CREWS Report 97-02

appeared in
'Third International Workshop on Requirements Engineering: Foundation for Software Quality RESFQ',
June 16-17, 1997, Barcelona, Spain

A Software Tool and Method for Scenario Generation and Use

Neil Maiden, Shailey Minocha, Keith Manning and Michele Ryan

Centre for HCI Design
City University
Northampton Square
London EC1V 0HB

Tel: +44-171-477 8984
Fax: +44-171-477 8859
E-Mail: [S.Minocha, N.A.M.Maiden]@city.ac.uk

A Software Tool and Method for Scenario Generation and Use

Neil Maiden, Shailey Minocha, Keith Manning and Michele Ryan

Centre for HCI Design, City University
Northampton Square, London EC1V 0HB
E-Mail: N.A.M.Maiden@city.ac.uk

Abstract

Scenarios, in most situations, are descriptions of required interactions between a desired system and its environment which detail normative system behaviour. There is considerable current interest in the use of scenarios for acquisition, elaboration and validation of system requirements. However, despite this interest, there remains a lack of methods and software tools to generate and use scenarios during the requirements analysis phase. In this paper, we outline the architecture of a toolkit for semi-automatic generation of scenarios. We have derived complex taxonomies of exceptions to help a requirements engineer to predict non-normative system behaviour in a scenario. We have outlined a method of cause-consequence analysis to explore the occurrence of problem exceptions and their effects on system behaviour.

Keywords: Scenario-based Requirements Engineering, Scenario Generation, Requirements Acquisition and Validation, Exceptions.

1. INTRODUCTION

Our studies of current scenario use in requirements engineering indicate that there is lack of systematic guidance for scenario use during requirements acquisition and validation, or even to generate useful and complete scenarios in the first place. It is our belief that more method and tool support for systematic scenario generation and use is needed in the short term to realise the many potential advantages of scenario use during requirements engineering.

The ESPRIT 21903 'CREWS' (Co-operative Requirements Engineering With Scenarios) long-term research project is tackling the scenario problem head-on. First, to help requirements engineers generate a limited set of salient scenarios, it proposes a toolkit for semi-automatic generation of scenarios. Next, it identifies the presence and occurrence of three types of exceptions during scenario analysis to ensure correct and complete requirements. The exceptions are sources of non-normative or exceptional system behaviour as they prevent the system from delivering the required service. One of the toolkit's modules, known as Scenario Wizard, guides the user of the toolkit to systematically explore all the courses in a scenario. The scenario wizard is based on a scenario walkthrough method to ensure systematic exploration of a scenario to resolve incompleteness and inconsistencies in the requirements specifications.

The architectural design and computational mechanisms of the CREWS toolkit build on results from the earlier ESPRIT 6353 'NATURE' basic research action [4]. NATURE identified a large set of problem domain templates or abstractions, or Object System Models (OSMs) which we discuss later on. Each OSM encapsulates the knowledge of normative system-behaviour of all application-domains which are instances of that OSM or problem domain template. A scenario of an application domain which is derived from the NATURE's OSM, thus, describes normative course of behaviour through the scenario. The identification of exceptions and their inclusion in a scenario to explore the non-normative system behaviour, as proposed in CREWS, contributes to the non-normative content of a scenario. Such information-rich scenarios generated from the CREWS toolkit can be useful to guide thinking and discussion during scenario analysis about possible design solutions or mechanisms to eliminate the occurrence of exceptions or mitigate their effects when they occur.

2. THE CREWS APPROACH

The toolkit and walkthrough method support a simple scenario-based requirements engineering paradigm. In its simplest form we envisage the use of scenario alongside a requirements document containing atomic requirements statements stored in a requirements management tool such as Requisite PRO, or as a part of our toolkit. Each scenario enacts or simulates a part of the requirements specifications document. Stakeholders walkthrough the scenario to acquire new system requirements or to validate the completeness and correctness of existing requirements. Desirable changes to requirements statements are identified during the walkthrough. We believe that the more complete the scenario the better the requirements acquisition and validation activities will be. This completeness can come from ensuring that all the basic and alternative courses of scenarios are explored and exploited in a systematic manner.

The wizard, as a part of the toolkit, guides the systematic generation and exploration of a scenario for both its normative and non-normative behaviour. The guiding/advisory mechanism of the wizard is based on the scenario walkthrough method. This method is currently being developed. A part of this method is the cause consequence analysis which helps the requirements engineer to explore the presence and effects of exceptions in a scenario.

Scenario-based requirements engineering is dependent on a sound understanding of the relationship between a scenario and a requirement. We have identified 8 unique dimensions for navigating in the space of the socio-technical system comprising of the machine (software system) and its environment. These dimensions are: Content, scope or coverage or boundary, structure, atomicity, quantification of non-functional requirements, abstraction, design knowledge and view or nature. We are deriving transformation rules or guidelines for generating scenarios from requirements and vice versa through this multi-dimensional navigation. These guidelines would be incorporated in our method of scenario analysis.

3. CLASSIFICATION OF EXCEPTIONS: PREDICTING USEFUL ALTERNATIVE COURSES

Each scenario describes one or more threads of 'normative' behaviour of a software system and consists of agents (human or machine), actions having start events and end events, stative pre- and post-conditions on actions, objects, their states and state transitions and a goal state. It represents a normative usage-situation of a system, which can be a normal sequence of tasks that a user performs to achieve a desired goal. Alternatives or variants to this basic course of events, such as errors that can occur are described as 'alternative courses' [3]. Thus every scenario may have an alternative course, that is, a non-normative state (condition), or, a non-normative event (behaviour) may occur in a scenario. The non-normativeness of a usage context or a scenario implies an inappropriate, or undesirable, or unsafe state or behaviour of a system. Each non-normative state or event, critical or non-critical, is an effect or consequence of an underlying cause or multiple causes existing in the system or in the surrounding environment. Each cause of an inappropriate system performance may be composed of two or more necessary conditions or exceptions.

Exceptions must be explored during requirements analysis as this can help in clarifying and elaborating requirements, and identifying additional or missing requirements for robust design alternatives. The new requirements/constraints that arise to eliminate the exceptions or mitigate their effects on the system performance should be included in the system specifications. We have identified three types of exceptions: generic, permutation and problem exceptions. Each provides the toolkit with a theoretical basis for asking informed 'what-if' questions to prompt the user to explore different alternative courses through a scenario. Generic exceptions are those exceptions that relate to the basic components of a scenario. When different scenarios (permutations of scenarios) are combined or linked to one another, several exceptions can arise in terms of the mappings between the basic components of a scenario, that is, actions, agents, key objects, events, states, etc. These exceptions are termed permutation exceptions. Permutation exceptions can be identified, for example, when one analyses the temporal semantics of two scenarios, that is, comparing the event-action sequence in the two chains in terms of time.

Problem exceptions are, in essence, unexpected events or states which occur in the environment and give rise to alternative courses through a scenario. Examples include mistakes which actors make, failures in mechanical or software systems, breakdowns in communication between actors and unusual situations which arise in organisations. Problem exceptions which are sufficient for the occurrence of an undesired behaviour are said to be the causes. Alternative courses arising from this causes are the consequences of these problem exceptions.

We are undertaking an extensive classification of problem exceptions taken from disciplines as diverse as cognitive science, safety-critical systems, and human factors for system design. Rather than develop a single classification scheme for all problem exceptions we are developing 6 orthogonal classifications, each of which has a particular focus and an existing literature from which it is derived. The first 5 classifications define problem exceptions which are specific to human agents, machine (non-human) agents, interactions between a human agent and a machine agent, communication between two or more machine agents, and communication between two or more human agents. The sixth, final classification describes exceptions which are specific to the organisation, its environment, or models of the organisation and environment. As a first effort, we intend these classifications to cover the whole spectrum of software systems and their environments as a basis for comprehensive coverage by the toolkit and the walkthrough method. The remainder of this section explores these 6 classifications in more detail, and gives examples of useful 'what-if' questions to generate from each of these classifications:

Human Exceptions: People are often responsible for numerous alternative courses through a scenario. We have classified a large number of exceptions which can give rise to such courses from existing taxonomies of human error from cognitive engineering [1,11,12] and human factors research [13]. For example we classify human exceptions as either physiological, anatomical or cognitive. In turn our classification follows Reason's model of human error [12] and classify cognitive exceptions as knowledge-, rule-or skill-based, each of which gives rise to a large number of individual problem exceptions such as mental lapses, attention failures and insufficient knowledge to complete a task;

Machine Exceptions: Software-intensive systems themselves are often responsible for alternative courses through a scenario. Again we have classified problem exceptions which can give rise to such courses [5]. Exceptions can relate to the entire machine (e.g. power failures, hanging machines) or their component parts (e.g. stuck item, loose item, broken item);

Exceptions due to Human-Machine Interaction: Unforeseen human-computer interactions can also lead to numerous alternative courses which are relevant when defining system requirements for error-handling. This time our classification draws on taxonomies of interaction failures from human-computer interaction [9,10] and consequences from poor interface design (e.g. [5]). For example, poor design and non-adherence to design guidelines can give rise to usability problems, poor feedback mechanisms, or inadequate error-recovery mechanisms;

Exceptions due to Human-Human Communication: Scenarios often involve more than one actor or human agent. Communication breakdowns between people have important consequences (for example the London Ambulance Service failure), and are an important source of alternative courses for scenarios. We have classified problem exceptions specific to human-human communication based on theories from computer-supported collaborative working. Examples of exceptions which can give rise to alternative courses include communication breakdowns or misunderstandings;

Exceptions due to Machine-Machine Communication: Scenarios also often involve more than one machine agent, and exceptions specific to their communication can also give rise to alternative courses. Again we are classifying such exceptions as the basis for a useful set of what-if questions.

We are now populating the toolkit with problem exceptions arising from these 6 classifications. Each exception is defined using current object-oriented concepts as either the state of an agent (human or machine), an operation undertaken by an agent (again either a human or machine) or attributes of messages which pass between human or machine agents. We have so far identified over 200 problem exceptions. Furthermore, to make problem exceptions more useful during scenario use, each problem exception has two attributes, one defining the possible likelihood of the exception in different classes of problem domain, and the other defining the severity of consequence of the exception arising in the problem domain. This represents a major advance over current classifications, and will improve systematic generation of useful scenarios. We have also added a third attribute to indicate possible consequences of the exception when the consequence can be predicted. Yet another extension is the inclusion of generic requirements, which are tentative recommendations for requirements which avoid or overcome undesirable problem exceptions. A detailed example of a generic requirement is included in the example of the toolkit's use. Here we present a sample of the taxonomies of problem exceptions, their possible consequences and a proposed set of generic requirements in Table 1.

The requirements engineer can select the relevant exceptions in the generated scenarios to guide the inquiry process of scenario analysis. The exceptions will enable the requirements engineer to ask the 'right' questions from other stakeholders to either predict or investigate any unplanned system behavior. Additionally these taxonomies will serve as checklists for the requirements engineer to guide thinking and stimulate the thought process to uncover 'new' requirements and clarify known requirements. This will help detect incompleteness or ambiguity in requirements. Earlier work [2] concentrated on identifying the obstacles during scenario analysis where obstacles imply as those conditions that can result in the non-achievement of the goal state in a scenario. We, in contrast, have a broader scope for problem exceptions which could be missing data entry validation checks, or non-adherence to user interface guidelines, hardware or software faults, design flaws or errors, hazards, obstacles, critical incidents [5], etc. leading to system failures, mishaps, loss events, accidents, etc.

Category / Sub-Category	Sources of Exceptions	Consequences	Generic Requirements

Human Exception - Physiological	Work Environment - *Noise, lighting, work timings, shift arrangements, temperature, ventilation*	Performing wrong or undesirable actions due to environmental disturbances, improper or no verbal communication with fellow human operators due to noise	Improvement of environmental conditions through better organisational planning
Cognitive	Mental Model of the system - *incorrect mental model, incomplete task knowledge*	Unable to cope with 'emergency' situations, poor or no diagnosis, incorrect decision-making	Explore training requirements, job suitability and replacements if required as per the required skills
Skills	Task knowledge- *lack of expertise, inadequate experience, lack of training or poor quality of training*	Poor or no diagnosis, incorrect decision-making, performing wrong or undesirable actions	Explore role-responsibility allocation structure for task suitable operators as per skills and experience, explore training requirements
Machine Exception - Hardware / Peripheral Equipment	Power supply - *failure or fluctuations*	Whole system or network comes to a halt, malfunctioning of controls and instruments due to power fluctuation	Explore 'avoidance' or recovery mechanisms, like uninterrupted and regulatory power supplies
	Communication - *faulty network connectivity, transmission line failures*	System comes to a halt, faulty or impossible machine to machine communication in a distributed system	Better distributed system planning by procurement of improved networking capabilities

Table 1 A sample set of problem exceptions, consequences and generic requirements

4. CAUSE-CONSEQUENCE ANALYSIS OF PROBLEM EXCEPTIONS OF THE SCEANRIO WALKTHROUGH METHOD

The presence of one exception may give rise to another and so on, triggering a chain of non-normative events leading to an undesirable or inappropriate system performance. We illustrate these causal relations of problem exceptions leading to an unplanned behaviour through example.

Consider the case when an operator of a process control system is not able to control certain parameters optimally (human exception) as s/he does not have the access rights for some functions/information of the machine (equipment), is not trained enough to perform the allocated job, or is working in stressful conditions. These reasons of an operator's inability to perform an allocated responsibility actually reflects on the role allocation structure and planning of the organisation (organisation exception). The human error caused by the cumulative effect of human exception (source) and organisation exception (trigger) may lead to the malfunctioning of the control system (machine exception) and, ultimately, cause a system breakdown.

This aim of this example is not to demonstrate a generic path for causal relations of problem exceptions but it illustrates the occurrence of interaction between the different types of problem exceptions. It also leads to a technique of exploring problem exceptions in a scenario - Cause Consequence Analysis (CCA). CCA starts with a problem exception and determines the consequences that could result from it using a forward search and the causes of the problem exception (if any) using the backward search.

The procedure of CCA during scenario analysis starts with the selection of a problem exception. The requirements engineer explores its effect(s) on system behaviour by exploiting the causal relations to simulate the propagation of the potential effects in the normative task-flow in a scenario. This is the forward search approach of causal analysis which involves identifying or predicting the problem exceptions following a causal path upstream along the flow of events in a task, that is, given the causes, determine the consequences.

Alternatively, if the consequences are known from any previous histories of undesirable system behaviour, the requirements engineer can start from the consequences to determine the cause(s) or problem exceptions by following the causal path of events. This is the backward search approach which involves investigating any previous histories of undesirable performances or failures and identifying the causal factors or problem exceptions, that is, determine the causes from the effects. These techniques of forward and backward searches, as illustrated here, can be integrated in the method of scenario analysis. This approach of CCA is very similar to hazard analysis techniques in safety engineering [5].

5. CREWS TOOLKIT

As a part of the ESPRIT 6353 'NATURE' basic research action [4], a large set of problem domain templates or abstractions, known as Object System Models (OSMs), have been identified to provide domain-specific guidance to requirements engineers. Each model describes the fundamental behaviour, structure, goals, objects, agents, constraints, and functions shared by all instances of one problem domain category in requirements engineering [8]. The 13 top-level OSMs are resource returning, resource supplying, resource usage, item composition, item decomposition, resource allocation, logistics, object sensing, object messaging, agent-object control, domain simulation, workpiece manipulation and object reading. As an example, car rental, video hiring or book lending libraries are applications that belong to the problem domain of resource hiring which is a specialisation of the resource returning OSM.

In CREWS, it is proposed to use the OSMs to provide guidance for scenario-based requirements acquisition and validation, and, in particular, as the basis for automatic generation of the core scenarios. Generation identifies permutations of OSM features to generate a set of possible scenarios. The fundamental components of both OSMs and scenarios are agents, events, objects, states and state transitions. These can be manipulated, as a set, to determine different permutations, or scenarios, for a problem domain. Each individual permutation is called a scenario chain and, is, in essence, a single thread of behaviour in the software system. It is described using agents, events, objects, states, actions and state transitions, all of which are semantics of an OSM. The permutations can be extended using exceptions to define unforeseen situations and events in problem domains. The toolkit takes a compositional approach to scenario generation. It extends each normative scenario using different classes of exceptions. The result is a systematically generated scenario which includes the most suitable classes and levels of problem exceptions, and hence alternative courses. This systematic generation then provides a transparent foundation for systematic scenario use. We illustrate this use in the remainder of the paper. All screen layouts are from the prototype of the toolkit. Detailed treatment of the mechanism of scenario generation and the toolkit's architecture is available in [6].

The example problem domain is the lending library. Assume that three object system models have been retrieved from the NATURE database using its own pattern retrieval mechanisms [7]. These describe resource hiring, resource repairing and object sensing. Resource hiring is an abstraction of book lending, resource repairing is an abstraction of book repair and journal binding, and object sensing is an abstraction of the library's security gates. Because we are interested in book circulation control the user selects the resource hiring model. The toolkit is able to generate 4 initial scenarios from this model: Resource-Loan, Resource-Return, Resource-Reserve and Resource-Unreserve. These are shown to the user who can choose one or more scenario chains to add exceptions and parameters for scenario generation (Figure 1).

Consider the Resource-Loan scenario chain. In natural language it reads 'a start-loan event takes place which starts a resource-loan action. During this action a lender agent changes the state of a resource from available to with-borrower-agent. The borrower agent is involved in the resource-loan action'. Clearly the loan of a book is an example of such a scenario. However it says nothing about the critical boundaries between system and its environment, and in particular what agents are involved, what are human and what are mechanical. For example, is the Resource-Loan action fully manual or automatic or somewhere in between ? This is where the toolkit's agent-interaction patterns come in.

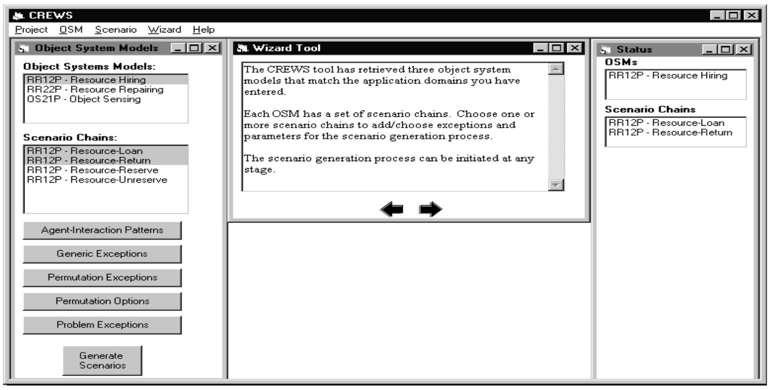

Fig. 1 Retrieval of OSMs and scenario chains

The toolkit offers 15 reusable agent-interaction patterns, each of which describes a canonical pattern of communication actions between 1, 2 or 3 agents, each of which is a human or a machine. The use of these patterns is driven from the observation that the same patterns of communication occur during certain types of transaction, therefore likely agent-interaction patterns can be predicted. For example there are different types of lending systems. These can be fully manual (lender and borrower agents are human) or automated (human lender and borrower agents with one additional machine agent, or human borrower agent with automated lender agent). Basic patterns of communication are reusable across problem domain, and, if linked to a scenario, enable the toolkit to provide more meaningful guidance during a walkthrough. Returning to the example, we shall assume that an automated lending system is desired. Therefore there are two agents. The borrower is a human agent and the lender agent is a machine/computer system. The result is shown in Figure 2. The gives our complete normative scenario for Resource-Loan.

The next stage is to start defining the parameters for alternative courses through the Resource-Loan scenario. The user chooses relevant generic exceptions to be included by selecting them from the list of what-if questions in the toolkit (Figure 3).

Similarly the user can select two or more chains to add permutation exceptions to a combination of scenario chains, that is, permutations of scenario chains. There is a flexibility of adding the permutation exceptions to permutations of same scenario chains or permutations of different scenario chains which have a related and dependent event-action sequence. Figure 4 shows a sample of taxonomies of problem exceptions that the user can choose from to be included in scenarios. The toolkit also enables the requirements engineer to constrain the number and content of generated scenarios through the entry of certain parameters. The scenario content would also depend upon the requirements engineer's choice of what-if questions for the exceptions.

The user can revise the default generation parameters, and finally request for generation of scenarios. The toolkit presents the user with a list of the generated scenarios and the user chooses one or more scenarios to walkthrough. This walkthrough can be modified to suit user needs, for example whether or not to use sequence diagrams, or the format for presenting alternative courses. Figure 5 presents a scenario of Resource-Loan that is generated from the toolkit. The screen is divided into 2 main parts. The left-hand side of the screen shows the simple normative course which is generated for Resource-Loan, containing events, actions, agents, objects and so on. Each of these elements can be clicked on at a time to enable effective walkthrough of the scenario. We envisage that the toolkit will also show a sequence diagram generated from the normative course elements. The user will be able to interact with and edit this sequence diagram directly if desired. The right-hand side of the screen shows possible alternative courses which are generated for Resource-Loan. These courses are as complete as possible (according to parameters set by the user during scenario generation) to ensure complete and systematic use of the scenario. Let us consider the scenario in more detail.

Assume that the user chooses the event element "Borrower Interacts-With Machine". The toolkit displays all alternative courses to consider for that element on the right-hand side of the screen, see Figure 6.

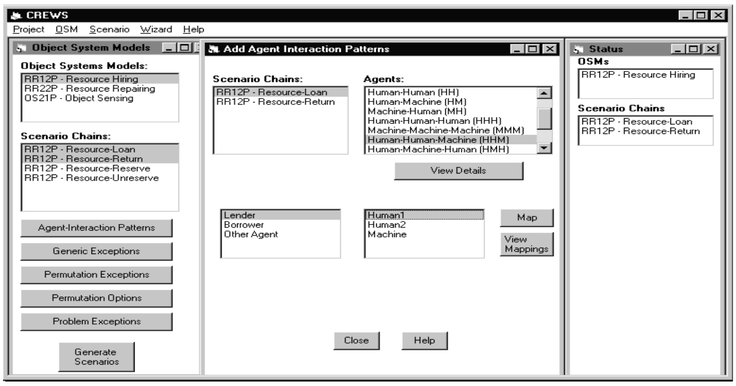

Fig. 2 Choosing agent-interaction patterns

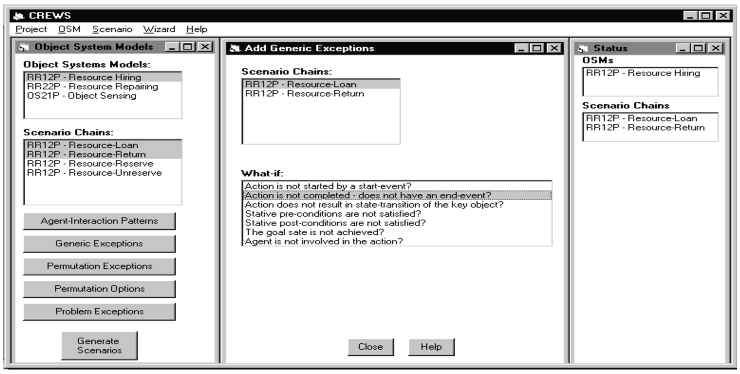

Fig. 3 Choosing generic exceptions

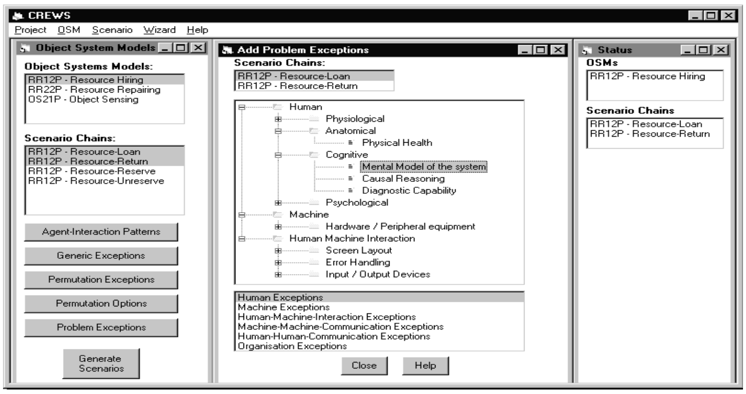

Fig. 4 Choosing problem exceptions

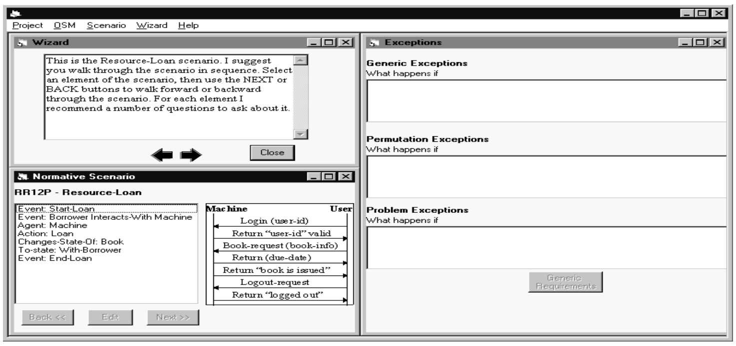

Fig. 5 A generated scenario presented in terms of its components

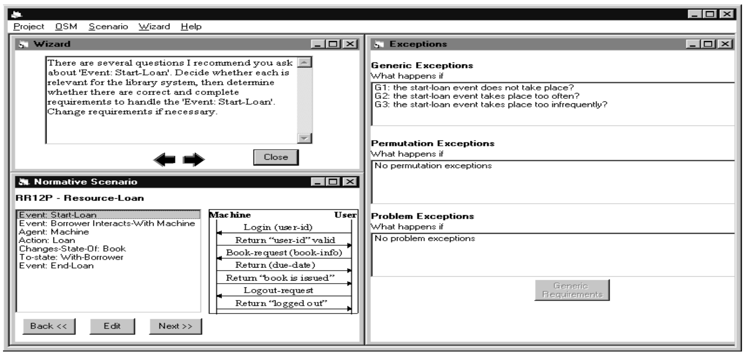

Fig. 6 Exploring a generic exception alternative course

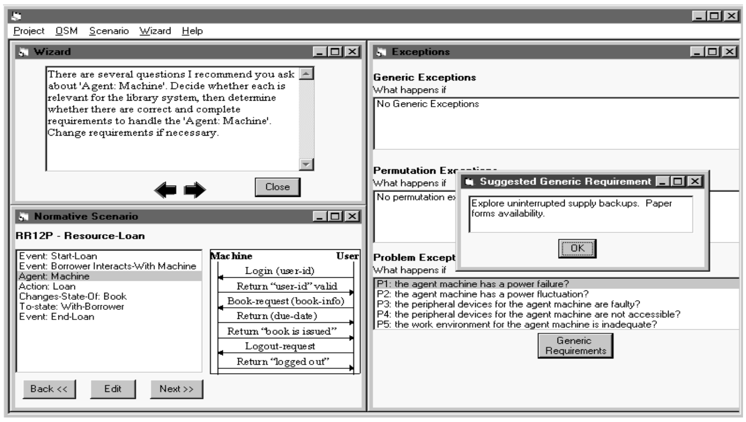

Fig. 7 Exploring a problem exception and candidate generic requirements

The alternative courses are divided into those for generic exceptions, permutation exceptions and problem exceptions. Each is given a unique identifier to make traceability between requirements and scenarios easier to do. The wizard determines that there are 3 generic exceptions to consider for the chosen element. These exceptions encourage the user to ask and explore possible alternative courses about events and actions in the context of the Resource-Loan scenario, see Figure 6.

For example, what happens if the interaction between the borrower and the machine does not complete, for whatever reason, or this interaction does not result in the desired change to state of the book, for whatever reason ? The user uses CCA to determine the likelihood, cause, consequences and impact of each alternative course. The user did not request permutation or problem exceptions for this type of event, therefore none are shown. It is important to remember that the toolkit does not try to predict all alternative courses. Rather it proposes some alternative courses which the user can use as a 'seed' from which to explore other alternative courses using the cause-consequence analysis. Use of cause-consequence analysis broadens the potential usefulness for the toolkit.

Assume that the user walks on through the scenario and highlights the agent 'machine' in the normative course. This time the wizard recommends that the user consider a number of alternative courses about problem exceptions, see Figure 7. Two of these problem exceptions are: the agent-machine has a power failure and the agent-machine has a power fluctuation. The wizard again encourages the user to determine the likelihood, causes, consequences and impacts of each alternative course. However this time, if the alternative course is relevant, the wizard also suggests candidate generic requirements to either avoid the problem arising from the course or overcome the problem entirely. The problem can be avoided by an uninterrupted power supply and overcome by the availability of simple-to-use paper forms system to enable the lender to continue the Resource-Loan action.

6. CONCLUSIONS

The main goal of the CREWS long-term research project is to guide systematic generation and use of scenarios for requirements acquisition and validation. The toolkit and the scenario walkthrough method are being developed to achieve this goal. Our approach to scenario generation and its analysis for both normative as well as non-normative behavior by the walkthrough method will help in determining any missing requirements or possible flaws in design due to incomplete or inconsistent requirements. We have developed a prototype of the toolkit. We are demonstrating the prototype in industrial environments to acquire further requirements for the toolkit, to make it responsive to user needs.

REFERENCES

[1] Hollnagel E. (1993) '*Human Reliability Analysis Context and Control'*, Academic Press.

[2] Hsi I. and Potts C. (1995) 'Towards Integrating Rationalistic and Ecological Design Methods for Interactive Systems', Georgia Institute of Technology, Graphics, Visualisation and Usability Centre *Technical Report*, 1-15.

[3] Jacobson I., Christerson M., Jonsson P., and Overgaard G. (1992) *'Object-Oriented Software Engineering: A Use-Case Driven Approach'*, Addison-Wesley.

[4] Jarke M., Bubenko Y., Rolland C., Sutcliffe A.G. and Vassiliou Y. (1993) 'Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis', *Proceedings 1st IEEE Symposium on Requirements Engineering*, IEEE Computer Society Press, 19-31.

[5] Leveson N.G. (1995) *'Safeware: System Safety and Computers'*, Addison-Wesley Publishing Co.

[6] Maiden N.A.M. (1996) 'Scenario-based requirements acquisition and validation', submitted to *Journal of Automated Software Engineering*.

[7] Maiden N.A.M. and Sutcliffe A.G. (1996) 'Analogical Retrieval in Reuse-Oriented Requirements Engineering', *Software Engineering Journal*, **11**, 281-292.

[8] Maiden N.A.M., Mistry P. and Sutcliffe A.G. (1995) 'How People categorise Requirements for Reuse: a Natural Approach', *Proceedings 2nd IEEE Symposium on Requirements Engineering*, IEEE Computer Society, 148-155.

[9] Nielsen, J. (1993) 'Usability Engineering', Academic Press, New York.

[10] Norman D.A. (1988) *'The Psychology of Everyday Things'*, Basic Books, New York.

[11] Rasmussen J., Pejtersen A.M. and Goodstein L.P. (1994) '*Cognitive Systems Engineering*', John Wiley & Sons, Inc.

[12] Reason J. (1990) *'Human Error'*, Cambridge University Press.

[13] Sutcliffe A.G. and Rugg G. (1994) 'A taxonomy of error types for failure analysis and risk assessment', *Technical Report no. HCID/94/17*, Centre for HCI Design, City University, London.