at the software level, in which the data are compiled, the configuration is managed, and the SB is searched. The objects of the SEF that comprise the fixed part are: scenario, version, operation, release, relationship, history, and component. These objects are distributed throughout repositories located in the bottom layer of the SET.

The methods provided by all fixed objects are implemented in ASP in the software level and use information in the data repositories as arguments.

A scenario-based CBS development in the SET is carried out in an instantiation of the SEF. The configurable objects are parameterized at the moment an SEF is instantiated by the user. These objects, as opposed to the fixed objects, use arguments provided by the user during instantiation. The middle, software layer has both the class methods and the code responsible for the instantiation itself, i.e., the database configuration and the code and navigation strategies of each particular instance. The configurable objects are: presentation, trace, and rationale. The presentation object, responsible for the data presentation, is the only object whose implementation permeates all three layers. Even though the navigational strategy is implemented mostly by code located in the middle layer, there are navigational elements also in the other two layers. The database layer has the pointers to other artifacts. These pointers are distributed in the scenario contents and in a special table called trace, located in the SB in the database layer. This table keeps track of the traces among scenarios and other objects. The interface layer has the normal

navigational mechanisms found in browsers, such as the “back” button and the history of previously visited pages.

The way that a rationale object is presented in the SEF is directly related to the operations performed over the scenarios. It is stored in the operation history repository of the SB and keeps information related to other fixed elements of the SEF.

The next subsection describes an example of a session with the SET during which some SEF elements are instantiated.

5.3 Framework Instantiation

The SET allows users to instantiate the SEF through a sequence of pages at the moment the SB for the application is being created. The user defines the parameters for the instance. These parameters indicate the information to be captured in the instance. Figure 14 shows an example of a page in which the user can indicated that he wishes to capture the rationale and to set for which operations the rationales are captured.

Once instantiation is finished, the user may begin to populate the SB. If the user chose to capture the rationale for scenario creation operations, he has to enter the reasons for the creation of each new scenario entered. However, when a scenario results from the *Include* operation, the user has to enter only the scenario contents.

From SEF instantiation on, every modification of the scenarios in the SB will be the result of the application of one or more operations. The user first chooses the operation and the scenarios involved. The second step is to supply the necessary information and modifications for the chosen operation, e.g., in a split, it is necessary to choose names for the new scenarios and to distribute the contents of the original scenario into the new scenarios, rewriting the contents whenever necessary. For a *Modify* operation, the user must change the contents of the scenario in question. For an *Encapsulate* operation, the user chooses the components of the involved scenarios that end up in the resulting scenario. These changes can be requested by the user through the use of a form containing spaces to be filled with the data and rationale for one operation.

Once the SEF is instantiated and the SB is populated, visualization of the evolutionary process is possible along any of the three dimensions, version, release, and tracing. Figures 15 a and b show a sequence of screens in which the user selects a specific release and then can examine every operation that was performed on the scenarios that participate in the release, as suggested should be possible throughout Section 2.

5.4 Underlying Process Model

The SET supports the evolution of scenarios sets by implementing the following process:

- Apply similarity measures [8] to the scenarios of a release *R*.
- Identify relationships held among the scenarios in *R*. The results of this step is illustrated by Figure 2.
- For each identified relationship in *R*, the SET provides a set of suggested operations [8, 7] that can be applied to the subset of *R* related by the relationship.
- For one relationship of a subset of *R*, if more than one operation is possible, the user decides which operation, if

any, is to be applied. The user may choose not to apply any of the suggested operations.

- Apply chosen operations to the scenarios.
- The result is a new release of scenarios, as described in Subsection 2.3 and illustrated by Figure 11.

5.5 Evaluation of the SET

The SET was used to implement the simulation of the Light Control System (LCS) for the Computer Science Building at the University of Kaiserslautern. The full implementation of the LCS is described in [8]. The SET was used to implement also the graduate studies database for the Departamento de Informática at the Pontifícia Universidade Católica in Rio de Janeiro. For both of these implementations, the SET was found to be useful. The SET documented all relationships and permitted all operations identified as essential in Section 2. The full implementation is described in [7].

The process to instantiate the SEF to make the SET for each of these projects was straightforward, but quite frankly, contained too many steps. However, once the instance of the framework was available for use, the inclusion of scenarios and the subsequent evolutionary process was facilitated by the presence of the scenario specification template and the limited choice of operations.

Thus, the SET proved successful as a proof-of-concept prototype for a scenario evolution management tool based on the SEF.

Because some of the users of the SET were not familiar with the use of scenarios and of the operations available, it proved necessary make a help tool for the SET that described heuristics for choosing operations to apply to the current set of scenarios.

In the end, some groups started to use the tool to jointly visualize the scenarios while working collaboratively on them. This was a pleasant surprise because the tool was not designed specifically for collaborative work.

5.5 Summary

This section has presented the prototype of an SET that implements the SEF. The SET allows a user, through a configuration session, to create instances of the SEF that best suit his needs.

6 OTHER WORK

Our work differs from other work both in the concept of scenarios and in the tools for working with scenarios.

6.1 Concepts

The CREWS scenario classification framework [50] offers a perspective on the dynamics of scenario evolution. A scenario may be either a transient or a persistent artifact, depending on its usage in the software development process. Persistent scenarios are created, transformed, and deleted by the execution of operations. The CREWS framework provides five scenario operations, namely, *Refine*, *Expand*, *Delete*, *Integrate*, and *Capture*. The CREWS *Refine*, *Expand*, and *Delete* operations are equivalent to our intrascenario operations of *Modify*, *Include* and *Remove*,

respectively. The CREWS *Integrate* operation relates to the notion that a scenario is a story. As such, several scenarios can be integrated to provide a broader view of functionality. Thus, the CREWS *Integrate* operation corresponds to our *Fuse*, *Encapsulate*, and *Consolidate* operations of the second group of interscenario operations, described in Subsection 2.1. Our approach distinguishes between the operations by the amount of overlapping information shared by the argument scenarios. Our approach thus allows for a richer tracing among the scenarios and requires documenting the reasons for invoking the *Integrate* operation.

Our approach, however, does not provide any operation corresponding to the CREWS scenario capture process. We offer only the intrascenario *Add* operation that allows the addition of a new scenario to the SB. Of course, whenever a scenario is added to the SB, some elicitation is likely to have taken place to get the information in that scenario. The source of this information is requested in the rationale portion of the interface for the *Add* operation, but from where the information came is not made explicit by the choice of the operation itself. One possibility, as suggested by CREWS framework, is to distinguish the operation of adding a brand new scenario from that of adding a previously written scenario, e.g., reused from another project. There would be a new operation added to our taxonomy, *Reuse*, for this latter purpose, thus reserving *Add* for the former purpose.

Antón and Potts [2] did some of the early work on scenario frameworks. Their framework differentiates scenarios according to two criteria: (1) what scenarios are and (2) to what they are applied. Scenario representations can be classified also according to seven dimensions: (1) ontology, (2) emphasis, (3) surface structure, (4) frame of reference, (5) span, (6) level of detail and (7) mood. The scenario component structure is an extension of the scenario classification framework proposed by the CREWS Project [50], in which the authors recognize the dimensions of goals, contents, format, and life cycle with which to classify scenario usage.

Subsection 1.1 notes that “scenarios”, as used in this paper, are closer to the literature’s use cases than to “scenarios” meaning use-case instances. However, there is more to our scenarios than to use cases. The usual use-case approach concentrates on the interface between the macrosystem and the software alone, with little regard to the context and environment in which the CBS will be deployed [29, 5, 52]. Indeed, according to Leffingwell and Widrig [35], the use-case technique can be used to “understand the behavior of the system we are going to develop, as opposed to understanding the behavior of the business the system is going to operate with”. Schneider specifies that use cases are “to be used to describe the outwardly visible requirements of a system” [52]. In our approach, we use scenarios as a means to understand and model the macrosystem as a whole before dealing with CBS interfaces.

Also Kulak and Guiney [33] observe the necessity of identifying dependencies among use cases. Actions such as including a new use case, adding functionality to the CBS, or removing functionality from a use case, can be done only if the dependencies among use cases are identified. They describe a tool called the Content Matrix that summarizes the use cases of a CBS. This matrix includes information that relates use cases to each other and that ranks use cases by priority and importance. This matrix includes also for each use case, information on its pre- and postconditions, assumptions, exceptional and alternative paths of action, triggers, author, and creation date. Their final use case output bears more similarities to our approach to scenario description than to the mainstream

representation for use cases [29, 52, 3]. However, they do not make a distinction among types of dependencies and offer no support to aid identification of the dependencies.

In order to organize use cases, Armour and Miller [3] show how to determine use case dependencies from the use cases' pre- and postconditions. This concept is similar to the determination in our SEF of dependencies based on scenario preconditions and possible precedence relationships. Armour and Miller added also the concept of description in order to capture the information represented in the use-case model. There are three levels of descriptions: initial, base, and elaborated. The levels are used progressively to describe "the activities performed, alternative flows, conditional logic to document exceptions and alternative processing" [3]. Some use cases are organized also into a group to make what is called a business function package that summarizes a responsibility of the system viewed from a business's functional perspective.

Cockburn [14] suggests keeping some additional information with each use case. Besides the use case's authors, release number, and goals, one should keep the use case's business value, complexity, completeness, performance requirements, and external interfaces. In the event that only part of the functionality is delivered in a release of the CBS, it should be noted with each use case, (1) the release in which any part of the use case's functionality shows up and (2) the release in which the full functionality of the use case will be delivered.

The common theme in our and all of these other approaches is to store information above and beyond the basic use case with each use case. This additional information ensures traceability and allows for minimal configuration management.

6.2 Tools

The PrimeCrews software environment addresses scenario tracing [24, 25]. In its current version, the environment automates the generation of the links between usage scenarios and goal models using typed relationships; These types are: *positive*, *negative*, *attains goal*, and *failure*. The PrimeCrews tracing strategy is based on the conceptual model provided by the CREWS predecessor project, NATURE [30]. NATURE aimed at the development and evaluation of Novel Approaches to Theories Underlying Requirements Engineering. It implemented a portion of the IBIS model [15] to capture design decisions made during the requirements engineering process. The original NATURE dependency model provided eighteen different types of dependency links that can be established between traced objects [44]. There is no limitation on the granularity of objects that can be traced; a sentence, argument, or even a word may be specified as a trace object and connected to any other object using a typed dependency link.

IBIS is one of the default rationale techniques that can be instantiated by the proposed SET tool. Full support for IBIS's required data is provided and the records IBIS creates are made available to all stakeholders. We allow IBIS records for other conceptual models, such as object models, entity–relationship diagrams, and business models. Our basic trace object is the scenario. We do not allow trace links between scenario elements and other artifacts, although each link has an optional text box into which users may add comments about the link. We do not support typed tracing links in our SET.

Another tool that provides automated support is proposed by the CREWS-SAVRE team [55, 56, 41]. This tool allows semiautomatic generation of instances from use cases. This tool should be used only during requirements elicitation. However, it allows the identification of inconsistencies and the detection of incompleteness in the requirements as they are compared to the generated instances.

Ian Alexander has built a scenario-management add-on to the Doors tool [1]. It allows creating, editing, and managing use cases (scenarios, in this paper's terminology) that are kept in the same environment as are the requirements. Direct traces from use cases to requirements are thus obtained as an extension of the core functionality of Doors, which includes enabling copy-and-link and linking by attribute, by filtering, and by drag-and-drop. The tool, however, does not provide for internal tracing among the use cases.

CaliberRM™* [6] is a collaborative Requirements Management System. CaliberRM focuses on the capture and maintenance of requirements, which are stored as natural language sentences. Similar to the SET approach, a glossary of terms is used to encourage consistent usage of terms. The tracing strategy used by the tool assumes a requirements perspective, i.e., it links requirements to other software artifacts, such as use cases, files, and tests, but is not concerned whether these artifacts are interrelated.

RequisitePro [16] focuses on requirements and their management. In RequisitePro, a use-case specification consists of mainly textual descriptions, as in the SET. However, the RequisitePro use cases are kept separate from the requirements in the visual modeling tool's storage. It has become clear that it is important to relate use cases to requirements and to manage the use cases along with the requirements. Consequently, RequisitePro explicitly traces from each requirement to the related uses cases. It traces from requirements also to implementation artifacts. In this sense, the evolution of use cases is supported by the tool. It is not clear from the literature whether it is possible to use differences between versions of a use case (scenario in our vocabulary) for analysis. However, since RequisitePro uses the current standard UML representation for use cases [5], and this representation lacks specific expressions of context, resources, goals, and exceptions, it is hard to see how RequisitePro would even able to address all of the issues that the SET can.

7 CONCLUSIONS

The difficulties in managing scenario-based CBS development have been systematically reported in the RE literature [50, 31, 60, 58]. The lack of a method that supports the use and evolution of scenarios has been a keen impediment against integrating scenarios into RE practice.

In an attempt to reduce the problems and to provide automated support for scenario evolution, we embarked on a study of scenario-based CBS development. At first, we focused on deepening our understanding of scenario evolution. We organized several detailed case studies of scenario-based CBS development. The results of the case study led to formulation of an SEM. Based on a set of identified relationships and operations, the SEM explains the variables involved in scenario evolution. It makes a clear distinction between process and product. By restricting users

* CaliberRM is a trade mark of Borland Corporation.

to a limited set of operations, we force some discipline in the evolutionary process, facilitating overall understanding.

The main contribution of this paper are the observations that

1. as requirements evolve over time, so do requirements captured in the form of scenarios, and
2. scenarios are first class requirements artifacts that must be managed as any other requirements artifact, such as a sentence of specification, and as any other software artifact, such as code or a makefile.

It is already well established that the management of evolving artifacts benefit from a software development environment [59] that allows keeping a full history of versions of artifacts and tracing between temporally and semantically related artifacts.

The use of scenarios to support the requirements specification and development of several nontrivial software systems, has shown

1. that scenarios change as much as any other artifact,
2. that it must be possible to trace among related scenarios and among scenarios and other artifacts, and
3. what sorts of scenario manipulation operations are done in the process of deriving requirements and the eventual system from the scenarios.

These relationships and operations have formed the basis of the SEM.

Starting with the SEM and incorporating software management ideas, we propose the SEF as a means to make systematic the use and management of scenarios in the context of CBS development. The SEF is configurable, and by the instantiation of hot spots, it allows its users to define personal strategies for managing scenario evolution. Thus, the SEF is an organization that allows the systematization and evolution of scenarios according to engineering principles, i.e., safely and in a repeatable fashion [28, 22, 46].

The SET is a demonstration prototype whose purpose is to prove the concept that tool support for scenario evolution and management is possible to implement and to demonstrate that the SET functions as envisioned. The SET offers automated support for the SEF. Implemented on top of a Web platform, the SET uses the standard Web browser GUI tools to enhance the visualization of evolution. One of the greatest advantages of scenario-based CBS development is facilitating validation with clients. For this task, the visualization offered by the SET plays a significant role [56, 41, 50].

The most important contribution with regard to tracing of artifacts is the semantics associated to each tracing link. When, for example, scenario *A* is modified to yield scenario *B*, that modification is *O* one of the scenario evolution operations described in Subsection 2.1. As a consequence, in the trace that is maintained in the SB, there is a link from *A* to *B*, and that link indicates that *O* was used to derive *B* from *A*. Additional information associated with the link include a statement of the rationale behind the modification. In this way, the tracing links capture the history of the scenario evolution. The set of operations defining the semantics of links are precisely those identified during the case studies of the use of scenarios to support the requirements specification and development of several nontrivial software systems.

ACKNOWLEDGMENTS

The authors thank the anonymous referees of a previous submission of this paper. Breitman was supported in part by a CNPq (Brasil) Doctoral Studies Scholarship. Leite was supported in part by CNPq (Brasil) and by a "Cientista do Nosso Estado" award from Faperj (Brasil). Berry was supported in part by NSERC (Canada) grant NSERC-RGPIN227055-00.

REFERENCES

[1] Alexander, I. "Capturing Use Cases with DOORS", *Proceedings of the Fifth International Symposium on Requirements Engineering (RE'01)*, Toronto, ON, p. 264, August 2001

[2] Antón, A.; Potts, C. "A Representational Framework for Scenarios of System Use", *Requirements Engineering Journal*, **3**: 3 & 4, pp. 219–241, 1998

[3] Armour, F.; Miller, G. *Advanced Use Case Modeling*, Addison Wesley, Reading, MA, 2001

[4] Bersoff, E.; *et al. Software Configuration Management*, Prentice Hall, 1980

[5] Rumbaugh, J.; Jacobson, I.; Booch, G. *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading, MA, 1999

[6] "CaliberRM, Collaborative requirements management system", Borland Corporation, 2003, `http://www.borland.com/caliber/`

[7] Breitman, K.K. "Evolução de Cenários", Ph.D. Dissertation, Departamento de Informática, PUC-Rio, May 2000

[8] Breitman, K.K.; Leite, J.C.S.P. "Scenario Evolution: A Closer View on Relationships", *Proceedings of the Fourth International Conference on Requirements Engineering (ICRE'00)*, pp. 102–111, 2000

[9] Breitman, K.K.; Leite, J.C.S.P. "Suporte Automatizado A Gerência da Evolução de Cenários", *I Workshop de Engenharia de Requisitos (WER98)*, Maringá, PR, pp. 49–56, 1998

[10] Breitman, K.K.; Leite, J.C.S.P. "A framework for Scenario Evolution", *Proceedings of the Third International Conference on Requirements Engineering (ICRE)*, Colorado Springs, CO, pp. 214–221, 1998

[11] Breitman, K.K.; Leite, J.C.S.P. "Processo de Software Baseado em Cenários", *II (Ibero-American) Workshop on Requirements Engineering*, Buenos Aires, Argentina, pp. 95–105, September 1999

[12] Carroll, J.M. *Scenario Based Design: Envisioning Work and Technology in System Development*, John Wiley and Sons, New York, NY, 1995

[13] Carroll, J.M. *Making Use: Scenario-Based Design of Human-Computer Interactions*, MIT Press, 2000

[14] Cockburn, A. *Writing Effective Use Cases*, Addison Wesley, Reading, MA, 2001

[15] Conklin, J.; Begemann, M. "gIBIS: a Hypertext Tool for Exploratory Policy Discussion", *ACM Transactions on Office Information Systems*, **6**, pp. 303–331, 1988

[16] Connor, C. "Rational Product Integration Focus: Rose and RequisitePro", Rational Software Corporation, `http://www.therationaledge.com/rosearchitect/mag/archives/winter99/f4.html`

[17] Conradi, R.; Westfechtel, B. "Version Models for Software Configuration Management", *ACM Computing Surveys*, **30**: 2, pp. 232–282, 1998

[18] Date, C.J. *An Introduction to Database Systems*, Seventh Edition, Addison Wesley, Reading, MA, 1999

[19] Dömges, R.; Pohl, K. "Adapting Traceability Environments to Project-Specific Needs", *IEEE Software*, **41**: 12, pp. 54–63, December 1998

[20] "Requirements Capture, Documentation, and Validation", Dagstuhl Seminar Report 242, 13.06.99–18.06.99 (99241), Schloss Dagstuhl, June 1999

[21] Filippidou, D. "Designing with Scenarios: a Critical View of Current Research and Practice", *Requirements Engineering Journal*, **3**: 1, pp. 1–22, 1998

[22] Ghezzi, C.; Jazayeri, M.; Mandrioli, D. *Fundamentals of Software Engineering*, Prentice Hall International Editions, 1991

[23] Gotel, O.; Finkelstein, A. "An analysis of the Requirements Traceability Problem ", Imperial College Department of Computing, Technical Report TR-93-41, 1993

[24] Haumer, P.; Pohl, K.; Weidenhaupt, K. "Requirements Elicitation and Validation with Real World Scenes", *Transactions on Software Engineering*, **SE-24**:12, pp. 1036–1055, December 1998

[25] Haumer, P.; Heymans, P.; Jarke, M.; Pohl, K. "Bridging the Gap Between Past and Future in RE: A Scenario Based Approach", *Proceedings of the Fourth IEEE International Symposium on Requirements Engineering (RE99)*, Limerick, Ireland, pp 66–73, 1999

[26] Hicks, D.; Legget, J.; Nurnberg, P.; Schanse, J. "A Hypermedia Version Control Framework", *ACM Transactions on*

Information Systems, **16**: 2, pp. 127–160, April 1998
[27] Hsia, P. *et al.* "Formal Approach to Scenario Analysis", *IEEE Software*, **11**: 2, pp. 33–41, 1994
[28] Humphrey, W. *A Discipline for Software Engineering*, SEI Series in Software Engineering, Addison Wesley, Reading, MA, 1995
[29] Jacobson, I. *et al. Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley/ACM Press, Reading, MA, 1992
[30] Jarke, M.; Pohl, K.; Dömges, R.; Jacobs, S.; Nissen, H.W. "Requirements Information Management: The NATURE Approach", *Ingenerie des Systemes d'Informations*, Special Issue on Requirements Engineering **2**:6, 1994
[31] Jarke, M.; Tung Bui, X.; Carroll, J.M. "Scenario Management: an Interdisciplinary Approach", *Requirements Engineering Journal*, **3**: 3 & 4, pp. 155–173, 1998
[32] Jarke, M.; "Requirements Tracing", *Communications of the ACM*, **41**: 12, pp. 32–36, December 1998
[33] Kulak, D.; Guiney, E. *Use Cases: Requirements in Context*, Addison Wesley/ACM Press, Reading, MA, 2000
[34] Kyng, M. "Creating Contexts for Design *", Scenario-Based Design: Envisioning Work and Technology in System Development*, John Wiley and Sons, New York, NY, pp. 85–107, 1995
[35] Leffingwell, D.; Widrig, D. *Managing Software Requirements, A Unified Approach*, Addison Wesley, Reading, MA, 2000
[36] Leite, J.C.S.P.; A.P.M. Franco. "A Strategy for Conceptual Model Acquisition", *Proceedings of the IEEE International Symposium on Requirements Engineering (RE'93)*, San Diego, CA, pp. 243–246, January 1993
[37] Leite, J.C.S.P.; Oliveira, A.P. "A Client-Oriented Requirements Baseline", *Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE'95)*, York, UK, pp. 108–115, March 1995
[38] Leite, J.C.S.P. *et al.* "Enhancing a Requirements Baseline with Scenarios", *Requirements Engineering Journal*, **2**: 4, pp. 185–198, 1997
[39] Leite, J.C.S.P.; Hadad, G.D.S; Doorn, J.H.; Kaplan, G.N. "A Scenario Construction Process", *Requirements Engineering Journal*, **5**: 1, pp. 38–61, 2000
[40] Leonardi-Maiorana, V.; Balaguer, F. "Una Estrategia de Analisis Orientada a Objetos Baseada em Escenarios", *Anais da II Jornadas de Ingernieria de Software JIS97*, Donostia, San Sebastian, Spain, 1997
[41] Maiden, N.A.M.; Minocha, S.; Manning, K.; Ryan, M. "CREWS-SAVRE: Systematic Scenario Generation and Use", *Proceedings of the International Conference on Requirements Engineering*, IEEE Computer Society Press, pp. 148–155, 1998
[42] McGraw, K.; Harbison, K. *User-Centered Requirements: the Scenario-Based Engineering Process*, Laurence Erlbaum Associates, 1997
[43] Mikkelsen, T.; Pherigo, S. *Practical Software Configuration Management*, Hewlett Packard Professional Books, Prentice Hall, 1997
[44] Pohl, K. "Enabling Requirements Pre-Traceability", *Proceedings of the Second International Conference on Requirements Engineering (ICRE)*, IEEE Computer Society Press, Colorado Springs, CO, pp. 76–85, 14–18 April 1996
[45] Potts, C.; Takahashi, K.; Antón, A. "Inquiry-Based Requirements Analysis", *IEEE Software*, pp. 21–32, March 1994
[46] Pressman, R. *Software Engineering: a Practitioner's Approach*, McGraw Hill, 1992
[47] Ramesh, B.; Stubbs, C.; Edwards, M. "Implementing Requirements Traceability: a Case Study", *Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE95)*, York, UK, pp. 89–95, March 1995
[48] Ramesh, B.; Jarke, M. "Toward Reference Models of Requirements Traceability", *IEEE Transactions on Software Engineering*, **27**: 1, pp. 58–93, 2001
[49] Ridao, M.; Doorn, J.; Leite, J.C.S.P. "Domain Independent Regularities in Scenarios", *Proceedings of the Fifth International Symposium on Requirements Engineering (RE'01)*, Toronto, ON, pp. 120–127, August 2001
[50] Rolland, C.; Achour, B.; Cauvet, C.; Ralyté, J.; Sutcliffe, A.; Maiden, N.; Jarke, M.; Haumer, P.; Pohl, K.; Dubois, E.; Heymans, P. "A Proposal for a Scenario Classification Framework", *Requirements Engineering Journal*, **3**, pp. 23–47, 1998
[51] Rosson, M.B.; Carroll, J. *Narrowing the Specification Implementation Gap in Scenario-Based Design: Envisioning Work and Technology in System Development*, John Wiley and Sons, New York, NY, pp. 247–278, 1995
[52] Schneider, G.; Winters, J. *Applying Use Cases: a Practical Guide*, Addison Wesley, Reading, MA, 1998
[53] `www.stones.les.inf.puc-rio.br/Karin/exemplo/index.html`
[54] Sutcliffe, A. "Requirements Rationales: Integrating Approaches to Requirement Analysis", *Proceedings of the Symposium on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, Ann Arbor, MI, pp. 33–42, 1995
[55] Sutcliffe, A. "Scenario-Based Requirements Analysis", *Requirements Engineering Journal*, **3**, pp. 48–65, 1998
[56] Sutcliffe, A.; Maiden, N.A.M. "Supporting Scenario Based Requirements Engineering", *IEEE Transactions on Software Engineering*, **SE-24**:12, pp. 1072–1088, December 1998
[57] Tichy, W.F. "Design, Implementation, and Evaluation of a Revision Control System", *Proceedings of the Sixth International Conference on Software Engineering*, IEEE, Tokyo, Japan, pp. 58–67, 1982
[58] van Lamsweerde, A.; Darimont, R.; Letier, E. "Managing Conflicts in Goal-Driven Requirements Engineering", *IEEE Transactions of Software Engineering*, **24**: 11, pp. 908–926, November 1998

[59] Wasserman, A.I. “Automated Tools in the Information System Development Environment”. *Automated Tools for Information Systems Design*, H.J. Schneider and A.I. Wasserman (Eds.), Elsevier/North-Holland, Amsterdam, pp. 1–9, 1982

[60] Weidenhaupt, K.; Pohl, K.; Jarke, M.; Haumer, P. “Scenario Usage in System Development: Current Practice”, *IEEE Software*, **15**: 2, pp. 34–45, March 1998

[61] Whirfs-Brock, R.; Wilkerson, B.; Wiener, L. *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, NJ, 1990

[62] Yeh, R.; Ng. P. “Software Requirements: a Management Perspective”, *System and Software Requirements Engineering*, M. Dorfman and R. Thayer (Eds.), IEEE Press, pp. 450–461, 1990

[63] Zorman, L. “Requirements Envisaging through Utilizing Scenarios: REBUS”, Ph.D. Dissertation, University of Southern California, 1995