CREWS Report 97-01

appeared in
'Proceedings of the Workshop on Human Error and Systems Development',
19-22nd March, 1997, Glasgow University, Scotland, United Kingdom, organised by Dr. Chris Johnson

A Co-operative Scenario-based Approach to Acquisition and Validation of System Requirements : How Exceptions can help !

Neil Maiden[1], Shailey Minocha[1], Michele Ryan[1], Keith Hutchings[2] and Keith Manning[1]

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

[2]Philips Research Laboratories
Cross Oak Lane
Redhill
Surrey RH1 5HA

Tel: +44-171-477 8412
Fax: +44-171-477 8859
E-Mail: [N.A.M.Maiden, S.Minocha]@city.ac.uk

Contact Details :

Dr. Shailey Minocha
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@city.ac.uk

A Co-operative Scenario based Approach to Acquisition and Validation of System Requirements : How Exceptions can help !

Neil Maiden[1], Shailey Minocha[1], Michele Ryan[1], Keith Hutchings[2] and Keith Manning[1]

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

[2]Philips Research Laboratories
Cross Oak Lane
Redhill
Surrey RH1 5HA

Tel: +44-171-477 8412
Fax: +44-171-477 8859
E-Mail: [N.A.M.Maiden, S.Minocha]@city.ac.uk

Abstract

Scenario based requirements analysis is an inquiry based collaborative process which enables requirements engineers and other stakeholders to acquire, elaborate and validate system requirements. A scenario, in most situations, describes the normative or expected system behaviour during the interactions between the proposed system and its environment. To account for non-normative or undesired system behaviour, it is vital to predict and explore the existence or occurrence of 'exceptions' in a scenario. Identification of exceptions and inclusion of additional requirements to prevent their occurrence or mitigate their effects yields robust and fault-tolerant design solutions.

In this paper, we outline the architecture of a toolkit for semi-automatic generation of scenarios. The toolkit is co-operative in the sense that it aids a requirements engineer in systematic generation and use of scenarios. The toolkit provides domain knowledge during requirements acquisition and validation of normative system behaviour. It also provides systematic guidance to the requirements engineer to scope the contents of a scenario.

Furthermore, we have identified three kinds of exceptions: generic, permutation and problem exceptions, and have derived complex taxonomies of problem exceptions. We propose to populate the toolkit with lists of meaningful and relevant '*what-if*' questions corresponding to the taxonomies of generic, permutation and problem exceptions. The exceptions can be chosen by the requirements engineer to include them in the generated scenarios to explore the correctness and completeness of requirements. In addition, the taxonomies of problem exceptions can also serve as checklists and help a requirements engineer to predict non-normative system behaviour in a scenario.

Keywords:

Socio-technical system, Co-operative requirements engineering, Scenario based requirements engineering, Scenario generation, Exceptions.

1. Introduction

Scenarios, in our context, are descriptions of required interactions between a desired system[1] and its environment. Scenario-based requirements engineering helps requirements engineers and other stakeholders develop a shared understanding of the system's functionality. Scenarios, derived from a description of the system's and stakeholder's goals, capture the system's expected or normative behaviour. However, to ensure robust and flexible design solutions, it is essential to investigate the occurrence of 'exceptions' in the system and its environment. The exceptions are sources of non-normative or exceptional system behaviour as they prevent the system from delivering the required service.

The ESPRIT 21903 CREWS (Co-operative Requirements Engineering With Scenarios) long-term research project proposes the use of scenarios for both requirements acquisition and validation. Furthermore, it identifies the presence and occurrence of exceptions and emphasises the importance of exploring these exceptions during scenario analysis to ensure correct and complete requirements. First, to help requirements engineers generate a limited set of salient scenarios, this paper describes the architecture of a toolkit for semi-automatic generation of scenarios. Next, we have identified three basic types of exceptions: generic, permutation and problem. The generic and permutation exceptions are the exceptions that arise in the basic event-action-sequence of a scenario or a combination of scenarios. Problem exceptions arise due to the interactions of a software system with its social, operational and organisational environments and provide additional knowledge with which to explore the non-normative behaviour. Furthermore, we have derived complex taxonomies of problem exceptions. We propose to populate the toolkit with lists of '*what-if*' questions corresponding to these taxonomies of generic, permutation and problem exceptions. The requirements engineer can select the relevant exceptions in the toolkit to include them in generated scenarios in order to guide the process of scenario-analysis with other stakeholders. In addition, a requirements engineer can use the taxonomies of problem exceptions as checklists during scenario analysis or in any other technique of requirements analysis.

The architectural design and computational mechanisms of the CREWS toolkit build on results from the earlier ESPRIT 6353 'NATURE' basic research action (Jarke et al. 1993). 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, has normative information content. 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. **Figure 1** represents a model of scenario generation in terms of contributions from NATURE and CREWS. 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.

[1] System, here, is not just a pure software system but is a socio-technical system which has social, technical and organisational dimensions with the user as an integral part of the system.

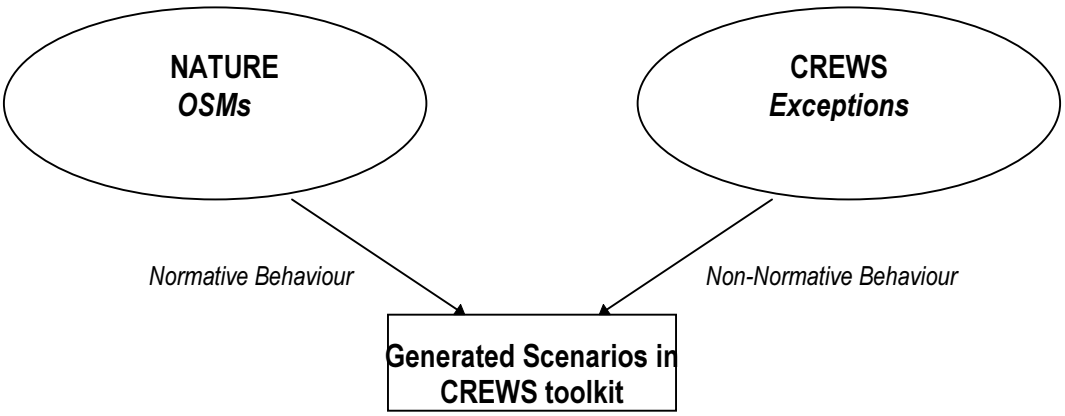

Figure 1 Model of Scenario Generation in CREWS toolkit

First, in Section 2, we discuss several definitions of scenarios and explore their role in requirements engineering. In Section 3, we discuss various types of exceptions which are useful during scenario analysis. We then discuss problem exceptions and present a classification of them. We present a subset of the six taxonomies of problem exceptions that we have derived as answers to 'what can go wrong ?' question across the six dimensions of the classification framework. The toolkit's architecture and the process of scenario generation along with an example are detailed in Section 4. Finally, we present our present directions for future research work in Section 5.

2. Scenarios and Scenario - based Requirements Engineering

Scenarios have been found to be useful in many disciplines. This is interesting to us given the multi-disciplinary nature of requirements engineering. Examples, scenes, narrative descriptions of contexts, mock-ups and prototypes are all different terms for scenarios in the areas of human computer interaction (HCI), requirements engineering and information systems. In HCI, a scenario is often defined as a detailed description of a usage context which helps the designer to explore ideas, consider the appropriateness of design and user support, and other aspects of the environment (Carroll 1995). Scenarios, in the area of information systems, have been defined as partial descriptions of system and environment behaviour arising in restricted situations (Benner et al. 1992). In the context of requirements engineering (Hsia et al. 1994), scenarios have been defined as 'possible ways to use the system to accomplish some function the user desires'. A similar definition is described by Potts et al.(Potts et al. 1994): 'particular cases of how the system is to be used. More specifically, a scenario is a description of one or more end-to-end transactions involving the required system and its environment'. Scenario-analysis helps to evaluate design alternatives, validate designs, notice ambiguities in system requirements, and to uncover missing features or inconsistencies. In object-oriented analysis (Jacobson et al. 1992), scenarios have been defined as 'use-cases', a use-case being a sequence of transactions between an 'actor', who is outside the system, with the system. The use-case approach focuses on the description of the system interactions with its environment.

We can determine the basic characteristics of a scenario from these other disciplines. A scenario is, in essence, a description of required interactions between a system to be built and its environment to achieve some purpose. It can be seen as a behavioural requirement, albeit one which is external to the system to be built, and does not relate to its internal state changes that are unforeseen to the environment. Advantages of such scenarios are numerous. They can be used to anchor communication and negotiation amongst requirements engineers and other stakeholders for acquiring and clarifying requirements. A typical scenario analysis session involves a walkthrough by the requirements engineers and stakeholders to validate the task description or functionality simulated in a scenario.

A scenario captures a 'basic course of events'. 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, and errors that can occur are described as '*alternative courses*' (Jacobson et al. 1992). 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. This will help achieve completeness of requirements as the system specifications would then have the desired system goals as well as the constraints within which the system may operate while achieving these objectives. These constraints would arise from quality considerations (including safety), user interface guidelines, data-input limitations (data-entry validations), and performance considerations (such as system response times).

Despite the extensive use of scenarios in requirements acquisition (Jacobson et al. 1992) and validation (Sutcliffe 1997), it has been reported (Gough et al.1995) that generation of a useful set of scenarios is tedious and time-consuming. Few guidelines exist to aid definition of the structure and contents of a scenario. A requirements engineer must have a considerable understanding and knowledge of the problem domain and the scope of scenario analysis to efficiently generate scenarios. Also, without methodical guidance, it is difficult to detect the non-normative behaviour or the presence and effect of exceptions in a scenario during the inquiry process of scenario analysis.

It is our aim, in this paper, to demonstrate how the toolkit proposed in CREWS uses NATURE's OSMs to provide systematic guidance to the requirements engineer to generate scenarios detailing normative system behaviour. The identification of categories of exceptions and proposed taxonomies of problem exceptions, as a part of research work in CREWS, can further facilitate the requirements engineer to append exceptions to the generated scenarios through the toolkit.

3. Types of Exceptions and Taxonomies of Problem Exceptions

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. We have identified two types of exceptions that can be identified using the basic semantics of a scenario. These are generic and permutation exceptions. The third category of exceptions that we have identified are the problem exceptions. Problem exceptions are those exceptions that arise due to the software system's interaction with the external environment, that is, with the social environment (humans and their interactions), with other software or hardware systems in a distributed environment, or with the organisational environment, including business processes and goals. The identification of problem exceptions gives an integrated exploration of exceptions that can arise in the environments around the software system.

Generic exceptions are those exceptions that relate to the basic components of a scenario simulating a behavioural requirement. A sample set of *what-if* questions listing the generic exceptions to explore the non-normativeness in the event-action sequence of a scenario is: action is not started by a start-event ?; action is not completed, that is, it does not have an end-event ?; action does not result in state-transition of the key object ?; stative pre-conditions are not satisfied ?; stative post-conditions are not satisfied ?; or the goal state is not achieved ?; etc.

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. A representative set of *what-if* questions to guide the identification of such exceptions is: an event that should precede an event happens later ?; two agents perform the same action at the same time ?; the start-events of the actions in two scenario chains are the same and happen at the same time ?; the state-transition of the same key object takes place at the same time involving two different agents ?, etc.

A *problem exception* is a *state* (*condition*) or an *event* that is necessary but not sufficient for the occurrence of an undesired or non-normative behaviour of the system. It can exist within a system or can occur during the execution of the system or it can exist in the environment of the system. It is 'something wrong' in the system (including the operator or 'user') or the environment. It could be a hardware component fault, a design fault in hardware or software, a software fault, an operator (human) or an organisational condition or action, or an undesirable feature of the human-computer interface.

If a problem exception is regarded as being *sufficient* for the occurrence of an undesired behaviour, it is said to be *the cause*. In a complex system, it is difficult to identify a single cause (or a problem exception) for its malfunctioning or unplanned behaviour. This is because the existence of one or more problem exceptions give rise

to the occurrence of another set of problem exceptions by propagation and chain reaction through the system interfaces generating an undesired state of the system.

Figure 2 illustrates the chain of events leading to an unplanned behaviour of a system. A problem exception can act as a *source* when it is an initiating state or condition for the system's deviation from intended behaviour. It can also be a *trigger* when its occurrence activates another problem exception (the source) or a set of problem exceptions. A combination of the source(s) and trigger(s) may cause the system's undesired behaviour or it may give rise to another set of problem exceptions. The *effects* or consequences of the propagation of such a *causal chain* of *source - trigger - problem exception(s) - non-normative system function(s)* would be the system's inability to deliver its required service.

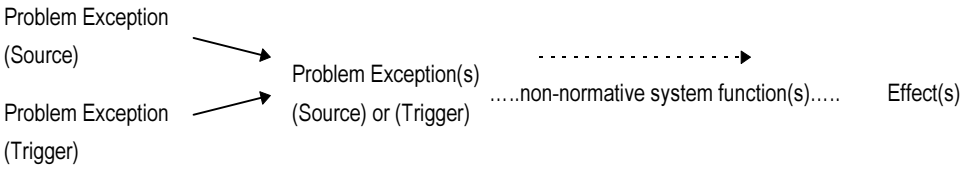

Figure 2 Conceptual Model of Non-Normative System Behaviour

Earlier work (Hsi and Potts 1995) 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[2], obstacles, critical incidents, etc. leading to system failures, mishaps, loss events, accidents, etc. We now propose a classification of problem exceptions. The classification is general in the sense that, while populating it, we plan to cover the whole spectrum of software systems and their environments - from safety critical systems such as avionics and nuclear power plants, or process control environments, to non-safety critical office systems. This is an interesting research challenge we have undertaken, and success is by no means guaranteed.

Classification Framework of Problem Exceptions

There are several taxonomies of problem exceptions in the areas of cognitive science and engineering, safety engineering, and usability engineering in the literature. It is difficult to categorise their domains over orthogonal axes as some of the studies overlap, especially, taxonomies of human error and human mental models (Rasmussen and Vicente 1989), (Norman 1988), (Reason 1990), (Hollnagel 1993). We are making an attempt to bring all these diverse themes of work together in a single classification framework. Secondly, our aim is to apply the derived taxonomies of problem exceptions under this general classification scheme to the area of requirements engineering through scenarios.

Any scenario, which represents a sequence of actions, can involve human agent(s) (H) and machine agent(s) (M). A human agent may also communicate with another human agent and a machine may pass information to another machine in a distributed system.

[2] We will follow the terminology of Leveson's book (Leveson 1995, Chapter 9) for all safety terms such as error, failure, risk, hazard, reliability, etc.

The minimum set of interactions between H and M are: H, M, HM, HH and MM. Each interaction pattern in this set can give rise to an inappropriate or undesired system performance. Based on this set of possible interactions, *we identify five types of exceptions* (**Figure 3**): Human exceptions, Machine exceptions, exceptions that arise due to Human-Machine Interaction, exceptions that arise due to Machine-Machine Communication or exceptions that arise due to Human-Human Communication. Apart from these five categories, we have identified a sixth type - *Organisation exceptions*, that is, exceptions that arise due to organisational structure or social conditions.

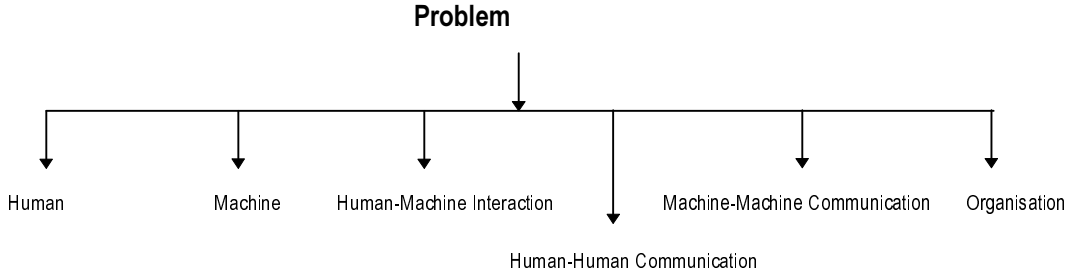

Figure 3 Classification Scheme of Problem Exceptions

A human agent, as an integral component in the social environment of the software system, takes decisions, performs actions, etc. During this interaction, a deviation of normal behaviour of a human agent may result in an inappropriate system performance. The causes of deviations in the human agent's actions or behaviour are called human exceptions. These exceptions have been termed as the causes of *human errors* in cognitive engineering (Reason 1987), (Reason 1990), (Norman 1988) and human factors engineering (Sutcliffe and Rugg 1994). For example, human exceptions may arise due to insufficient knowledge, memory lapses, incorrect mental model, etc. When the failure of a machine due to conditions such as power failures, or hanging, or getting disconnected from the network, etc. give rise to an inappropriate system performance, then such technical causes are called as *machine exceptions* or technical exceptions. These are the sometimes considered as causes of *technical failures* in the literature (Leveson 1995). Design errors, or non-adherence to user interface guidelines, etc. may give rise to situations when a human agent is unable to take a decision or diagnose a problem while interacting with a machine. This results in an undesired system performance. Such exceptions are termed as arising due to *human-machine interaction* mismatches (Nielsen 1993), (Rasmussen et al. 1994), (Hollnagel and Kirwan 1996). These may arise due to usability problems, poor feedback mechanisms, or inadequate error-recovery mechanisms. When a human agent is not able to communicate appropriately or as desired to another human agent, either through speech, documents, or any other media and which may cause an unacceptable system performance, then such causal factors are called *human-human communication* exceptions (Reason 1990). Exceptions due to human-human communication may arise due to communication mismatch between peers as a result of unclear task allocation or lack of co-ordination at management level. The importance of verbal communication between controllers in the London Ambulance Service's control room was overlooked in the infamous computer-aided despatch system disaster, and was one reason for the system's ultimate failure (Dowell and Finkelstein 1996). A situation may arise when a machine is unable to communicate correctly to another machine in a network or

distributed system. This may be caused due to an exception of the type - *machine-machine communication.*

Causal Relations of Problem Exceptions

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 this causal relations of problem exceptions leading to an unplanned behaviour through an example.

Consider the case when an operator of a process control system is not able to optimally control certain parameters (human exception) as s/he does not have the access rights for some functions/information of the machine (equipment), or 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. In this case, the sequence of events can be represented as:

Human Exception (source) $\oplus$ Organisation Exception (trigger) $\Rightarrow$ Human Error $\Rightarrow$ Machine Exception $\Rightarrow$ 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. The example has two important messages: First, if a problem exception is known, the requirements engineer can explore its effect(s) on system behaviour by exploiting these causal relations to simulate the causal chain of events in the normative task-flow in a scenario. For example, when power failure is identified as a problem exception, the requirements engineer will explore all that can be caused due to a power failure in the scenario of the application domain in question. Following this causal explanation and determining the possible effects of power failure and the severity of these effects, the requirements engineer will add requirements to the scenario description to eliminate the occurrence or to diminish the effect of power failure on the system's environment. In a safety-critical system, in order to avoid a power failure, an additional and significant requirement would be to include the availability of uninterrupted power supply machines in the system specifications. However, in the context of a book-lending library, this problem exception may be tackled by including the availability of paper-forms or other mechanisms where the library staff members can manually issue books, fine borrowers, or borrowers can reserve books, etc.

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. These techniques of forward and backward searches, as illustrated here, can be integrated in the method of scenario analysis. This approach of causal analysis is very similar to hazard analysis techniques such as HAZOP in safety engineering (Leveson 1995). We are currently developing a method to incorporate causal analysis within scenario based requirements engineering.

Taxonomies - Populating the Classification framework

We have derived a set of taxonomies as answers to 'what can go wrong' question along the six dimensions of the classification framework. We have considered the taxonomies available in the literature while populating our classification scheme of problem exceptions. We present a sample of these taxonomies in **Table 1**. A more detailed and complete set of taxonomies is beyond the scope of this paper. To supplement and validate the derived taxonomies, we are conducting field studies through knowledge elicitation techniques (Maiden and Rugg 1996) to gather information from experienced requirements engineers, system designers, and also end-users. The taxonomies are proposed to be populated in the toolkit. 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 behaviour. 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.

Exception Type	Category	Sources of Exceptions
Human	Physiological	Work Environment - *Noise, lighting, work timings, shift arrangements, temperature, ventilation*
		Stress - *Reactions to stress*
		Attention capacity - *over attention or inattention, perceptual confusion*
		Adaptation - *reaction to changes in system and environment*
		Mental Load - *tired, stressful*
	Anatomical	Physical Health - *disability, sick or injured, poor physical co-ordination, fatigue*
	Cognitive	Mental Model of the system - *incorrect mental model, incomplete task knowledge*
		Causal Reasoning - *delayed feedback from the system, perceptive power for the consequences*
		Diagnostic Capability - *depending on diagnostic support from the system, task knowledge*
	Psychological	Morale - *management policies and attitudes*
		Motivation - *boredom due to repetitive tasks*
		Disturbance - *environmental distractions due to noise, lighting, work place set-up, etc.*
Machine	Hardware / Peripheral equipment	Power supply - *failure or fluctuations*
		Peripheral devices / instruments - *faulty or inaccessible*
		Communication - *faulty network connectivity, transmission line failures*
		Work environment - *inadequate or non-availability of support staff*
Human Machine Interaction	Screen layout	Widget layout - *improper choice or unsuitable icons, user interface controls, user interface cues or metaphors for the task and user*
		Information Presentation - *over load and poor spread of information, inconsistent, wrong choice of colours, colour combinations and fonts, non-conformance to human factor guidelines, data entry validations, improper dialogue design and navigational flow, slow system response times in information retrieval, unsuitable decision aids for the task and user*
		Design guidelines - *non-conformance to platform-dependent GUI guidelines, internationalisation requirements if applicable ?*
	Error Handling	Feedback - *non-indicative warning messages, alerting techniques such as alarms, flashing and reverse video, delayed response times*
		Error recovery mechanisms - *absent or slow*
	Input / Output devices	Keyboards, pointing devices, sound, monitors, display panels, indicators, etc. - *faulty or unsuitable*

Table 1 A Sample Set of Problem Exceptions

4. CREWS Toolkit

As a part of the ESPRIT 6353 'NATURE' basic research action (Jarke et al. 1993), 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. These models are similar to analysis patterns (Coad et al.1995), problem frames (Jackson 1995), or clichés (Reubenstein and Waters 1991). However, NATURE has produced the first extensive categorisation of requirements engineering problem domains from domain analysis, case studies, software engineering books, etc., in the form of over 200 OSMs with 13 top-level OSMs held in a hierarchical object oriented deductive database. 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. The proposed OSMs in NATURE have been validated through empirical studies (e.g. Maiden, et al. 1995), and tools have been constructed to use them in requirements structuring (Maiden and Sutcliffe 1993), critiquing (Maiden and Sutcliffe 1994) and communication, and requirements reuse (Maiden and Sutcliffe 1992).

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 or initial 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 (Potts et al. 1994). 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. Furthermore, features in OSMs are interconnected, thus enabling the imposition of useful constraints on scenario generation. A computational mechanism to generate these permutations has been designed to generate scenarios in this manner. It is being implemented in the CREWS toolkit along with the facility to append exceptions to the scenarios.

We are currently developing a throw-away prototype of the CREWS toolkit. We plan to conduct tests in the industry using a scenario-based requirements elicitation technique (Sutcliffe 1997) to determine additional requirements for the toolkit in order to scope the structure, content and forms of presentation. In addition, we will be conducting usability studies for the look-and-feel of the user interface and navigational flow of the toolkit. The prototype is currently being reviewed internally in our centre along with its iterative development. We aim to submit our results of our internal reviews and ‘real user’ testing of the prototype in the near future.

We now illustrate the process of scenario generation using the NATURE’s OSMs and present the screen layouts of the prototype of toolkit to demonstrate it. Detailed treatment of the mechanism of scenario generation and the toolkit’s architecture is available in (Maiden 96). In this paper, we focus on the facility of adding exceptions to the scenarios through the toolkit to analyse the non-normative system behaviour, both in terms of its event-action analysis within a scenario as well as it’s interactions with the external environment. The screen layouts in **Figures 4 - 11** are from the prototype of the toolkit.

Example

Consider the example of an application domain as a book-lending library. In the toolkit, first, the application domain facts are captured through an interactive dialogue with the requirements engineer (the 'user of our toolkit'). The requirements engineer enters the details of agents, events, actions, etc. through a dialogue to provide domain-specific information. These inputs are matched with the stored OSMs in the database to retrieve them. The retrieval yields three OSMs: Resource Hiring, Resource Repairing and Object Sensing. Each OSM has scenario chains in the database. The requirements engineer selects the OSM of resource hiring. There are four core or initial scenarios in this application domain: Resource-Loan, Resource-Return, Resource-Reserve and Resource-Unreserve which are retrieved and are shown on the display when the user selects an OSM (**Figure 4**). The requirements engineer has the option to choose one or more scenario chains to add exceptions, parameters for scenario generation such as constraining the number of scenarios to be generated, etc., and the agent interaction patterns. The agent-interaction patterns map the agents, machine or human, to the agents in the scenario chain. Agent types and patterns of interaction are critical for scenario generation. Object system models include abstractions of agents but say little about them because agent types and interactions are not facts which discriminate between categories of problem domain.

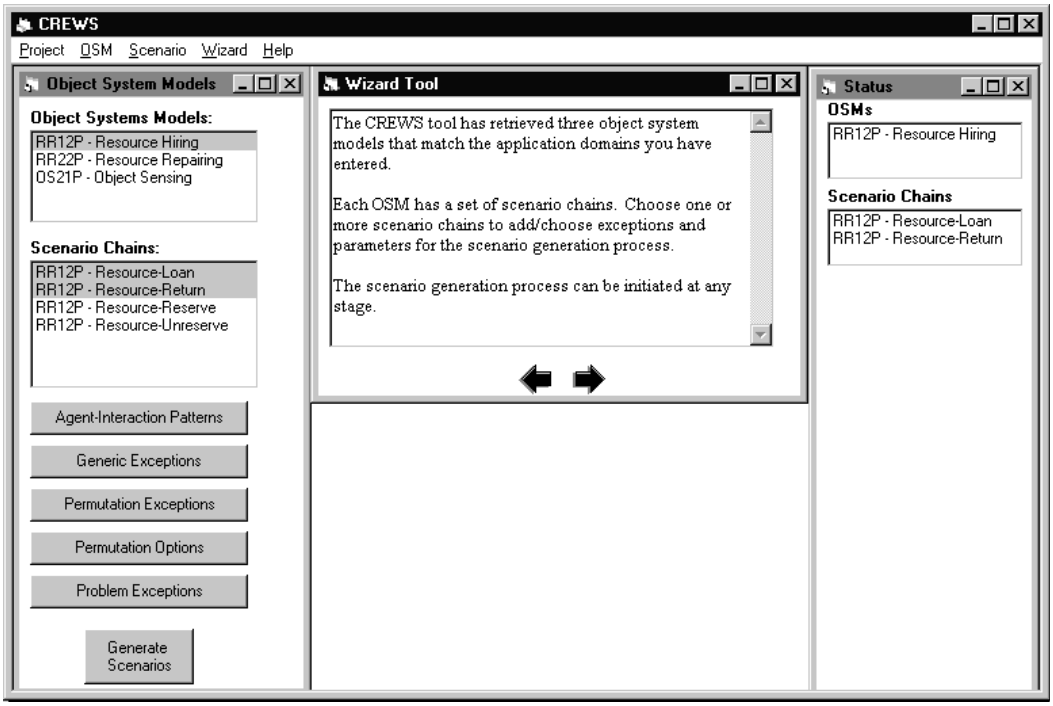

Figure 4 Retrieval of OSMs and Scenario Chains

Let us consider the initial scenario chain: Resource - loan. In natural language this initial scenario reads 'lender lends a resource to the borrower'. An example: in the library domain, the resource is a book and a borrower requests for the issuing of a book from the lender who is a library staff member. (Here, we are considering an example of a library where the lender brings the book to the library desk and the library staff member interacts with the computer system (machine) to issue the book to the lender). On selecting a scenario chain, the requirements engineer has the flexibility to choose either the generic exceptions, or the permutation exceptions, or to choose

the problem exceptions to include the exceptions to a scenario chain or combination of scenario chains, or to enter the parameters for scenario generation, or to map the agent interaction patterns.

If the requirements engineer chooses the option to map the agent interaction patterns. Lender, Borrower and Other Agents are the agents in the initial scenario chain. They would be mapped to human and machine agents as follows (**Figure 5**):

Borrower: Human agent (in real-world, a student or staff member in a university library);

Lender: Human agent (librarian);

Other Agent: Machine agent (Computer system).

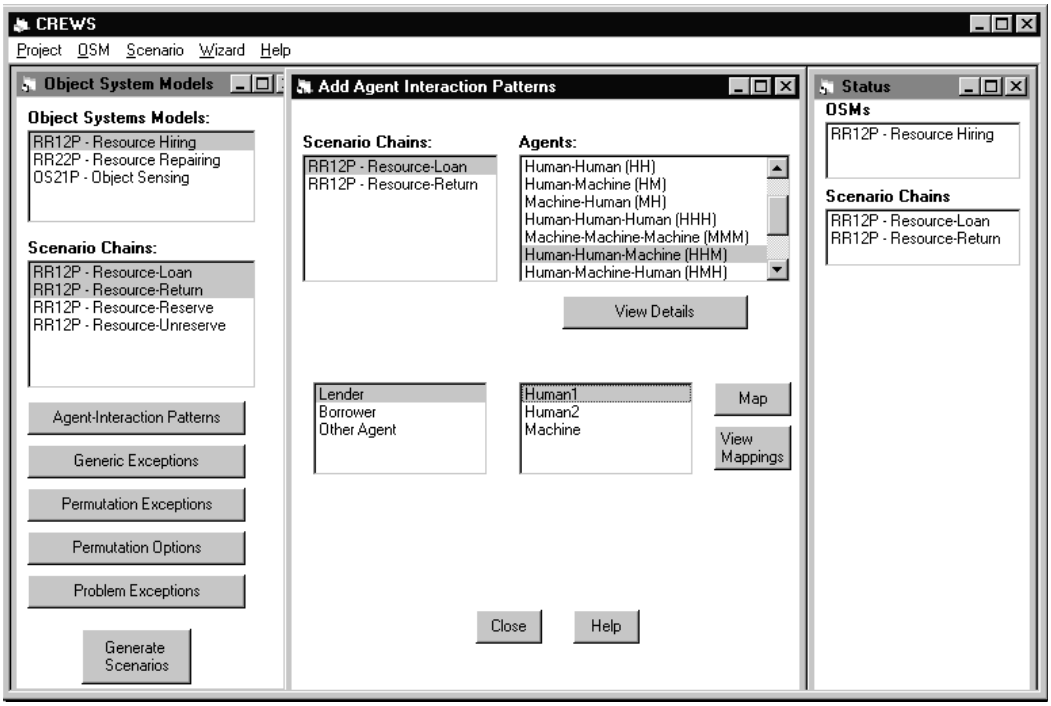

Figure 5 Choosing Agent Interaction Patterns

Next, say, the requirements engineer chooses to add generic exceptions by selecting the from the list of *what-if* questions in the toolkit (**Figure 6**).

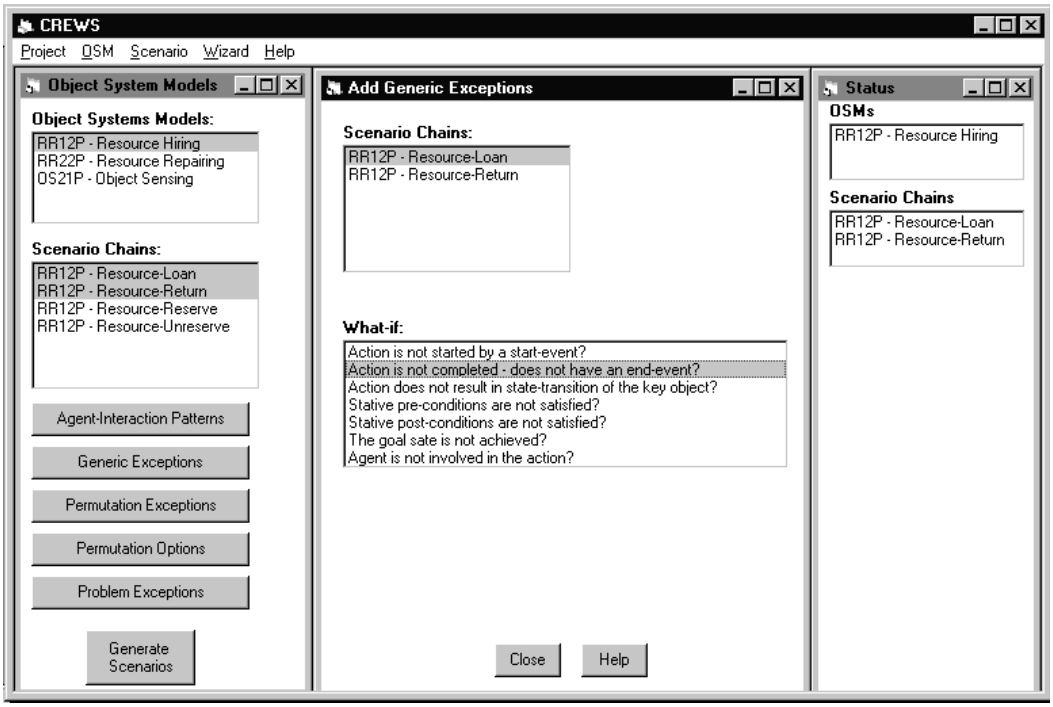

Figure 6 Choosing Generic Exceptions

The requirements engineer can select two or more chains to add permutation exceptions to a combination of scenario chains, that is, permutations of scenario chains (**Figure 7**). 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.

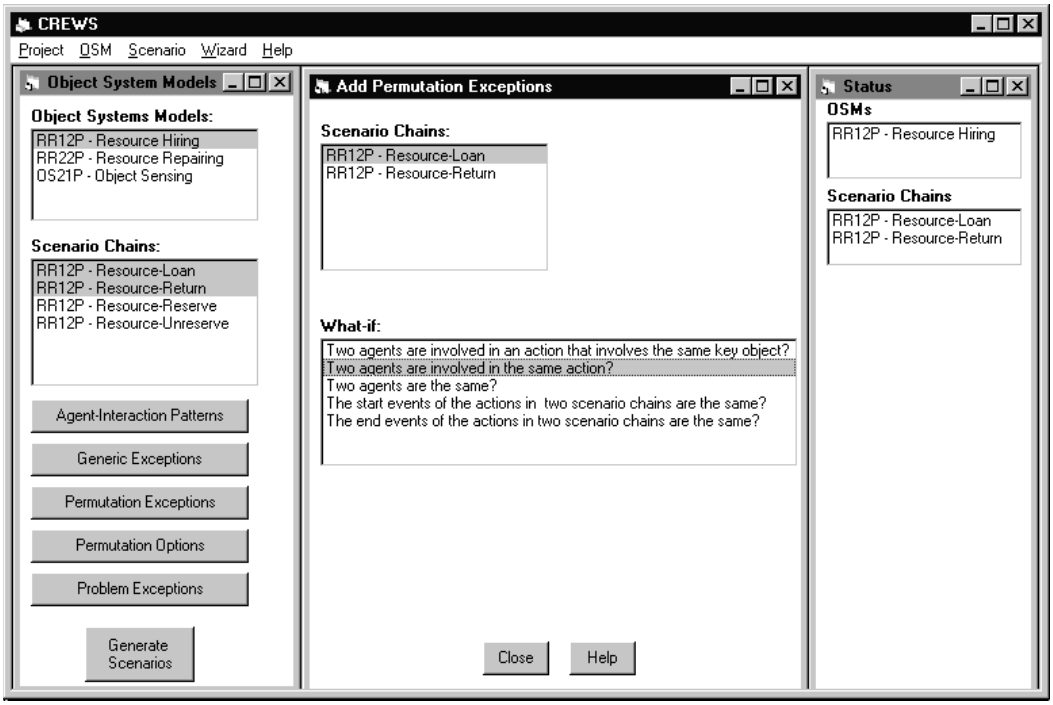

Figure 7 Choosing Permutation Exceptions

Figure 8 shows the sample of taxonomies of problem exceptions that the requirements engineer can choose from to include them in the scenarios.

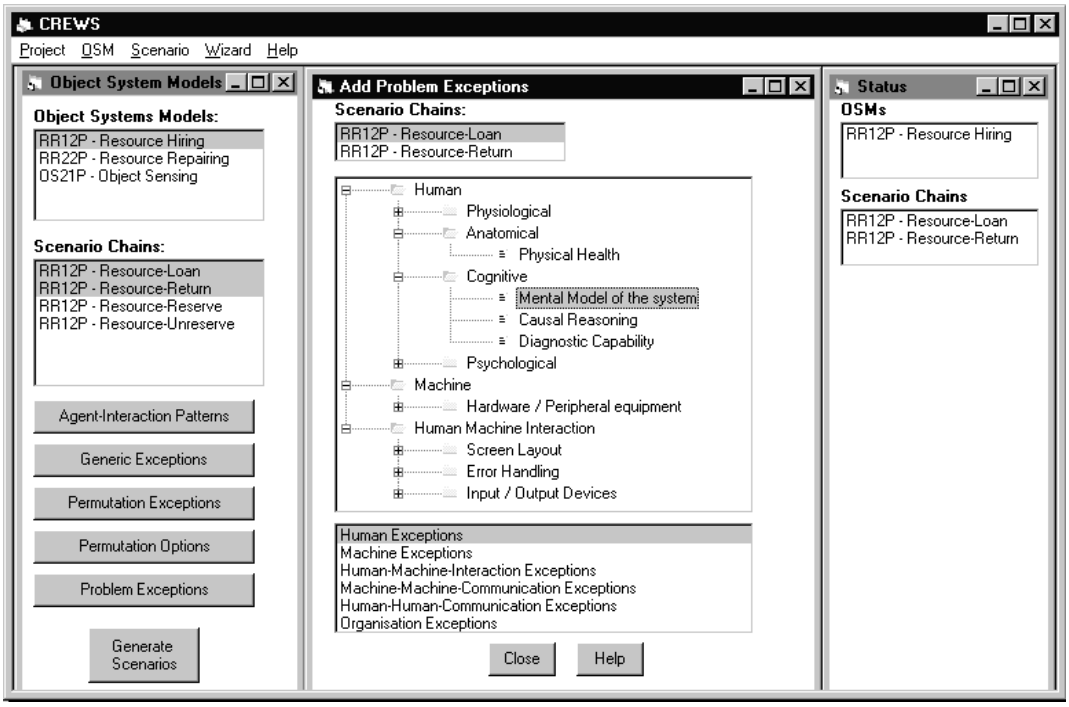

Figure 8 Choosing Problem Exceptions

The number and content of the generated scenarios would be constrained by the parameters entered by the requirements engineer **(Figure 9)**. The content would also depend upon the requirements engineer's choice of what-if questions for the exceptions. The scenarios are generated from the toolkit after the requirements engineer's initiates the generation process and are presented in the form of a list (**Figure 10**). The toolkit will provide an option to the user to view any individual scenario as a sequence diagram or as a structured description in terms of its constituents: basic semantics and exceptions appended to it.

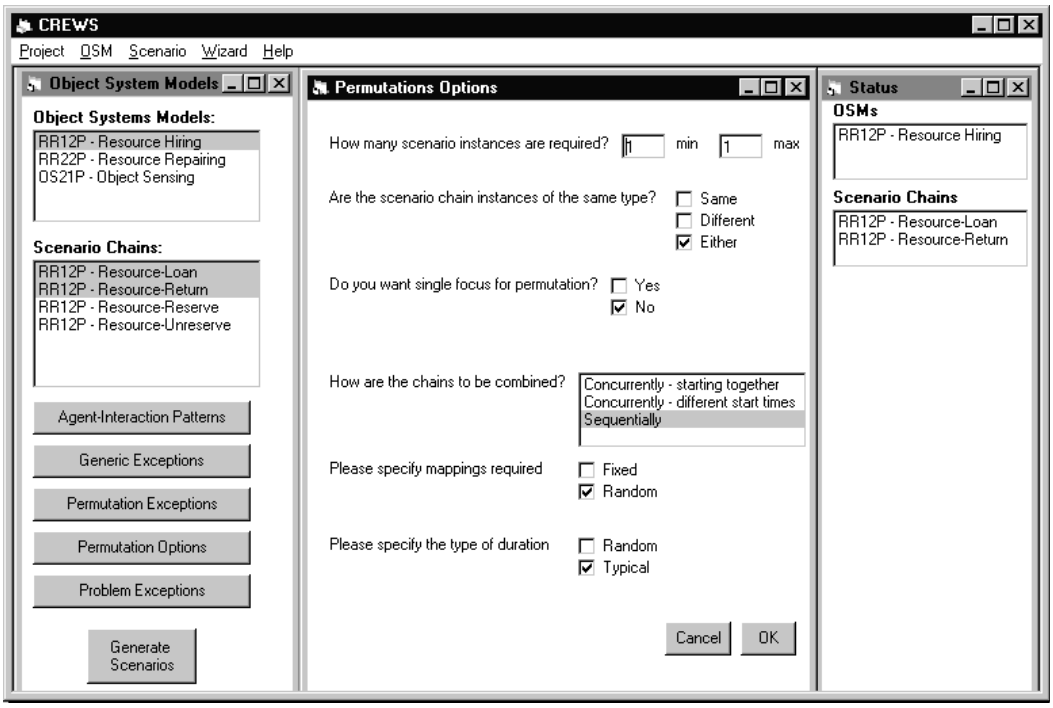

Figure 9 Choosing Permutation Options for the Scenario Generation Mechanism

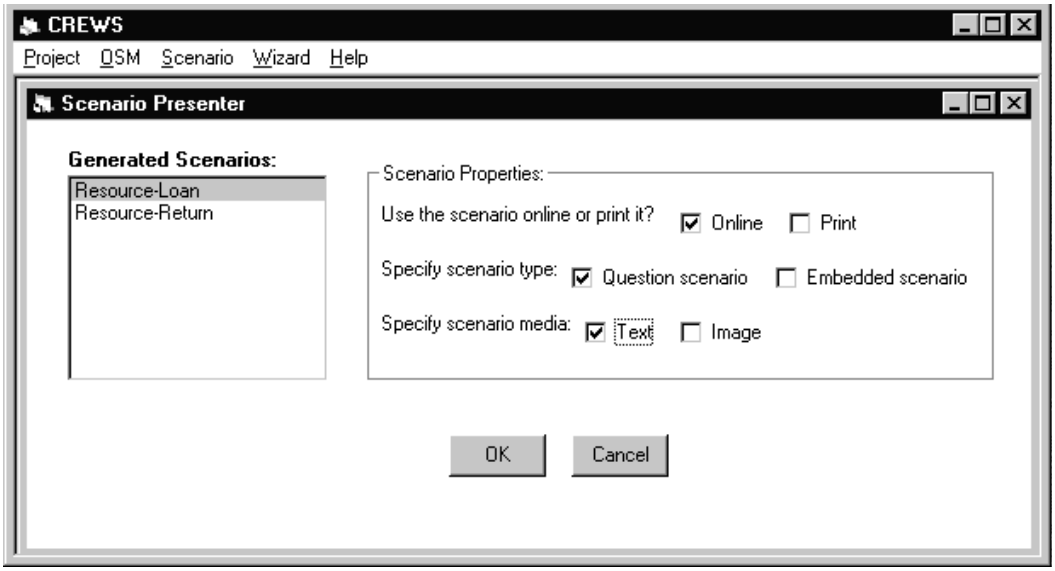

Figure 10 Generated Scenarios

5. Future Research Work

One of our immediate research goals is to identify taxonomies of problem exceptions for application domains which are instances of the 13 top-level OSMs from the general classification framework. We intend performing empirical studies and using knowledge elicitation techniques (Maiden and Rugg 1996) to assemble and validate the taxonomies in such application domains. We also propose to suggest corresponding generic requirements or guidelines to the requirements engineer for these derived taxonomies. The toolkit would then be populated with these application-domain-specific problem exceptions and their generic requirements. For example, a generic requirement to cater for an exception due to human machine interaction could be 'Design for error tolerance'. This means: (a) make errors observable, (b) provide error-recovery mechanisms. A requirements engineer would map these generic requirements into actual requirements in a scenario for an effective human machine interaction: (a) provide feedback by alarms and give warning displays, (b) provide reverse (compensating) actions. The generic requirements corresponding to application-domain problem exceptions can, thus, aid the requirements engineer to identify new and complete requirements.

We also plan to propose some taxonomies by populating the classification framework across several other dimensions such as cause - effect (consequence), or severity-likelihood, or failure-mode - effect analyses of problem exceptions. This may involve including some of the taxonomies existing in the literature: requirements engineering (Fields et al. 1995): logic errors in the software due to incorrect requirements; safety engineering (Hollnagel 1993), (Leveson 1995): studies in safety-critical systems; hazard analysis; accident analysis, task and human error analysis, etc.; usability and human factors engineering (Nielsen 1993) and cognitive engineering (Norman 1988), (Rasmussen and Vicente 1989), (Reason 1990), (Rasmussen et al. 1994): ecological interface design, models of human error, task models, human-task mismatches, diagnostic support, decision-support and identification of decision requirements, etc.

In addition, we are currently developing a method for scenario analysis which would complement the traditional task analysis techniques. This method would involve causal analysis (Rasmussen 1991), (Rasmussen et al. 1994), as a part of scenario analysis, to explore the occurrence of problem exceptions. It would involve forward and/or backward search methods of causal analysis. The forward search approach would involve 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. The backward search approach would involve 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. To further systematise and elaborate the method, we are also looking into other accident or hazard analysis techniques: Fault Tree Analysis used in aerospace, electronics and nuclear industries (Leveson 1995), HAZOP analysis (Leveson 1995) which is a hazard analysis technique used in the chemical process industry, and the automation of HAZOP and its application to software requirements specification through Deviation Analysis (Reese 1995). A comprehensive approach to scenario analysis guided by our proposed method will help in determining any missing requirements or possible flaws

in design due to incomplete requirements which can contribute to the likelihood of an inappropriate system performance.

References

Benner, K.M. Feather S., Johnson W.L. and Zorman, L.A. (1992) 'Utilising Scenarios in the Software Development Process', *IFIP WG 8.1 Working Conference on Information Systems Development Process*, 117-134.

Carroll J.M. (1995) 'The scenario perspective on System Development', in *Scenario-based Design: Envisioning work and Technology in System Development*, **Ed. J. M. Carroll**.

Carroll J.M., Mack R.L., Robertson S.P. and Rosson M.B. (1994) 'Binding objects to scenarios of use', *International Journal of Human-Computer Studies*, **41**, 243-276.

Coad P., North D. and Mayfield M. (1995) '*Object Models: Strategies, Patterns and Applications*', Englewood Cliffs, Prentice Hall.

Dowell J. and Finkelstein A.C.W. (1996) 'A Comedy of Errors: the London Ambulance Case Study', *Proceedings 8th International Workshop on Software Specification and Design*, IEEE Computer Society Press, 2-4.

Fields, R.E., Wright, P.C., and Harrison, M.D. (1995) 'A Task Centred Approach to Analysing Human Error Tolerance Requirements', *Proceedings 2nd IEEE Symposium on Requirements Engineering*, IEEE Computer Society Press, 18-26.

Gough P.A., Fodemski F.T., Higgins S.A. and Ray S.J. (1995) 'Scenarios - an Industrial Case Study and Hypermedia Enhancements', *Proceedings 2nd IEEE Symposium on Requirements Engineering*, IEEE Computer Society Press, 10-17.

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

Hollnagel E. and Kirwan B. (1996) 'Practical Insights from Studies of Operator Diagnosis', *Proceedings 8th European Conference on Cognitive Ergonomics*, EACE, 133-137.

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.

Hsia P., Samuel J., Gao J., Kung, D., Toyoshima, Y. and Chen, C. (1994) 'Formal Approach to Scenario Analysis', *IEEE Software*, **11**, 33-41.

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

Jackson M. (1995) *'Software Requirements and Specifications'*, ACM Press/Addison-Wesley.

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.

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

Lewycky, P. (1987) 'Notes towards understanding of accident causes', *Hazard Prevention*, 6-8.

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

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

Maiden N.A.M. and Rugg, G. (1996) 'ACRE: Selecting methods for Requirements Acquisition', *Software Engineering Journal*, **11**, 183-192.

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.

Maiden N.A.M. and Sutcliffe A.G. (1994) 'Requirements Critiquing Using Domain Abstractions', *Proceedings IEEE Conference on Requirements Engineering*, IEEE Computer Society Press, 184-193.

Maiden N.A.M. and Sutcliffe A.G. (1993) 'Requirements Engineering by Example: An Empirical Study', *Proceedings IEEE Symposium on Requirements Engineering*, IEEE Computer Society, 104-112.

Maiden N.A.M. and Sutcliffe A.G. (1992) 'Exploiting Reusable Specifications Through Analogy', *Communications of the ACM*, **34**, 55-64.

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

Potts C., Takahashi K. and Anton A.I. (1994) Inquiry-Based Requirements Analysis', *IEEE Software*, **11**, 21-32.

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

Rasmussen J. (1991) 'Event analysis and the problem of causality', in **J. Rasmussen**, **B. Brehmer and J. Leplat, editors.** *Distributed Decision Making: Cognitive Models for Co-operative Work*, John Wiley & Sons, New York, 247-256.

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

Rasmussen J. and Vicente K.J. (1989) Coping with Human Errors through System Design: Implications for ecological Interface Design', *Int. J. Man-Machine Studies*, **31**, 517-534.

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

Reason J. (1987) 'A Preliminary Classification of Mistakes', in **J. Rasmussen, K. Duncan, and J. Leplat, editors.** *New Technology and Human Error*, John Wiley & Sons, New York, 45-52.

Reese J.D. (1995) *Software Deviation Analysis*, Ph.D. thesis, University of California, Irvine, California.

Reubenstein H.B. and Walters R. C. (1991) 'The Requirements Apprentice: Automated Assistance for Requirements Acquisition', *IEEE Transactions on Software Engineering*, 17, 226-240.

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.

Sutcliffe A.G. (1997) ‘A Technique Combination Approach to Requirements Engineering’, *Proceedings IEEE International Symposium on Requirements Engineering*, 65-74.