CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements[1]

N.A.M. Maiden

Centre for HCI Design, City University, Northampton Square, London, UK
Tel: +44-171-477-8412
Fax: +44-171-477-8859
e-mail: N.A.M.Maiden@city.ac.uk

Abstract: *This paper reports research into semi-automatic generation of scenarios for validating system requirements. The research was undertaken as part of the ESPRIT IV 21903 'CREWS' long-term research project. The paper presents the underlying theoretical models of domain knowledge, computational mechanisms and user-driven dialogues needed for scenario generation. It describes how CREWS draws on theoretical results from the ESPRIT III 6353 'NATURE' basic research action, that is object system models which are abstractions of the fundamental features of different categories of problem domain. CREWS uses these models to generate normal course scenarios, then draws on theoretical and empirical research from cognitive science, human-computer interaction, collaborative systems and software engineering to generate alternative courses for these scenarios. The paper describes a computational mechanism for deriving use cases from object system models, simple rules for use case composition, taxonomies of classes of exceptions which give rise to alternative courses in scenarios, and a computational mechanism for generation of multiple scenarios from use case specifications.*

Keywords: Requirements engineering, domain modelling, scenario modelling, scenario generation, scenario walkthrough.

1 REQUIREMENTS ENGINEERING

Determining requirements for software-intensive systems is in need of new tools and methods. Although requirements engineering research has led to considerable advances, it has focused on the improvement of methods (e.g. Sommerville et al. 1993), tools (e.g. Johnson et al. 1992) and languages (e.g. Mylopoulos et al. 1990) which enable formal specification and validation of requirements. In comparison, acquisition of requirements with stakeholders has been neglected. Such acquisition is different because requirements are often expressed as linguistic expressions (Sutcliffe & Maiden 1993) which are not often amenable to formalisation or computational reasoning. One alternative is to use of informal scenarios to aid requirements acquisition and validation. Scenarios have received a lot of recent attention (e.g. Jacobson et al. 1992, Regnell et al. 1995, Graham 1996, Carroll 1995), however there is no consensus how best to use scenarios for requirements acquisition and validation. This paper reports work, undertaken as part of the ESPRIT 21093 'CREWS' (Co-operative Requirements Engineering With Scenarios) long-term research project. It

[1] This work was partly funded by the European Commission ESPRIT 21903 'CREWS' (Co-operative Requirements Engineering with Scenarios) long-term research project.

describes a prototype software tool for semi-automatic scenario generation, the development of which raises important issues about use case composition, scenario walkthroughs and computational mechanisms for scenario generation and use.

The ESPRIT 6353 'NATURE' basic research action (Jarke et al. 1993) has identified a large set of domain templates, known as object system models, to guide requirements engineers. Object system models are models of the fundamental behaviours, states, objects, agents, goals, constraints and functions which belong to members of possible categories of requirements engineering problem domain. In this sense, these models are similar to problem frames (Jackson 1995) or clichés (Reubenstein & Waters 1991). NATURE has derived a data base of object system models (e.g. Sutcliffe & Maiden, in press), undertaken empirical validation of these models (e.g. Maiden et al. 1995) and constructed tools to exploit the completeness, cohesiveness and structure of these models during activities such as requirements structuring (Maiden & Sutcliffe 1993), critiquing (Maiden & Sutcliffe 1994) and communication, as well as reuse (Maiden & Sutcliffe 1992).

This paper proposes the use of NATURE's object system models to provide guidance for scenario-based requirements acquisition and validation, and in particular as a basis for semi-automatic scenario generation. In practice until now, scenario generation has tended to be a craft undertaken by a few skilled individuals (Jarke et al. 1997, Potts et al. 1994). There are few guidelines with which to structure and bound scenarios, although exceptions (e.g. Cockburn 1995) are emerging. Studies undertaken as part of CREWS revealed that, as a result, scenarios generated by hand are often incomplete, in cohesive and lack structure (Jarke et al. 1997). This has led to a scenario generation 'bottleneck' in which scenarios are difficult and time-consuming to generate. To overcome this bottleneck CREWS proposes a software tool called CREWS-SAVRE to provide systematic guidance for scenario generation. NATURE's object system models provide an effective starting point for this scenario generation because each model describes reusable patterns of actions which are coherent and consistent descriptions of problem domains.

This paper has four remaining sections. First it reviews NATURE's object system models which are a starting point for scenario generation in CREWS. It then presents a scenario in which a user uses the CREWS-SAVRE software tool to generate and walkthrough a scenario for validating requirements for the 1992 London Ambulance Service Computer-Aided Despatch system (Dowell & Finkelstein 1996). CREWS-SAVRE's underlying architecture and computational mechanisms are then described. The paper ends with future plans for further development and evaluation of the CREWS-SAVRE tool and problems still to be overcome in scenario generation and use.

2. NATURE'S SET OF OBJECT SYSTEM MODELS

There have been several attempts to produce sets of problem models for reuse during requirements engineering (e.g. Reubenstein & Waters 1991, Constantopoulos et al. 1991). However, NATURE represents the first systematic attempt to model the space of requirements engineering problem domains. Its theoretical justification draws on hierarchical models of natural categories (e.g. Rosch 1983) and mental schemata (e.g. Riesbeck & Schank 1989) from cognitive science. It hypothesises that object system models are equivalent in categorisation, scale, content and structure to mental abstractions which are often recalled by requirements engineers. This is anticipated to maximise reuse of domain knowledge and lead to better recognition, understanding and adaptation of object system models. Alternative means of detecting and validating such object system models, such as domain analysis (Prieto-Diaz 1990), are time-consuming and difficult, and do not provide sufficient exposure to different domains to enable effective abstraction.

Object system models define all features of one category of requirements engineering problem domain which discriminate it from other problem domain categories. Each model is, in essence, an object pattern for one category of problem domain, similar to object patterns reported elsewhere (e.g. Coad et al. 1995, Buschmann et al. 1996). Object system models are defined in a systematic hierarchical structure. Models at different levels in this structure are distinguished using different knowledge types with different powers of discrimination. These knowledge types are the basis of our problem modelling language (Sutcliffe & Maiden in press). The object system models are structured in 13 hierarchies, see Figure 1. The top-level object system model of each hierarchy is defined using actions, states, agents, objects, states, state transitions and domain structure belonging to all members of one problem category. Specialisation of each of these models, in the form of systematic addition of different knowledge types (e.g. goal states, events), generates a space of over 200 leaf-node object system models. Each leaf-node model is defined using states, state transitions, events, objects, agents, domain structure, preconditions on transitions, goal states, and object properties and attributes.

The 13 top-level object system models 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. Each model includes one or more actions, events which start and end these actions, agents involved in these actions, and action decompositions. For example, domain simulation models describe complex environment simulations for human agents. The top-level model is specialised through addition of different domain structures (e.g. user is inside/outside the environment, number of users in the environment) to generate several more specific models. Each of these models is in turn specialised at lower levels of the hierarchies through addition of different event triggers (e.g. real-time nature of interaction), preconditions on actions (e.g. limits on user actions) and object properties (e.g. physical or conceptual). Furthermore, most of the resultant leaf-node object system models can be specialised through addition of different types of more domain-specific knowledge. Domain simulation models have different types of interaction devices which are application-specific (e.g. steering wheels, joysticks, keyboards). On the other hand, object sensing system models are specialised through addition of attributes which type sensing agents (e.g. radar, infra-red sensors, video cameras, pressure pads), moving objects (e.g. unidirectional or bi-directional objects) and spaces (e.g. three-dimensional versus two-dimensional space, moving versus fixed spaces). Indeed, object system models, when integrated with existing methods, provide an alternative starting point for more detailed domain modelling.

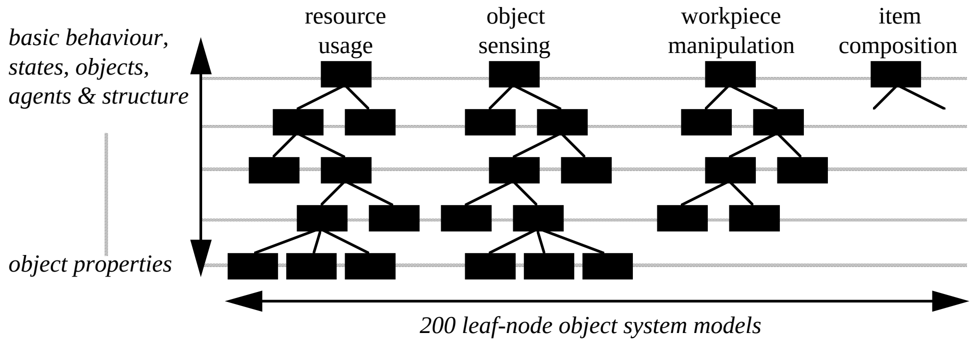

Figure 1: Graphical depiction of the hierarchical class structure of object system models.

To exploit object system models during requirements engineering tasks NATURE proposes the AIR (Advisor for Intelligent Reuse) toolkit, which is composed of six tools shown in Figure 2. The toolkit encourages iterative acquisition, definition and critiquing based on retrieval of more and lower-level object system models. More details of these tools are given in Sutcliffe & Maiden (in press).

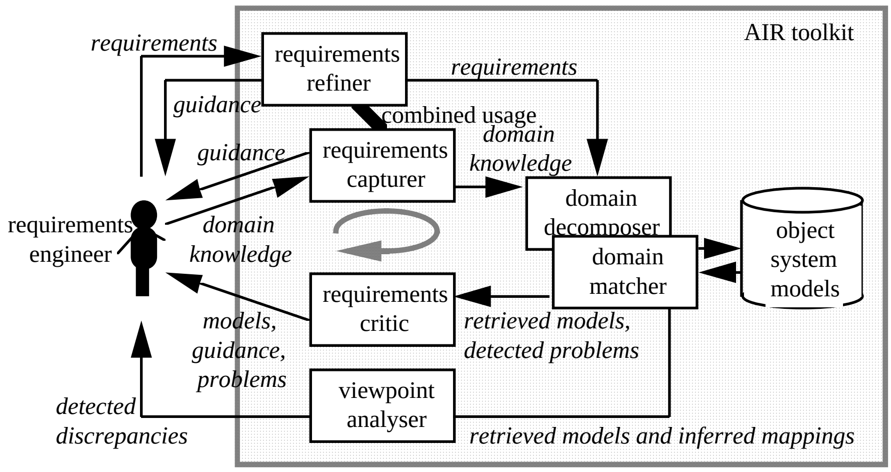

Figure 2: Outline architecture of NATURE's AIR toolkit.

At the heart of the toolkit are two computational mechanisms which retrieve object system models. The domain matcher is a computational analogical reasoning mechanism for retrieving object system models (Maiden & Sutcliffe 1996a). Retrieval is achieved by first matching entered facts and requirements to a high-level object system then refining this match to specialisations of the object system until no further specialisation is possible. The second mechanism, the domain decomposer, retrieves multiple object system models from a larger requirements problem domain description (Maiden & Sutcliffe 1996b). The decomposer combines domain semantics with pattern matching algorithms to detect fact and requirement patterns for then retrieving single object system models using the domain matcher. As well as direct retrieval of object system models, both mechanisms enable the user to browse the data base of models and select best-fit ones according to problem domain categories. As such the author believes that the AIR toolkit provides a cost-effective means of reusing problem domain knowledge during the earliest stages of requirements engineering.

In the approach reported in this paper, CREWS also uses these models to generate complete, coherent and structured scenarios. Each object system model describes normative events, actions, agents and objects for problem domain categories, so if categories are known it is possible to generate normal course scenarios from models corresponding to these categories. However, NATURE's object system models do not describe non-normative events and agents which often give rise to alternative courses, similar to obstacles in Potts et al. (Potts et al. 1994) and banana-skins in Graham (1996). To fill this gap CREWS provides taxonomies of non-normal events and states synthesised from research in human error in cognitive science (e.g. Reason 1990), human-computer interaction (e.g. Rasmussen et al. 1994) and machine failures in complex systems (e.g. Leveson 1995) with which to generate alternative courses in scenarios. This paper describes the role of NATURE's object system models and CREWS exception classes in the semi-automatic scenario generation with the CREWS-SAVRE software tool.

3 USE CASES & SCENARIOS

Most authors agree that, in broad terms, use cases and scenarios are descriptions of a sequence of actions or events for a specific case of some generic task which the system is meant to accomplish (e.g. Wexelblat 1987). Both are useful for acquiring and validating requirements because both are informal descriptions of the future software system which are useful for communication, explanation and negotiation. Both encapsulate contextual information which is often missing from most conceptual models. Furthermore stakeholders do not need to learn formal syntax to describe and understand them. However, a consensus is emerging (e.g. UML 1997, Graham 1996, Cockburn 1995) about an important difference between use cases and scenarios which the CREWS-SAVRE software tool exploits during scenario generation.

Use Cases

CREWS treats a use case as a specification of actions (with start and end events, agents that are involved in the action and objects used in the action) and rules that govern how actions are linked together. Action-link rules include strict sequence (A then B), sequential but with the second action starting only after the first has started but necessarily not finished (A and meanwhile B), concurrent where there are no rules about ordering of the start and end events of the action (A and B), and alternatives between use cases (A or B). These link types build on formal temporal semantics and calculus from the ALBERT II specification language (Dubois et al. 1997). As such a single use case specifies all possible combinations of actions which involve human and machine agents in the problem domain. As an example consider a customer using a self-service restaurant. We can use the CREWS-SAVRE tool to define one use case which specifies all actions and action-link rules. Example actions are Use-Hot-Food-Counter, Use-Cold-Food-Counter, and Pay-At-Checkout. Possible action-link rules include [Use-Hot-Food-Counter OR Use-Cold-Food-Counter] and [Use-Cold-Food-Counter THEN Pay-At-Checkout].

Scenarios

In contrast one scenario is treated as one specific ordering of events, the ordering of which is dependent on the start- and end- events for each action specified in the originating use case, which in turn is specified in the action-link rules belonging to the use case. Therefore more than one scenario can be generated from a single use case if one or more action-link rules are not strict sequence (i.e. not A then B). The sequential ordering of events makes it simpler to present scenarios in a text format and to walk through scenarios during requirements validation. Returning to our self-service restaurant, the CREWS-SAVRE tool generates two scenarios containing four events from the above use case:

1: start[Use-Hot-Food-Counter], end[Use-Hot-Food-Counter], start[Pay-At-Checkout], end[Pay-At-Checkout].
2: start[Use-Cold-Food-Counter], end[Use-Cold-Food-Counter], start[Pay-At-Checkout], end[Pay-At-Checkout].

The remainder of this paper describes the CREWS-SAVRE software tool then presents two views of one example of tool use. The first describes use of the tool by a single user to generate and use scenarios. The second describes the tool's functions which enables this generation and use, including a computational mechanism for deriving use cases from object system models, simple rules for use case composition, taxonomies of classes of exceptions which give rise to alternative courses in scenarios, and a computational mechanism for generation of multiple scenarios from use case specifications. The paper makes explicit reference to 9 key features of the tool (FEATURE-1, FEATURE-2, etc) to enable the reader to trace between these two views.

4 SEMI-AUTOMATIC SCENARIO GENERATION IN THE CREWS-SAVRE SOFTWARE TOOL

CREWS proposes semi-automatic generation of scenarios from NATURE's object system models. Its aim is to generate: (i) a simple use case specification from actions and action-link rules described in object system models, and; (ii) scenarios (event sequences) from permissible permutations of actions in the use case. These permutations are then extended using classes of exceptions which define unforeseen situations and events in problem domains. Therefore, in its simplest form, CREWS-SAVRE uses NATURE's object system models to generate a normal course use case and CREWS exception classes to generate first-pass alternative courses for this use case. The power of the tool arises from linking these two sources of domain knowledge using an intelligent permutation-generating algorithm.

The CREWS-SAVRE tool has been developed on a Windows-NT platform using Microsoft Visual C++ and Access, thus making it compatible for loose integration with leading commercial requirements management (e.g. Requisite Pro, RTM) and computer-aided software engineering (e.g. SELECT ENTERPRISE) software tools.

The current version supports the CREWS approach to scenario generation and walkthrough, and has 6 main functions:

1: generation of a first-pass normal course use case specification from object system model(s);
2: support for user-led action composition using simple rules to generate a more complex, normal course use case specification;
3: support for user selection of alternative courses from a data base of classes of exception events and states which give rise to such courses, to elaborate the specification;
4: support for user selection of parameters which constrain scenario generation;
5: automatic scenario generation from the use case specification;
6: systematic scenario walkthrough to validate system requirements.

The next section demonstrate the main functions and 'look-and-feel' of the second version of the CREWS-SAVRE prototype during a retrospective scenario analysis of part of 1992 London Ambulance Service (LAS) Computer Ambulance Despatching (CAD) system (e.g. Dowell & Finkelstein 1996).

4.1 The LAS-CAD System

In 1992 the LAS was looking to computerise core activities such as incident management. The aim of the CAD system was to automate as much of the call-taking and resource identification, mobilisation and management functions as possible, thus reducing the workload and reliance on control room staff. A single control room operator was to be responsible for a single incident, from call-taking to despatch of an ambulance to the incident. A complex and integrated software and hardware system was designed and implemented. This system included a computer-based gazetteer, a resource allocation system and an automatic vehicle location system. However, following the system's introduction in October 1992, a number of social and technical problems arose. The entire system descended into chaos, was taken off-line and the LAS reverted to the manual despatching system. The next section describes how the CREWS-SAVRE tool can generate scenarios which describe communication between the control room operator and ambulance crew members, and how a tool-led scenario walkthrough would have identified some of the now-recognised CAD system failures in the form of missing or incomplete system requirements. The section after that will describe the core mechanisms which enable the tool to provide this type of support for a user.

4.2 Generating and Using Scenarios for the LAS-CAD System

A requirements engineer called Peter Reid is using CREWS-SAVRE to generate scenarios for the LAS-CAD system. Recall that CREWS scenarios describe hypothetical situations in the environment in which the required system will be used, so each scenario describes events, actions and agents in the LAS's operational system. Figure 3 shows the first stage of Peter's interaction with CREWS-SAVRE's Domain Modeller tool.

The Domain Modeller: Peter can either generate use cases from scratch, retrieve reusable use cases from a data base, or choose NATURE object system models from which to generate use cases. Peter chooses to generate new use cases from the object system models, which he first browses and retrieves from the AIR toolkit's data base. Figure 3 shows that two object system models have been retrieved, see the box in the top left-hand corner. The first, object messaging (OM), describes actions, events, objects and states common to message communication between sender and receiver agents. The second, object reading (OR), describes events, actions, objects and states common to reading information from an object device. The Domain Modeller generates structured English descriptions of each action (e.g. "21. [OM.send] controller sends mobilisation decision to crew") from attributes of the action in the originating object system model which Peter instantiates to describe normal actions during control-room to ambulance-crew communication (FEATURE-1).

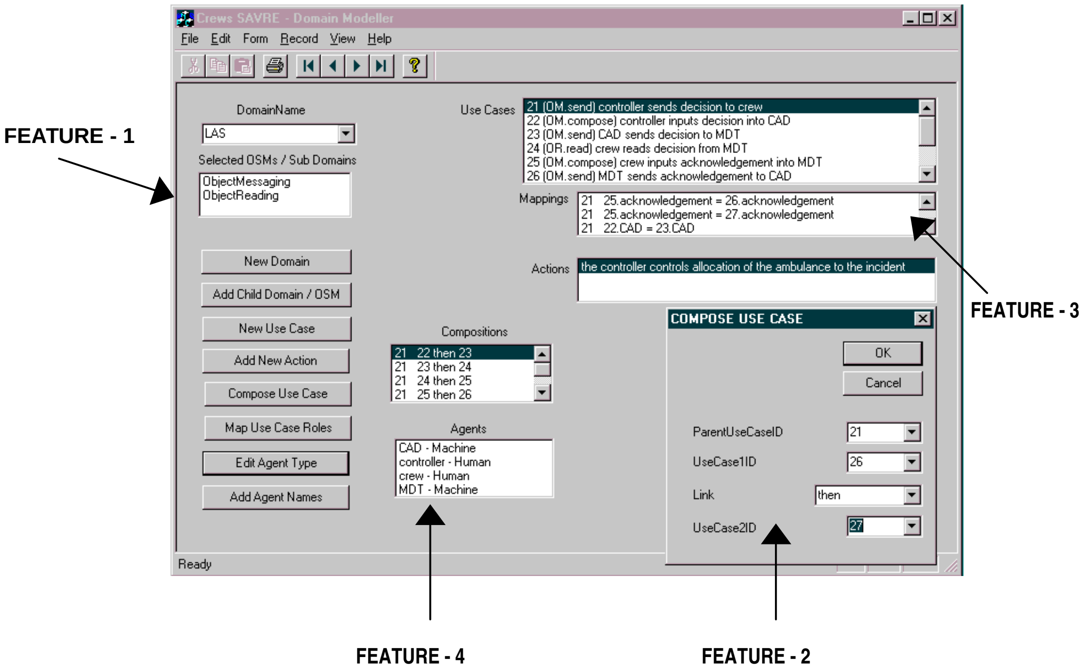

Figure 3: selection of use cases as a basis for scenario generation. Peter can compose actions together to form larger use cases.

Once Peter has selected a collection of use cases he can compose actions from these use cases together using action-link rules, as also shown in Figure 3. These rules shown state that use case number 21 (controller sends decision to crew) is a strict sequence of action numbers 22 to 27, and Peter is shown adding use case number 27 to this sequence (FEATURE-2). Peter is able to use the other function buttons shown in Figure 3 to define agent-roles, edit agent types, add agent names to the use case specification and, if appropriate, enter a natural language specification of actions in the use case. Each is described in turn:

- define agent-role mappings: agent-role mappings are an appropriate part of the use case specification. Each mapping links two agents described in two different actions. For example, Peter specifies that the agent "CAD" in use case number 22 is the same "CAD" agent as that in use case number 23, and that the agent "crew" in use case number 24 is the same as the agent "crew" in use case number 25 (FEATURE-3);

- edit agent-types: Peter can type each agent as either human (e.g. a controller), machine (e.g. a CAD), composite (e.g. an ambulance) or without type (FEATURE-4), although in the last case generation of alternative scenario courses will be less precise;
- add agent roles: to improve the readability of the scenario Peter can give names (simple strings) to agent-instances, for example the two ambulance-crew agents in the scenario are called Mary and Paul.
- enter natural language descriptions: the CREWS-SAVRE tool also supports natural language description of use cases and actions. Peter can enter the natural language description "the controller controls allocation of the ambulance to the incident", as also shown in Figure 3. CREWS uses libraries of sentence case patterns (e.g. Dik 1989) to parse natural language input into semantic networks prior to transforming the action description into a UML sequence diagram or, in this case, a CREWS-SAVRE use case specification. The natural language parser mechanism is linked to a mixed-initiative dialogue between Peter and the tool to resolve incompleteness, ambiguities, anaphoric references, as described at length in Achour (1997).

Process guidance from the CREWS-SAVRE Wizard then guides Peter to use the Scenario Modeller tool.

The Scenario Modeller: CREWS-SAVRE's wizard guides Peter to develop and generate one or more scenarios from this collection of use cases using the 4 command buttons shown on the left-hand side of Figure 4. These command buttons enable Peter to add and remove exception classes which give rise to different forms of alternative courses:

- the 'Select Generic Exceptions' command button enables Peter to add alternative courses which arise from the basic semantics of use cases and scenarios, that is events, actions, objects and states. For example a scenario can include events which repeat or do not occur, and actions which start but do not end;
- the 'Select Permutations Exceptions' button enables Peter to add alternative courses which arise from common combinations of events, actions, objects and states, for example two events which occur at the same time;
- the 'Select Permutation Options' button allows Peter to add more complex, tailorable alternative courses about frequencies and other temporal links between events and actions;
- the 'Select Problem Exceptions' button enables Peter to add alternative courses to explore familiar but non-normative events and states involving human and machine agents, their communication and their interaction. Figure 4 shows selection from a subset of the available machine-machine communication exception classes. A user is able to choose one or all exception classes in selected hierarchies. In this example Peter chooses all exception classes for machine-machine communication, in order to explore communication failures between the control room and ambulances.

The selected exception classes provide the input parameters to the scenario generation mechanism (FEATURE-5). Peter can also use the Scenario Modeller tool to select the desired topic focus for all scenarios (FEATURE-6), such as the action (22 [OM.compose] controller inputs decision into CAD) or the agent (controller). Peter then presses the 'GENERATE' button (FEATURE-7), chooses the required presentation format of the scenarios, and the tool generates a collection of scenarios for the use case specification. Now let us walk through a generated scenario with Peter.

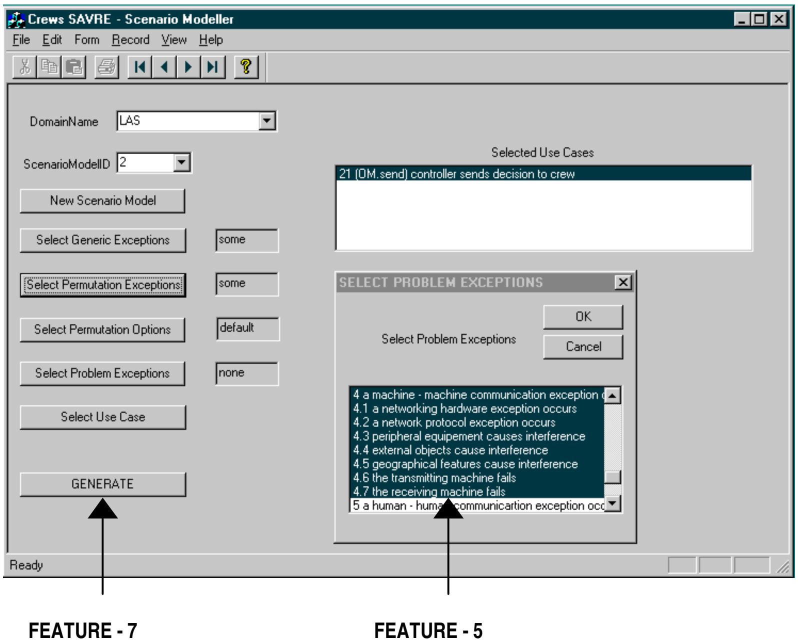

Figure 4. Selection of machine-machine communication exception classes for inclusion in scenarios for the CREWS-SAVRE tool to generate.

The Scenario Presenter: Figure 5 shows CREWS-SAVRE's standard presentation during scenario walkthroughs. The bottom right-hand corner shows process guidance given by the Wizard. The bottom left-hand corner shows the normal course of the generated scenario as a sequence of events which start or end actions in the use case. The type of each action is also shown. Peter uses the "Back" and "Forward" buttons to navigate through the scenario and select events. On the right-hand side are alternative courses generated automatically from Peter's earlier selection of generic, permutation and problem exception classes. These courses are presented in the form of what-if questions. For each selected event the tool presents alternative courses linked by the tool to that event (FEATURE-8), and advises Peter to decide whether each alternative course is (a) relevant, and (b) handled in the requirements specification. Figure 5 also shows part of requirement specification being validated.

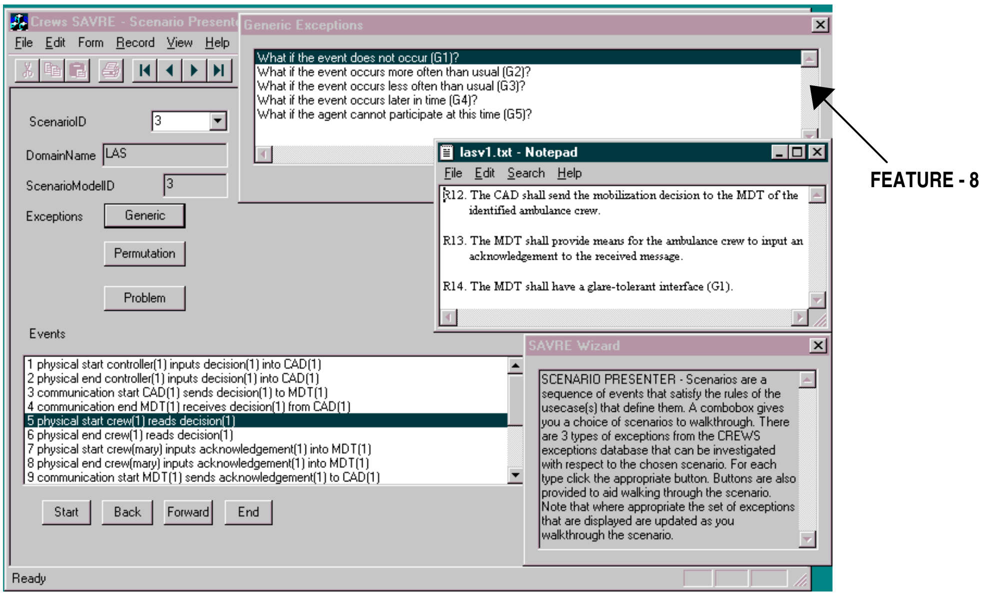

Figure 5: scenario walkthrough, exploring an alternative course arising from a generic exception. Possible additional requirements are discovered from exploring this course include giving crew-members portable communication devices such as walkie-talkies.

In Figure 5 Peter is exploring event-5, the start of the physical action [crew(1) reads decision(1)]. The tool proposes 5 alternative courses which were generated from the generic exceptions selected earlier. Peter is examining G1, what if event-5 does not occur. Each alternative course provides a 'seed' event or state that Peter can explore using techniques such as cause-consequence and risk analysis (Maiden et al. 1997a, 1997b). Peter decides that this course is relevant but not handled in the current specification. For example, one problem in the 1992 system was light and glare in the ambulance which meant that the incident details could not be seen on the MDT by crew members, hence the messages were not read. As a result Peter adds one or more relevant requirement statements to the specification. The one new statement added to the requirements notepad is 'R14: The MDTs shall have a glare-tolerant interface', with a reference back to exception G1 for traceability purposes. The Wizard guides Peter to explore the four other alternative courses G2-G5 in the same way.

Figure 6 shows another stage in the walkthrough. This time Peter is examining event-3 and alternative course 4.4, what if the CAD agent sends the decision (message) to the MDT agent but the decision (message) does to reach the MDT agent because of interference from external objects. Again this alternative course provides a 'seed' event or state that Peter can explore. This time Peter decides that this course is relevant and not handled by requirements in the current specification. London has a number of hills which led to communication blackspots which impeded use of the 1992 LAS-CAD system. However for this alternative course the tool is able to propose candidate generic requirements (FEATURE-9), see Figure 6. Peter can select one or more of these requirements and add them to the current requirement specification as he sees fit. In this example he chooses requirement GR1 "bypass the interference using a second communication channel". Again the wizard then guides Peter to explore the other alternative courses in the same way.

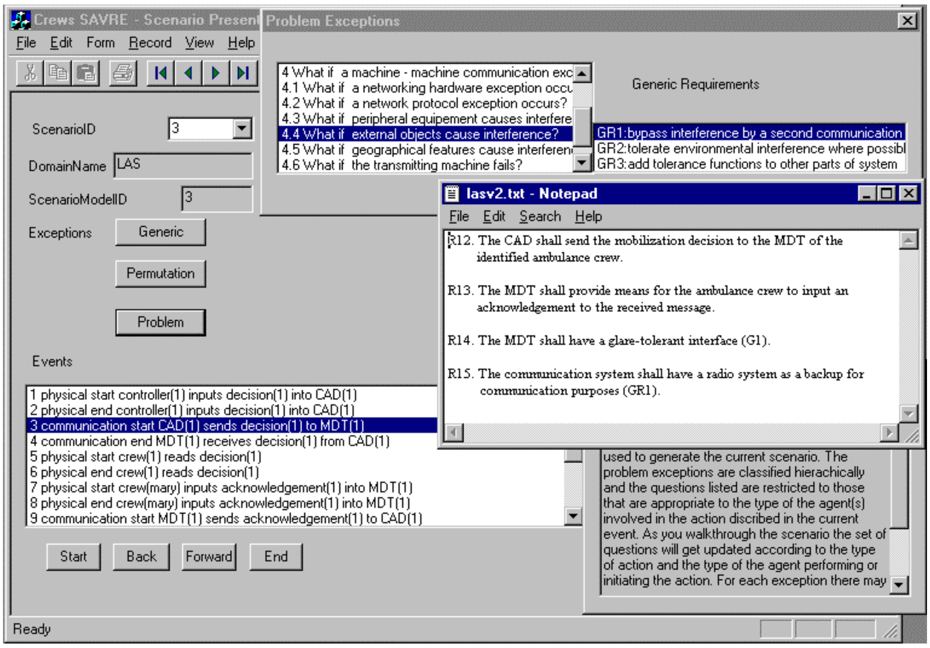

Figure 6. Scenario walkthrough, exploring an alternative course arising from a problem exception. At this stage the tool proposes three possible generic requirements to resolve or avoid the alternative course selected by Peter.

This simple example demonstrates some of the main functions of the CREWS-SAVRE tool. These functions include generation of use cases from object system models, action composition in a use case, automatic scenario generation, a software wizard which guides systematic scenario walkthrough and a data base of exception classes for generating useful alternative courses. The use cases and exception classes shown in the example are part of the tool and can be used to discover actual problems with the 1992 LAS-CAD system. The next section describes the computational mechanisms which enable the CREWS-SAVRE tool to provide such functionality.

5 COMPUTATIONAL MECHANISMS FOR SCENARIO GENERATION

The CREWS-SAVRE tool incorporates computational mechanisms to: (i) generate a use case specification from object system models: (ii) compose actions to form a larger use case specification: (iii) add classes of exceptions to be investigated during the scenario walkthrough to this specification, and: (iv) generate a useful set of scenarios from these use case specifications. Each is a novel contribution to scenario-based requirements engineering. Each is described in turn.

5.1 Generating Use Cases From Object System Models

NATURE's object system models describe the normative events, actions, agents and objects belonging to members of one category of problem domain. The CREWS-SAVRE tool generates the specification of a correct and coherent normal course use case using the retrieved object system models. In our example CREWS-SAVRE generates the use case specification shown in Figure 5 from the retrieved object messaging (OM) and object reading (OR) models. Let us explore how this happens.

The basic schematic structure and content of a CREWS-SAVRE use case is shown in Figure 7. An action is the core concept of the use case. Each action has one start event and one or more end events. Each action involves one or more agents. Each agent can be a human (e.g. a controller), machine (e.g. a MDT) or composite agent (e.g. an ambulance). Each agent-action involvement relation is specialisable to performs, initiates, ends, etc. Each action uses one or more objects, where an object is an infological representation of a durable real world phenomenon. Each action-object use relation is specialisable to accesses, reads, etc. Each action can also result in a state transition which changes the state of an object. Object states are sets of values which characterise an object at a given time. This basic structure of a use case will be extended to include agent roles, responsibilities and physical locations, devices and modes of communication actions, and resources available for successful completion of actions (see Sutcliffe 1997), although not in version-2 of the CREWS-SAVRE tool.

The basic schematic structure of NATURE's object system models is similar to that of a CREWS use case. An action is also a core concept of an object system model. Each action has one start event and one or more end events. Each action involves one or more agents, although the type of each is not specified. Each action can also result in a state transition which changes the state of an object. These similarities with the basic schematic structure of a CREWS use case means that a simple algorithm is sufficient to generate a normal course use case specification. Given one retrieved object system model (osm), one or more actions (ac) described in that model, events (ev) which start and end these actions, agents (ag) which are involved in these actions and objects (ob) used in that action, then the specification of the algorithm is:

```
FORALL osm
  [FORALL ac(ev,ag,ob) in osm
    [uc(ac,ev,ag,ob)=ac(ev,ag,ob)]]
```

where (uc(ac,ev,ag,ob)) is a simple use case specification specifying one action with start and end events, agents involved in the action and objects used in the action (FEATURE-1).

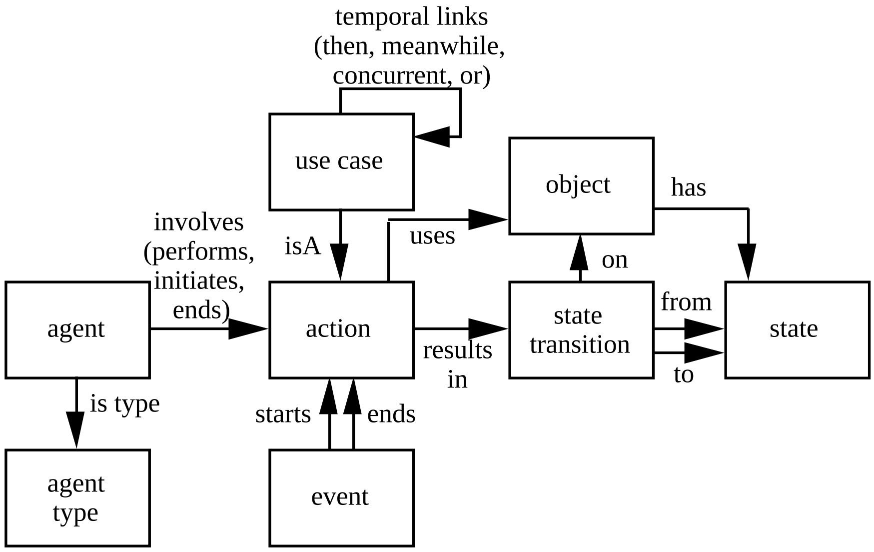

Figure 7. Meta-schema for modelling core concepts of a CREWS use case.

Returning to our example, Peter uses the AIR toolkit to retrieve the top-level object messaging (OM) model as an abstraction for both the incident-decision message and decision-acknowledgement message. The model itself describes two communication actions [sender-agent input message to sender-device] and [sender-agent send message to receiver-agent], thus two times retrieval of the models leads to the generation of 4 basic actions. Peter also retrieves the top-level object reading (OR) model to the communication action [reader-agent, read, information, from, object] for both the controller and crew agents. As a consequence the CREWS use-case-generation algorithm generates 6 basic actions:

action(22, OM.compose, sender-agent, input, message, to, sender-device).
action(23, OM.send, sender-agent, send, message, to, receiver-agent).
action(24, OR.read, reader-agent, read, information, from, object).
action(25, OM.compose, sender-agent, input, message, to, sender-device).
action(26, OM.send, sender-agent, send, message, to, receiver-agent).
action(27, OR.read, reader-agent, read, information, from, object).

The system allocates a unique action number (22-27) to each action. Peter then instantiates the 6 action specifications to fit the LAS problem domain using the functions shown in Figure 3.

5.2 More Complex Use Cases: Composing Actions

In the next stage Peter composes the 6 instantiated actions together using action-link rules to form a larger, more complex use case typical of those found in industrial practice (e.g. Jarke et al. 1997). He specifies a new top-level action 21 as a strict sequence of the actions 22, 23, 24, 25, 26 and 27. To enable him to do this CREWS provides the following simple action-link rules.

Action-link rules: CREWS defines 8 basic types of action-link rule between two actions A and B, where each event ev(X) represents the point in time at which the event occurs. For all action-link rules, ev(startA)<ev(endA) and ev(startB)<ev(endB), that is all actions have a duration and an event which start an action must occur before events which end the same action. The 8 types of action-link rule are:

- strict sequence (A then B): ev(endA)<ev(startB);
- part sequence (A meanwhile B): (ev(startA)<ev(startB)) AND ((ev(endA)>(ev(startB));
- inclusion (A includes B): (ev(startA)<ev(startB)) AND ((ev(endA)>(ev(endB));
- concurrent (A and B): no rules about ordering of events;
- alternative (A or B): (ev(startA) occurs) OR (ev(startB) occurs);
- parallel and equal (A parallel B): (ev(startA)=ev(startB)) AND ((ev(endA)=(ev(endB));
- equal-start (A starts-with B): (ev(startA)=ev(startB)) AND ((ev(endA)not=(ev(endB));
- equal-end (A ends-with B): (ev(endA)=ev(endB)) AND ((ev(startA)not=(ev(startB)).

These link types build on basic research in temporal logic (e.g. Moszkowski 1986) and formal temporal semantics and calculus from a real-time temporal logic called ALBERT-CORE which underpins scenario animation in CREWS using the ALBERT II specification language (Heymans & Dubois 1997). The current CREWS-SAVRE tool does not provide a complete set of action-link types. Rather it provides a set of useful and usable semantics based on action-link types in use cases from practice (Achour & Rolland 1997) and text books which describe use case analysis (e.g. Jacobson et al. 1992). In the LAS-CAD example Peter uses the strict sequence action-link rule and specifies action 21 as a strict sequence of the 6 actions (22 then 23, 23 then 24, 24 then 25, 25 then 26, 26 then 27), see FEATURE-2. Peter can then elaborate these links using separate object-mapping and agent-mapping rules to refine the normal course specification and enable generation of more precise and hence useful alternative scenario courses.

Object-mapping rules: CREWS defines two rule types for describing the mapping between two objects A and B in two different actions in the same use case specification:
- same (A=B): objectA=objectB;
- not same (A≠B): objectA≠objectB.

For example the LAS-CAD use case specifies two objects, "decision" and "acknowledgement". Peter specifies that the "decision" objects in actions 22, 23 and 24 are the same, that the "acknowledgement" objects in actions 25, 26 and 27 are the same, and that the "decision" object in action 24 and that "acknowledgement" object in action 25 are not the same.

Agent-mapping rules: CREWS defines five agent-mapping rule types for describing the possible combinations of two agents A and B in two different actions in a use case specification:

- same (A=B): objectA=objectB;
- not same (A≠B): objectA≠objectB;
- role-instance: (A role-instance B): agentA role-instance-of agentB;
- class-role: (A class-role B): agentA class-role-of agentB;
- composition: (A composition of B): agentA composition agentB.

The first two rules are the same as for object-mapping rules. The third and fourth rules exploit an important distinction in use cases between role-types, role-instances and agent-instances (FEATURE-4). CREWS defines an agent’s role-type as the formal role type of that object in a use case, similar to Blyth’s (1996) definition. Examples of agent role-types such as "controller" are common in use case specifications. However a use case specification must distinguish between instances of controllers if, for example, more than one controller is involved in actions in the use case. In natural language descriptions of use cases this distinction is made through explicit and anaphoric reference (e.g. "that controller" and "them") to agents. Therefore CREWS introduces the notion of a role-instance to differentiate between multiple instances of role-types in the same use case. Role instances are not object instances but rules which determine how role types might be instantiated. In our example "a controller" is a role, "that controller", "controller(1)" and "a controller, known as John" are all role instances, whereas "John Smith, the Head-Controller" is an agent instance which brings additional contextual knowledge to the use case in the form of general knowledge about John Smith the individual. Therefore extending the CREWS definition of a use case, each agent can be a role-type, role-instance or agent-instance. The example LAS-CAD use case specifies the agents "controller", "CAD", "MDT", "terminal" and "mary". Peter specifies that "controller" in actions 22 and 27 are the same agent, "CAD" in actions 22, 23 and 27 are the same agent, "mary" in actions 25 and 26 are the same agent, and "MDT" in action 23 is the same agent as "terminal" in action 24.

The fifth agent-mapping rule, composition, borrows the notion of whole-part structures from object-orientation. One agent A is said to be composed of another agent B if A is an assembly of parts of B, if A contains B, or if A is a collection of members of B (Coad & Yourdon 1991). In the example Peter specifies the agent "crew" as composed of agent "paul" and agent "mary". Therefore specification of the normal course of use case 21 is:

```
usecase specification(21,OM.send) has:
  actions:
    action(22, OM.compose, controller, input, decision, to, CAD).
    action(23, OM.send, CAD, send, decision, MDT).
    action(24, OR.read, crew, read, decision, from, terminal).
    action(25, OM.compose, mary, input, acknowledgement, to, MDT).
    action(26, OM.send, mary, send, acknowledgement, to CAD).
    action(27, OR.read, controller, read, acknowledgement, from, CAD).
  action-link-rules:
    action-link-rule(22 then 23).
    action-link-rule(23 then 24).
    action-link-rule(24 then 25).
    action-link-rule(25 then 26).
```

```
    action-link-rule(26 then 27).
  agent-mapping-rules:
    agent-mapping-rule(22.controller=27.controller).
    agent-mapping-rule(22.CAD=23.CAD).
    agent-mapping-rule(23.CAD=27.CAD).
    agent-mapping-rule(23.MDT=24.terminal).
    agent-mapping-rule(24.mary composes 25.crew).
    agent-mapping-rule(x.peter composes 25.crew).
    agent-mapping-rule(25.mary=26.mary).
  object-mapping-rules:
    object-mapping-rule(22.decision=23.decision).
    object-mapping-rule(23.decision=24.decision).
    object-mapping-rule(25.acknowledgement=26.acknowledgement).
    object-mapping-rule(26.acknowledgement=27.acknowledgement).
    object-mapping-rule(22.decision≠25.acknowledgement).
end usecase specification
```

This completes specification of the normal course use case. Peter then specifies alternative courses for the same use case using the Scenario Modeller tool (FEATURE-5).

5.3 Classes of Exceptions

There has been considerable research in human error in cognitive science (e.g. Reason 1990), interaction failures in human-computer interaction (e.g. Rasmussen et al. 1994) and machine failures in hazard analysis (e.g. Leveson 1995). CREWS draws on this research to provide exception class hierarchies which predict useful alternative courses in scenarios for requirements engineering. The CREWS-SAVRE tool accesses a data base of these classes during use case specification, scenario generation and scenario walkthrough. CREWS defines an exception as:

> "a state or event that is necessary but not sufficient for the occurrence of an undesired or non-normative behaviour of the required system. If the exception is regarded as being sufficient for the occurrence of such a behaviour, it is said to be the cause".

It distinguishes between three high-level types of exception class: generic, permutation and problem exception classes.

Generic and permutation exception classes: generic exception class definitions are based on the properties of events, actions, objects and states which in turn arise from CREWS definitions of events, actions, objects and states. For example event properties include the total number of occurrences of the event and frequencies of event occurrence over time, therefore CREWS defines generic exception classes about event frequencies and occurrence. Action properties include information inputs and outputs to the action, and exception classes are defined for abnormal action inputs and outputs. Examples of alternative courses generated from such generic exception classes are shown in Figure 5. The current data base contains 20 such generic exception classes.

Permutation exception classes are effective for determining unforeseen but important combinations of events, actions and states which often give rise to system failures (Leveson 1995). Elements such as events, actions, objects and states can be linked in different permutations according to their semantics. For example, two events can occur at the same time, two actions are the same instance of an action, and two agent role-types are fulfilled by the same agent instance in a single use case. The current data base contains 15 such permutation exception classes.

- **Problem exception classes**: CREWS has undertaken a more extensive classification of problem exceptions. It has developed 8 orthogonal classifications, each of which has a particular focus and literature. The first five classifications draw on CREWS definitions of actions and agents. An action can involve one or more human agents and/or one more machine agents, and a communication action between two or more human agents, two or more machine agents or a mix of two or more human and machine agents. This minimum set of possible communication actions enables us to define 5 problem exception classifications specific to human agents, machine 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:

- human agents: people are often give rise to numerous alternative courses in 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 science (Reason 1990, Norman 1988) and human factors research (Sutcliffe and Rugg 1994). Consider cognitive exceptions. Norman’s model of slips (Norman 1988) and Rasmussen’s three levels of human-task mismatches (Rasmussen 1994) are good examples of classic cognitive psychology error models. Starting with the basic classification framework of Rasmussen’s three levels of cognitive control, the hierarchy of cognitive model of exceptions has been populated by other models of human error, notably those of Reason and Norman. However, human exceptions cannot be adequately described by only cognitive factors; we have considered other performance affecting properties such as physical exceptions (anatomical, sickness, fatigue, age and gender);
- human-machine interaction: unforeseen human-computer interactions can also lead to numerous alternative courses. This time our classification draws on taxonomies of interaction failures from human-computer interaction (Rasmussen et al. 1994, Hollnagel and Kirwan 1996) and consequences from poor interface design (e.g. Nielsen 1993);
- human-human communication: scenarios often involve more than one human agent. Communication breakdowns between people have important consequences, and are an important source of alternative courses in scenarios. Exception classes have been derived from theories from computer-supported collaborative working (e.g. Shneiderman 1992). Examples include communication breakdowns and misunderstandings;
- machine-machine communication: scenarios also often involve more than machine agent, and exceptions specific to their communication can also give rise to alternative courses.

Other exception class hierarchies include information-model exception classes (e.g. the information is incorrect, out-of-date etc.), work-domain exception classes (e.g. poor lighting, noise etc.) and organisation situations (e.g. policies, social climate, management attitudes). All of these classes enable the tool to generate useful scenario alternative courses in a systematic manner. The current data base contains just over 100 exception classes. In the LAS-CAD example Peter selects exception classes such as PE4, that is all machine-machine exceptions (see Figure 4), to explore alternative courses during communication between the CAD and MDT agents shown in the scenario in Figure 6. As a result the use case specification include the additional statement for exception classes:

exception-classes:
PE4, PE4.1, PE4.2, PE4.3, PE4.4, PE4.5, PE4.6, PE4.7

The complete use case specification becomes the input to the scenario generation algorithm when Peter clicks the 'GENERATE' scenarios button (FEATURE-7).

5.4 Scenario Generation from Use Case Specifications

The CREWS-SAVRE tool generates one or more scenarios from a use case specification. The generation mechanism has two parts:

1) generation of each permissible normal course scenario from actions and action-link, agent-mapping and object-mapping rules in the use case specification. In the example the tool generates one scenario shown in Figures 5 and 6. This is because the use case specifies a strict sequence between actions, so that one event sequence permutation is possible. Specification of other action-link rules such as 'meanwhile' generate one scenario for each possible event sequence;
2) generation of alternative courses from classes of exceptions specified in the use case specification. Each class of exception gives rise to one or more alternative courses which are presented in the current version of the tool as "what-if" questions.

5.4.1 The Normal Course Scenario Generation Algorithm

The algorithm for normal course scenario generation is defined as a simple condition-action rule which is applied for all possible sequences of events (es) which start or end actions in the use case specification:

FORALL (es)
- IF (es consistent-with ad) AND (es consistent-with al) AND (es consistent-with uc)
- THEN (generate-scenario=es)

END

where (al) is all constraints imposed with action-link rules in the use case specification, (uc) is all constraints imposed with rules specified by the user in the specification, and (dc) is a constraint which ensures that all actions have a duration. The algorithm uses these three constraint types to constrain the space of all possible event sequences to determine the space of permissible event sequences, that is scenarios to generate. Consider a simple use case with two actions A and B. Event-1 starts action-A, event-2 ends action-A, event-3 starts action-B and event-4 ends action-B. The algorithm generates an event ending an action for each event which starts the same action. As a result there are 16 possible event sequences (A to P) for these four events:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1)	1	1	1	1	1	1	3	3	4	4	2	2	4	2	3	4
2)	2	2	3	3	4	4	2	2	2	2	1	4	1	3	1	3
3)	3	4	4	2	3	2	1	4	3	1	3	3	3	1	2	2
4)	4	3	2	4	2	3	4	1	1	3	4	1	2	4	4	1

Each of these 16 possible event sequences might be a scenario, however the 3 constraint types filter and hence reduce the number of permissible event sequences. Let us consider each constraint type in turn.

Action-duration (ad) constraints: all actions have a duration, so the start of an action must occur before the end of the action. Therefore the constraint applicable to all event sequences (constraint ad) is:

- (ev(startA) < ev(endA)) AND (ev(startB) < ev(endB))

where ev(startA), ev(endA), ev(startB) and ev(endB) are four events in the same event sequence which start and end Actions A and B. In the example event-1 must occur before event-2 and event-3 must occur before event-4, so the 4 permissible event sequences are A, C, D and O.

Action-link (al) constraints: now assume the use case specification specifies the action-link rule `(A then B)`, that is a strict sequence between actions A and B. As a consequence the constraint on permissible event sequences al is:

- ev(endA) < ev(startB)

where ev(endA) and ev(startB) are two events in the same event sequence and A and B are two actions in the use case specification. In our example event-2 must occur before event-3, so event sequence A (1,2,3,4) is the only permissible sequence. Now instead, assume the use case specification specifies the action-link rule `(A meanwhile B)`, a partial sequence between actions A and B. This time the constraint al is:

- ev(startA)<ev(startB) AND ev(endA)>(ev(startB)

This time event-1 must occur before event 3 and event-2 must occur before event-4, so event sequence D (1,3,2,4) is the only permissible event sequence. A third example is the use case specifies the rule `(A or B)`, so the constraint al is:

- (ev(startA)) OR (ev(startB))

so event-1 must occur or event-3 must occur. This time the algorithm generates two permissible event sequences (1,2) and (3,4) which are not in the original space of possible event sequences. The algorithm handles 'alternative' action-link rules through multiple application of the ul, al and uc constraints to multiple possible event sequence spaces. The algorithm: (i) generates one space of possible event sequences for each alternative or combination of alternatives; (ii) applies ul, al and uc constraints to all spaces of possible event sequences to determine permissible event sequences, and; (iii) removes duplicate permissible event sequences arising from the different spaces. In a fourth example the use case specification includes two or more action-link rules between the same two actions, for instance `(A meanwhile B) and (A ends-with B).`The constraint (al) is:

- ev(startA)<ev(startB) AND ev(endA)>ev(startB) AND ev(endA)=ev(endB)

This time event-1 must occur before event 3 and event-2 must occur at the same time as event-4, so event sequence D (1,3,(2&4)) is the only permissible event sequence. Although combining action-link rules gives Peter greater power of use case specification, illegal rule combinations such as `(A then B) and (A or B)` are always possible. The current version of CREWS-SAVRE does not check for inconsistent rule combinations, and is dependent on careful user specification for successful scenario generation. However, future versions of the CREWS-SAVRE tool will include a checker to filter out inconsistent use case specifications prior to scenario generation. Such a checker is a prerequisite to effective industrialisation of the scenario generation algorithm.

User-rule (uc) constraints: there are two constraint types which the algorithm can impose to restrict scenarios that are generated to those with a topic focus (FEATURE-6). The first restricts the algorithm to generate scenarios to include one or more actions selected beforehand by the user. It defines permissible event sequences as sequences (es) which include an event (ev) that starts the action A:

- (ev(startA) in es) AND (ev starts A)

In our simple example it is event-1 that starts action A, so the permissible event sequences are still A, C, D and O. The second constraint type defines permissible event sequences as sequences (es) which include an event (ev) that starts an action (A) which involve a predefined agent (ag):

- (ev(startA) in es) AND (ev starts A) AND (A involves ag)

These two user-rule constraints enable the user to have maximum control over scenario generation. Let us now examine the use of these constraints when generating scenarios from the more complex LAS-CAD use case specification.

Scenarios for the LAS-CAD problem domain: Figure 8 shows scenario generation from part of the use case which specifies actions 22, 23, 24 and 25, that is controller inputs the decision to CAD, CAD sends decision to MDT, crew reads decision from terminal, and Mary inputs acknowledgement into MDT. To make generation more interesting I changed the action-link rule between actions 23 and 24 to a partial sequence (23 meanwhile 24), so that action 24 starts after the start of action 23 and ends at some time after the end of action 23, that is the crew might continue to read the decision from the terminal after the acknowledgement actions begin. The algorithm generates the 4 scenarios.

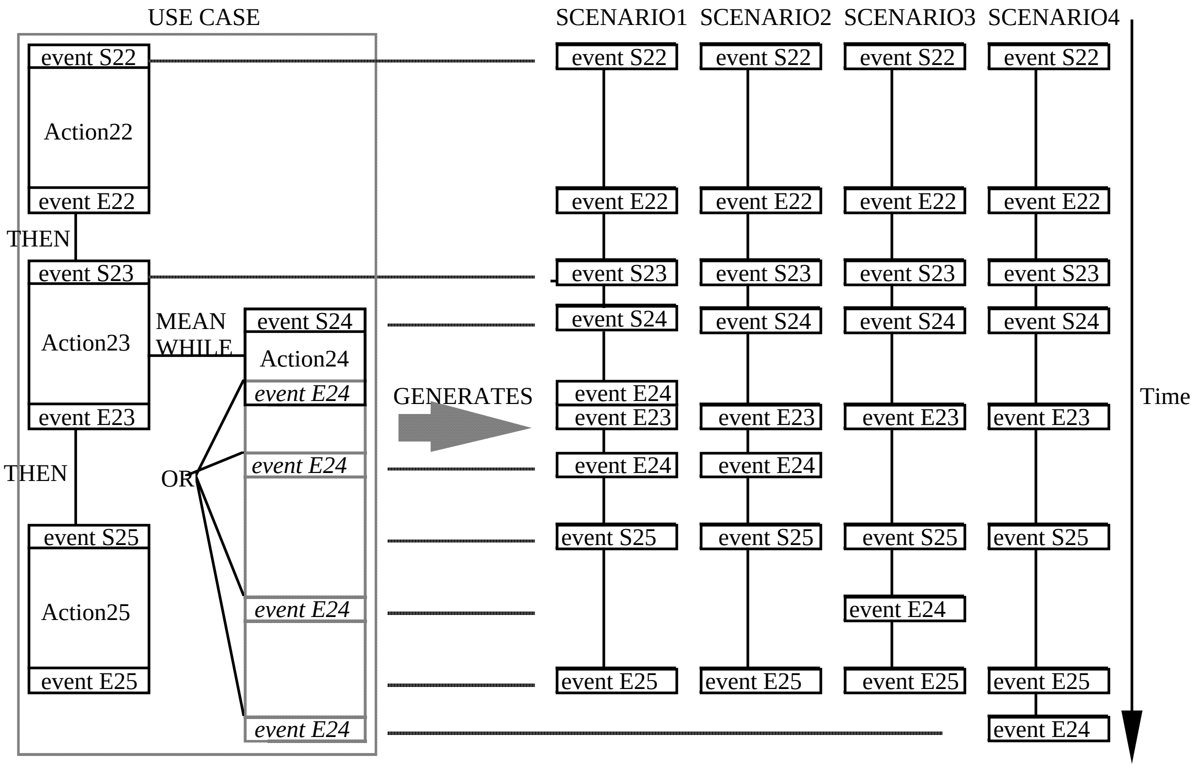

Figure 8. A graphical depiction of scenario generation from a simple use case. The use case definition leads to the 4 possible scenarios shown. The difference between each is the timing of event E25 which is the end of action 25.

In scenario-1 the crew ends the read-decision action before the send-decision action ends, so one possible consequence is the crew do not read the complete message and misinterpret it. In scenario-3 the crew ends the read-decision action after the start of the input- acknowledgement action, so possible system requirements include changing the input acknowledgement message on the MDT, displaying the entire message on the MDT, and disabling the MDT’s acknowledge function until the message is read. In scenario-4 the crew ends the read-decision action after the input-acknowledgement ends. Again one possible system requirement is to be able to delete and enter a complete new acknowledgement message. Scenario-2 represents the original normal course, in which the crew ends the read-decision action after the send-decision ends and before starting the input-acknowledgement action. The 1992 LAS-CAD system encountered all 4 scenarios, but problems with use of the MDT devices indicate that the requirements analysis for the LAS-CAD system did not predict these problems, hence subsequent poor design of the system.

5.4.2 The Alternative Course Scenario Generation Algorithm

The purpose of this algorithm is to generate permissible alternative course scenarios from the use case specification. The tool links alternative courses to each event in the normal course of a scenario using a simple type model of events, actions and exception classes, see Figure 9. Generic and permutation exception classes have a direct link to each event in the scenario, but links between events and problem exception classes are more complex (FEATURE-8). Each event starts or ends an action which involves agents and can change the state of objects. The CREWS meta-model defines permissible types for each action (physical, cognitive, communicative), agent (human, machine, composite) and exception class (human, machine, different types of communication, etc.). Sixteen heuristics use these types to select alternative courses for each event (ev), action (ac), agent (ag) and problem exception class (pe) in a scenario, for example:

- IF ((ev1 starts ac1) OR (ev1 ends ac1)) AND ac1(type=cognitive)
  THEN (pe(type=human) applies-to ev1).

- IF ((ev1 starts ac1) OR (ev1 ends ac1)) AND ac1(type=communicative)
  AND (ac1 involves ag1) AND ag1(type=machine)
  AND (ac1 involves ag2) AND ag2(type=machine)
  THEN (pe(type=machine-machine-communication) applies-to ev1).

We believe that these 16 heuristics provide an important basis for constraining the generation of alternative courses for each event, in addition to the user-defined constraints from parameters set during use case composition. Referring back to the selected problem exception classes, all classes are of type=machine-machine-communication, therefore the tool generates a what-if question for each event which starts or ends a communication action between the CAD and MDT machine agents, as defined in the second rule above. Consequences of this rule firing are seen in the scenario walkthrough in Figure 6. The Scenario Presenter presents 'what-if' alternative courses and generic requirement statements which are stored in a second but connected data base (FEATURE-9). These statements include mitigation and defensive mechanisms which handle the occurrence of one or more exception classes, thus providing more on-line help to the user. Other types of problem exception class and generic requirement statement are selected and presented for other events using the same ruleset.

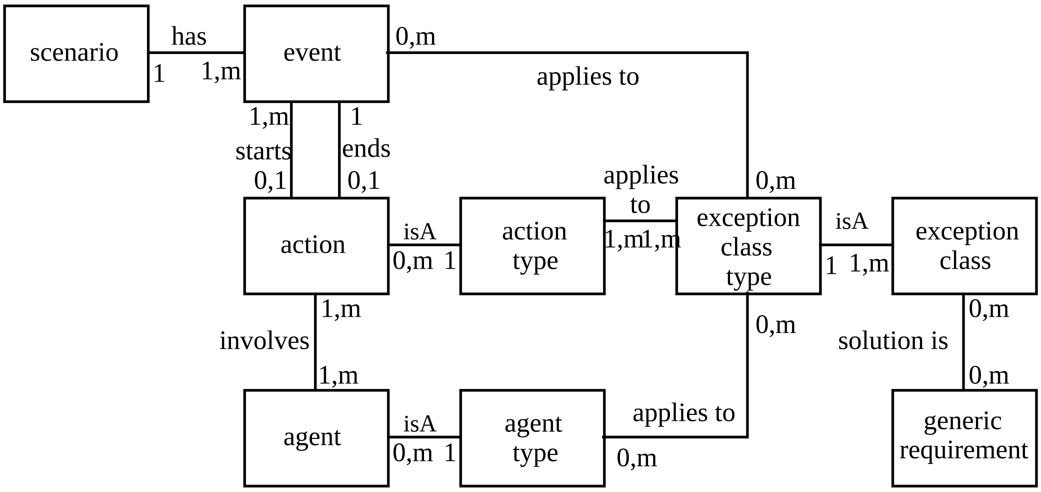

Figure 9: event, action, object and exception class types and links modelled in the CREWS-SAVRE tool. The links enable tool to navigate between event, action and exception class according to type in order to generate meaningful alternative courses for an event.

5.5 Future Development of CREWS-SAVRE

The second version of the CREWS-SAVRE tool provides core functions which enable use case specification, scenario generation and scenario walkthrough. Future versions will refine the scenario generation mechanism and provide more user functions. The first version was a simple prototype used to acquire user requirements for later versions of the tool (Maiden et al. 1998). To this end future versions will include scenario visualisation, functions for tailoring classes of exceptions to specific domains and an algebra for scenario composition to handle event interrupts common in real-time domains such as aircraft flight systems, see Maiden et al. (1998) for more details. The Scenario Modeller tool will be extended to enable users to assign probabilities of occurrence and scores indicating how severe the occurrence is to different classes of exception: it will encourage the user to undertake risk analyses during use case specification to determine probabilities which will enable the scenario presenter to prioritise alternative courses during scenario walkthrough. In addition, further theoretical research will link NATURE object system models to CREWS classes of exceptions, thus giving the CREWS-SAVRE tool some predictive power when selecting class of exceptions for categories of problem domain. The Scenario Modeller will also enable users to store and retrieve scenario generation profiles, that is reusable sets of parameters for generating different 'types' of scenarios for different acquisition and validation tasks, requirements standards and so on. Last but not least, CREWS hopes to link the CREWS-SAVRE tool to commercial object-oriented analysis tools such as SOMATiK (Graham 1996) and SELECT, to provide more user functionality and graphical modelling support during industrial evaluation of the CREWS-SAVRE tool. This integration will take place alongside loose integration with two other CREWS software prototypes for scenario-driven requirements acquisition (Pohl & Haumer 1997) and scenario animation during walkthrough (Heymans & Dubois 1997).

This will enable thorough evaluation of the CREWS approach during the second phase of the project.

6 FUTURE RESEARCH

The second version of the CREWS-SAVRE software tool combines a scenario-specific event-sequence permutation mechanism with interactive user functions to give the user extensive control over the scenario generation process. The division between: (i) use case specification; (ii) normal course scenario generation, and; (iii) alternative course scenario generation enables the user to intervene during and between specification and generation activities, thus ensuring all scenarios are relevant and their number is manageable. This semi-automatic and interactive scenario generation process contrasts with other approaches (e.g. Fickas & Nagarajan 1988) which exclude user involvement in scenario generation and walkthrough. However, despite this fragmented generation process, CREWS-SAVRE presents each generated scenario to the user as a coherent set of normal course event sequences, alternative courses and generic requirement statements to ensure that each scenario is simple to use and quick to browse and assimilate. It still allows the user to trace back into scenario generation process to extract rationale and, where appropriate, change the final scenario. A recent review of scenario use undertaken as part of CREWS (Pohl et al. 1997) reveals that capturing scenario rationale as traceability links was not addressed in existing scenario and requirements engineering tools, hence future versions of CREWS-SAVRE will embed simple requirements-scenario traces building on NATURE's process trace model (Pohl 1996).

The CREWS view of requirements engineering as embedding software systems in socio-technical environments (Jarke et al. 1993) means that CREWS scenarios are often non-deterministic, in contrast to scenarios in methods such as SCR (Archer & Heitmeyer 1997) which includes a specification simulator to generate and animate deterministic system behaviour. The ACME/PRIME method (Feblowitz et al. 1996) also relies on scenarios, this time to acquire business processes, however unlike CREWS there is little focus on non-deterministic alternative courses. CREWS-SAVRE also bears comparison with critiquing software tools such as the domain-oriented design environments from Fischer et al. (1991) and in particular with KATE (Fickas & Nagarajan 1988) which uses simulation scenarios of environment behaviour to critique atomic requirement statements. As a result KATE's scenarios assist exploration and elaboration of these requirement statements, but there is little focus on alternative course generation to explore requirements coverage which is one of the anticipated strengths of CREWS-SAVRE. In contrast the CREWS-SAVRE tool generates complex scenarios with ill-defined structure boundaries more familiar to socio-technical system scenarios reported in Eason & Harker (1996). However the key difference here is that CREWS-SAVRE provides prescriptive guidance and software tool support while the ORDIT method offers simple heuristic guidance.

One potential weakness of the CREWS-SAVRE approach is that scenario generation and walkthrough is domain-driven, in contrast to goal-driven scenario analysis from Potts et al. (1994) and task-driven use case modelling in SOMATiK (Graham 1996). If scenarios are to validate requirements, these requirements must inform scenario generation. The next version of CREWS-SAVRE will include a simple tool for selecting requirements to validate then guiding the transformation of these requirements to generate actions, agents, objects and states in a use case specification which constrain scenario generation, similar to NATURE's requirements transformation heuristics (Sutcliffe & Maiden 1993). For example, consider the simple transformation of an invariant state to an object state in a scenario: if the requirements engineer specifies an object state as an invariant, CREWS-SAVRE will generate all scenarios which include this state and actions which result in changes to that state. This extension to the CREWS-SAVRE tool will complete the scenario cycle shown in Figure 10 in which system requirements inform generation of CREWS scenarios which inform acquisition and validation of system requirements.

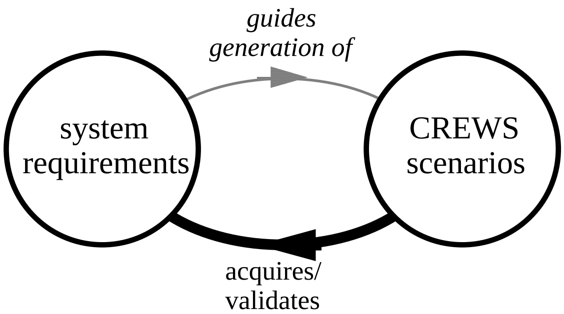

Figure 10: iterative cycle: system requirements inform generation of CREWS scenarios which inform acquisition and validation of system requirements.

Another limitation of CREWS-SAVRE's mechanistic approach to scenario generation is the possible lack of thematic progression or "story telling" in the scenarios. Potts (1995) reports that effective scenarios for requirements validation often have such a clear theme. Potential advantages from that the scenario is both interesting and easier to understand. Indeed, familiar stories can cue recall of similar experiences, problems and solutions (Schank 1982, Gick & Holyoak 1983). The next version of CREWS-SAVRE will prioritise theme scenarios which a user enters through a simple scenario authoring tool as a sequence of mandatory actions, agents, objects and states. CREWS-SAVRE will generate scenarios around this theme. It is also hoped to present theme scenarios in natural language through a reversal of the CREWS natural language parser (Achour 1997), transforming the scenario specification into a semantic network then parsing it into structured English, thus improving the naturalness of scenario presentation.

Finally the author believes that integrating basic research with user-centred system design, as reported in this paper, has been a surprisingly effective way of undertaking basic research. Within CREWS we advocate a gradual evolution of the research, methods and software tools throughout the project's duration, rather than developing revolutionary ideas and products so often found with the research-then-transfer approach (Potts 1993). The authors are also looking to make CREWS-SAVRE more domain-specific prior to its evaluation during case studies. Potts argues that "much of the research-then-transfer work that seeks to develop completely general tools is naive". Therefore, during the planned case studies, we anticipate incrementally adapting the CREWS-SAVRE tool to fit with organisations' requirements management tools and extending its data bases with domain-specific models and exception classes. If the case studies are successful they will demonstrate the importance of gradual evolution of research prototypes, as advocated by Potts (1993).

Acknowledgements

Part of this research was funded as part of the European Commission's ESPRIT III 6353 'NATURE' basic research action and 'CREWS' 21903 long-term research project. Special thanks goes to Alistair Sutcliffe, Shailey Minocha and Keith Manning.

References

Achour C.B., 1997, 'Linguistic Instruments for the Integration of Scenarios in Requirements Engineering', Third International Workshop on Requirements Engineering: Foundation for Software Quality, June 1997.

Achour C.B. & Rolland C., 1997, 'Introducing Genericity and Modularity of Textual Scenario Interpretation in the Context of Requirements Engineering', CREWS Technical Report, Centre de Recherche en Informatique, Universite de Paris 1, Paris, France.

Archer, Myla M. and Heitmeyer, Constance L., "Verifying Hybrid Systems Modeled as Timed Automata: A Case Study," Proceedings of HART '97, Grenoble, France, March 26-28, 1997.

Blyth A., 1996, 'Responsibility Modelling and its Application to the Specification of Domain Knowledge' , in 'Domain Knowledge for Interactive Systems Design', ed. A. Sutcliffe, D. Benyon & F. van Assche, Chapman & Hall, 48-60.

Buschmann F., Meunier R., Rohnert H., Sommerlad P., & Stal M., 1996, 'A System of Patterns: Pattern-Oriented Software Architecture', John Wiley.

Carroll J.M., 1995, 'The Scenario Perspective on Systems Development', in 'Scenario-Based Design: Envisioning Work and Technology in System Development', ed J.M. Carroll.

Coad P., North D. & Mayfield M., 1995, 'Object Models: Strategies, Patterns and Applications', Englewood Cliffs, Prentice-Hall.
Cockburn A., 1995, 'Structuring Use Cases with Goals', http://members.aol.com/acockburn/papers/usecase.htm.

Constantopoulos P., Jarke M., Mylopoulos J. & Vassiliou Y., 1991, 'Software Information Base: A Server for Reuse', Technical Report, FORTH Research Institute, Univ of Heraklion, Crete.

Dik S.C., 1989, 'The Theory of Functional Grammar, Part I: the Structure of the Clause, Functional Grammar Series, Vol 9, Foris Publications.

Dowell J. & 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.

Dubois P., Dubois E. & Zeippen J., 1997, 'On the Use of a Formal Representation', Proceedings of the 3rd International Symposium on Requirements Engineering, IEEE Computer Society Press, 128-137.

Feblowitz M., Greenspan S., Reubenstein H. & Walford R., 1996, 'ACME/PRIME: Requirements Acquisition for Process-Driven Systems', Proceedings 8th International Workshop on Software Specification and Design, IEEE Computer Society Press, 36-45.

Fickas S. & Nagarajan P., 1988, 'Critiquing Software Specifications', IEEE Software, November 1988, 37-47.

Fischer G. & Nakakoji K., 1991, 'Making Design Objects Relevant to the Task at Hand', Proceedings of AAAI'91, AAAI Press/MIT Press, 67-73.

Gick M.L. & Holyoak K.J., 1983, 'Schema Induction and Analogical Transfer', Cognitive Psychology 15, 1-38.

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

Graham I., 1996, 'Task Scripts, Use Cases and Scenarios in Object-Oriented Analysis', Object-Oriented Systems 3, 123-142.

Archer M. M. & Heitmeyer C. L., 1997, 'Verifying Hybrid Systems Modeled as Timed Automata: A Case Study," Proceedings of HART '97, Grenoble, France, March 26-28 1997

Heymans P. & Dubois E., 1997, 'Some Thoughts about the Animation of Formal Specifications written in the ALBERT II Language', CREWS Technical Report, Computer Science Department, University of Namur, Belgium.
Hollnagel E. & Kirwan B., 1996, 'Practical Insights from Studies of Operator Diagnosis', Proceedings 8th European Conference on Cognitive Ergonomics, EACE, 133-137.

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

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

Jarke M., Bubenko Y., Rolland C., Sutcliffe A.G. & 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.

Johnson W.L., Feather M.S. & Harris D.R., 1992, 'Representation and Presentation of Requirements Knowledge', IEEE Transactions on Software Engineering 18(10), 853-869.

Leveson N., 1995, 'Safeware: System Safety and Computers', Addison-Wesley.

Maiden N.A.M., Minocha S., Ryan M., Hutchings K. & Manning K., 1997a, 'A Cooperative Scenario-Based Approach to Acquisition and Validation of System Requirements: How Exceptions can Help!', Proceedings Workshop on Errors in Systems Development, University of Glasgow, Scotland, March 20-22, 1997.

Maiden N.A.M., Minocha, S., Manning, K. and Ryan, M., 1997b, 'A Software Tool for Scenario Generation and Use', Proceedings Third International Workshop on REFSQ, June 16-20, Barcelona, Spain.

Maiden N.A.M., Minocha S., Manning K. & Ryan M., 1998, 'SAVRE: Systematic Scenario Generation and Use', to appear in Proceedings 3rd International Conference of Requirements Engineering (ICRE), IEEE Computer Society Press, 1998.

Maiden N.A.M., Mistry P. & 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. & Sutcliffe A.G., 1996a, 'Analogical Retrieval in Reuse-Oriented Requirements Engineering', 1996, Software Engineering Journal 11(5), 281-292.

Maiden N.A.M. & Sutcliffe A.G., 1996b, 'Computational Mechanisms for Parallel Problem Decomposition During Requirements Engineering', Proceedings 8th International Workshop on Software Specification and Design, IEEE Computer Society Press, 159-163.

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

Moszkowski B., 1986, 'Executing Temporal Logic Programs', Cambridge University Press.

Maiden N.A.M. & Sutcliffe A.G., 1992, 'Exploiting Reusable Specifications Through Analogy', Communications of the ACM 34(5), April 1992, 55-64.

Mylopopoulos J., Borgida A., Jarke M. & Koubarakis M, 1990, 'Telos: Representing Knowledge about Information Systems', ACM Transactions on Office Information Systems 8(4), 325.

Nielsen J., 1993, 'Usability Engineering', Academic Press.

Norman D.A. 1988, 'The Psychology Of Everyday Things', Basic Books Inc.

Pohl K., 1996, 'PRO-ART: Enabling Requirements Pre-Traceability', Proceedings 2nd International Conference on Requirements Engineering, IEEE Computer Society Press.

Pohl K., Jarke M. & Weidenhaupt K., 1997, 'The Use of Scenarios during Systems Development - The Current State of Practice', submitted to ICRE.

Pohl K. & Haumer P., 1997, 'Modeling Contextual Information about Scenarios', 3rd International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'97), June 16-17 1997.

Potts C., 1995, 'Using Schematic Scenarios to Understand User Needs', Proceedings DIS95: Designing Interactive Systems, Ann Arbor, 247-256.

Potts C., 1993, 'Software Engineering Research Revisited', IEEE Software, 19-28.

Potts C., Takahashi K. & Anton A.I., 1994, 'Inquiry-Based Requirements Analysis', IEEE Software 11(2), 21-32.

Prieto-Diaz R., 1990, 'Domain Analysis: An Introduction', ACM SIGSOFT Software Engineering Notes 15(2), April 1990, 47-54.

Rasmussen J., Pejtersen A.M. & Goodstein L.P., 1994, 'Cognitive Systems Engineering', John Wiley & Sons.

Reason J.T., 1990, 'Human Error', Cambridge University Press.

Regnell B. Kimbler K. & Wesslen A., 1995, 'Improving the Use Case Driven Approach to Requirements Engineering, Proceedings 2nd IEEE Symposium on Requirements Engineering, IEEE Computer Society, 40-47.

Reubenstein H.B. & Waters R.C., 1991, 'The Requirements Apprentice: Automated Assistance for Requirements Acquisition', IEEE Transactions on Software Engineering 17(3), 226-240.

Riesbeck C.K. & Schank R.C., 1989, 'Inside Case-based Reasoning', Lawrence Erlbaum Associates, Hillsdale NJ.

Rosch E., 1983, 'Prototype Classification and Logical Classification: the Two Systems', New Trends in Conceptual Representation: Challenges to Piaget's Theory, edited K. Scholnick, Lawrence Erlbaum Associates, Hillsdale NJ.

Schank R.C., 1982, 'Dynamic Memory: A Theory of Reminding and Learning in Computers and People', Cambridge University Press.

Shneiderman B., 1992, 'Designing the User Interface: Strategies for Effective Human-Computer Interaction', Addison-Wesley.

Sommerville I., Rodden T., Sawyer P., Bentley R. & Twidale M., 1993, 'Integrating Ethnography into the Requirements Engineering Process', Proceedings 1st IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, 165-173.

Sutcliffe A.G. & Maiden N.A.M., in press, 'The Domain Theory for Requirements Engineering', to appear in IEEE Transactions in Software Engineering, 1998.

Sutcliffe A.G. & Maiden N.A.M., 1993, 'Bridging the Requirements Gap: Policies, Goals and Domains', Proceedings 7th International Workshop on System Specification and Design, IEEE Computer Society Press, 52-55.

UML, 1997, 'Unified Modelling Language: Method', Rational Corporation.

Wexelblat A., 1987, 'Report on Scenario Technology', MCC Technical Report STP-139-87, MCC, Austin Texas, 1987.