Supporting Scenario-based Requirements Engineering[i]

Alistair G. Sutcliffe, Neil A.M. Maiden, Shailey Minocha, and Darrel Manuel

Centre for HCI Design,
School of Informatics,
City University,
Northampton Square,
London EC1V 0HB, UK

a.g.sutcliffe@city.ac.uk

Tel: +44-171-477-8411
Fax: +44-171-477-8859

Abstract

Scenarios have been advocated as a means of improving requirements engineering yet few methods or tools exist to support scenario based RE. The paper reports a method and software assistant tool for scenario-based RE that integrates with use case approaches to object oriented development. The method and operation of the tool are illustrated with a financial system case study. Scenarios are used to represent paths of possible behaviour through a use case and these are investigated to elaborate requirements. The method commences by acquisition and modelling of a use case. The use case is then compared with a library of abstract models that represent different application classes. Each model is associated with a set of generic requirements for its class, hence, by identifying the class(es) to which the use case belongs, generic requirements can be reused. Scenario paths are automatically generated from use cases, then exception types are applied to normal event sequences to suggest possible abnormal events resulting from human error. Generic requirements are also attached to exceptions to suggest possible ways of dealing with human error and other types of system failure. Scenarios are validated by rule-based frames which detect problematic event patterns. The tool suggests appropriate generic requirements to deal with the problems encountered. The paper concludes with a review of related work and a discussion of the prospects for scenario based RE methods and tools.

1. Introduction

Several interpretations of scenarios have been proposed ranging from examples of behaviour drawn from use cases [29], descriptions of system usage to help understand socio-technical systems [30], and experience based narratives for requirements elicitation and validation [39], [50]. Scenarios have been advocated as an effective means of

[i] This research has been funded by the European Commission ESPRIT 21903 'CREWS' (Co-operative Requirements Engineering With Scenarios) long-term research project.

communicating between users and stakeholders and anchoring requirements analysis in real world experience [15]. Unfortunately scenarios are extremely labour-intensive to capture and document [20], [44]; furthermore, few concrete recommendations exist about how scenario-based requirements engineering (RE) should be practised, and even less tool support is available.

Scenarios often describe information at the instance or example level. This raises the question of how instance level information can be generalised into the models and specifications that are used in software engineering. Scenarios may be used to validate requirements, as 'test data' collected from the observable practice, against which the operation of a new system can be checked [40]. Alternatively, scenarios may be seen as pathways through a specification of system usage, and represented as animations and simulations of the new system [14]. This enables validation by inspection of the behaviour of the future system. This paper describes a method and tool support for scenario based requirements engineering that uses scenarios in the latter sense.

In industrial practice scenarios have been used as generic situations that can prompt reuse of design patterns [8], [48]. Reuse of knowledge during requirements engineering could potentially bring considerable benefits to developer productivity. Requirements reuse has been demonstrated in a domain specific context [31], however, we wish to extend this across domains following our earlier work on analogies between software engineering problems [51].

The paper is organised in four sections. First, previous research is reviewed, then in section 3 a method and software assistant tool for scenario based RE, CREWS-SAVRE (Scenarios for Acquisition and Validation of Requirements) is described and illustrated with financial dealing system case study. Finally, we discuss related work and future prospects for scenario based RE.

2. Previous Work

Few methods advise on how to use scenarios in the process of requirements analysis and validation. One of the exceptions is the Inquiry Cycle of Potts [39] which uses scenario scripts to identify obstacles or problems in a goal-oriented requirements analysis. Unfortunately, the Inquiry Cycle does not give detailed advice about how problems may be discovered in scenarios; furthermore, it leaves open to human judgement how system requirements are determined. This paper builds on the concepts of the Inquiry Cycle with the aim of providing more detailed advice about how scenarios can be used in requirements validation. In our previous work we proposed a scenario based requirements analysis method (SCRAM) that recommended a combination of concept demonstrators, scenarios and design rationale [50]. SCRAM employed scenarios scripts in a walkthrough method that validated design options for 'key points' in the script. Alternative designs were documented in design rationale and explained to users by demonstration of early prototypes. SCRAM proved useful for facilitating requirements elaboration once an early prototype was in place [48]; however, it gave only outline guidance for a scenario-based analysis.

Scenarios can be created as projections of future system usage, thereby helping to identify requirements; but this raises the question of how many scenarios are necessary to ensure sufficient requirements capture. In safety critical systems accurate foresight is a pressing problem, so taxonomies of events [24] and theories of human error [37], [42], have been used to investigate scenarios of future system use. In our previous research [49], we have extended the taxonomic approach to RE for safety critical systems but this has not been generalised to other applications.

Scenarios have been adopted in object oriented methods [29], [21], [10] as projected visions of interaction with a designed system. In this context, scenarios are defined as paths of interaction which may occur in a use case; however, object oriented methods do not make explicit reference to requirements per se. Instead, requirements are implicit within models such as use cases and class hierarchies. Failure to make requirements explicit can lead to disputes and errors in validation [17], [38]. Several requirements management tools have evolved to address the need for requirements tracability (e.g. Doors, Requisite Pro). Even though such tools could be integrated with systems that support object oriented methods (e.g. Rational Rose) at the syntactic level, this would be inadequate because the semantic relationships between requirements and object oriented models needs to be established. One motivation for our work is to establish such a bridge and develop a sound means of integrating RE and OO methods and tools.

Discovering requirements as dependencies between the system and its environment has been researched by Jackson [26] who pointed out that domains impose obligations on a required system. Jackson proposed generic models, called problem frames, of system-environment dependencies but events arising from human error and obligations of systems to users were not explicitly analysed. Modelling relationships and dependencies between people and systems has been investigated by Yu and Mylopoulos [54] and Chung [7]. Their i* framework of enterprise models facilitates investigation of relationships between requirements goals, agents and tasks; however, scenarios were not used explicitly in this approach. Methods are required to unlock the potential of scenario based RE; furthermore, the relationship between investigation based on examples and models on one hand and the systems and requirements imposed by the environment on the other, needs to be understood more clearly.

One problem with scenarios is that they are instances, i.e. specific examples of behaviour which means that reusing scenario based knowledge is difficult. A link to reusable designs might be provided if scenarios could be generalised and then linked to object-oriented analysis and design patterns [18]. Although authors of pattern libraries do describe contexts of use for their patterns [8], [19], they do not provide guidance about the extent of such contexts. Requirements reuse has been demonstrated by Lam et al. [30]; although, the scope of reuse was limited to one application domain of jet engine control systems. Many problems share a common abstraction [51], and this raises the possibility that if common abstractions in a new application domain could be discovered early in the RE process, then it may be possible to reuse generic requirements and link them to reusable designs. This could provide a conduit for reusing the wealth of software engineering knowledge that resides in reusable component libraries, as well as linking requirements to object-oriented solution patterns (e.g. [8], [19]). Building a bridge from requirements engineering to reusable designs is another motivation for the research reported in this paper.

Next we turn to a description of the method and tool support.

3. Method and Tool support for Scenario-based RE

The method of scenario-based RE is intended to be integrated with object oriented development (e.g. OOSE - [29]), hence use cases are employed to model the system functionality and behaviour. A separate requirements specification document is maintained to make requirements explicit and to capture the diversity of different types of requirements, many of which can not be located in use cases. Use cases and the requirements specification are developed iteratively as analysis progresses.

We define each scenario as “one sequence of events that is one possible pathway through a use case”. Hence many scenarios may be specified for one use case and each scenario represents an instance or example of events that could happen. Each scenario may describe both normal and abnormal behaviour. The method illustrated in Figure 1 uses scenarios to refine and validate system requirements. The stages of the method are as follows:

1. Elicit and document use case
In this stage use cases are elicited directly from users as histories of real world system usage or are created as visions of future system usage. The use case model is validated for correctness with respect to its syntax and semantics. The paper does not report this stage is in detail.

2. Analyse generic problems and requirements

A library of reusable, generic requirements attached to models of application classes is provided. A browsing tool matches the use case and input facts acquired from the designer to the appropriate generic application classes and then suggests high level generic requirements attached to the classes as design rationale ‘trade-offs’. Generic requirements are proposed at two levels: first, general routines for handling different event patterns, and secondly, requirements associated with application classes held in the repository. The former provide requirements that develop filters and validation processes as specified in event based software engineering methods (e.g. Jackson System Development [25]); while the latter provide more targeted design advice; for instance, transaction handling requirements for loans/hiring applications.

3. Generate scenarios

This step generates scenarios by walking through each possible event sequence in the use case, applying heuristics which suggest possible exceptions and errors that may occur at each step. This analysis helps the analyst elaborate pathways through the use case in two passes; first for normal behaviour and secondly for abnormal behaviour. Each pathway becomes a scenario. Scenario generation is supported by a tool which automatically identifies all possible pathways through the use case and requests the user to select the more probable error pathways.

4. Validate system requirements using scenarios

Generation is followed by tool-assisted validation which detects event patterns in scenarios and presents checklists of generic requirements that are appropriate for particular normal and abnormal event patterns. In this manner requirements are refined by an interactive dialogue between the software engineer and the tool. The outcome is a set of formatted use cases scenarios and requirements specifications which have been elaborated with reusable requirements.

Although the method appears to follows a linear sequence, in practice the stages are interleaved and iterative.

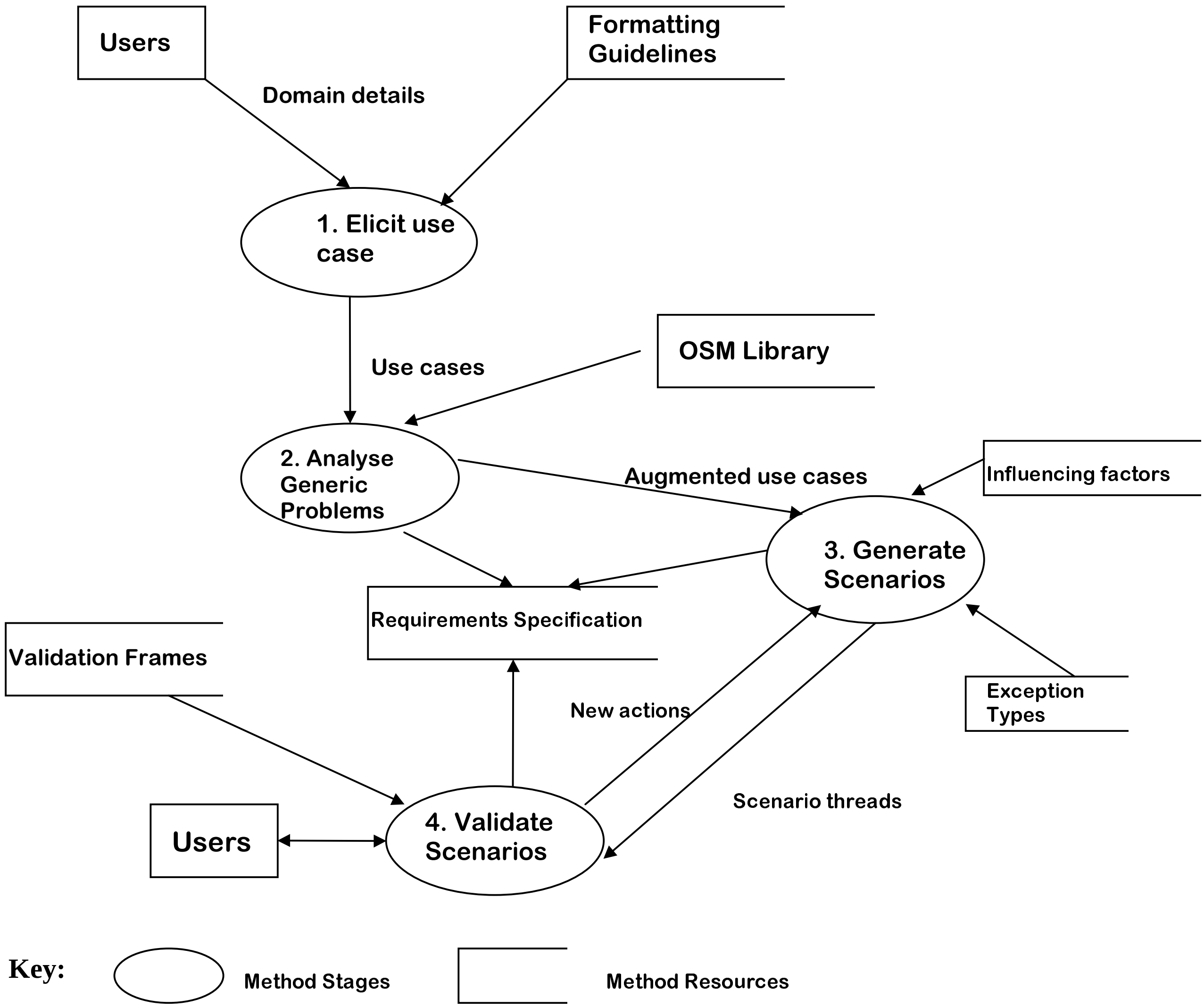


Figure 1 Method Stages for Scenario-based Requirements Engineering.

3.1. Schema for scenario based requirements modelling

In this section we describe the schema of scenario based knowledge shown in Figure 2. A use case is a collection of actions with rules that govern how actions are linked together, drawn from Allen's temporal semantics [2] and the calculus of the ALBERT II specification language [14]. An action is the central concept in both scenarios and use cases. The use case specifies a network of actions linked to the attainment of a goal which describes the purpose of the use case. Use cases are refined into lower levels of detail, so a complete analysis produces a layered collection of use cases. Use cases have user-defined properties that indicate the type of activity carried out, e.g. decision, transaction, control, etc. Each action can be sub typed as either cognitive (e.g. the buyer agrees the deal price on offer), physical (e.g. the dealer returns the telephone receiver), system-driven (e.g. the dealing systems stores information about the current deal) or communicative (e.g. the buyer requests a quote from the dealer). Each action has one start event and one end event. Actions are linked through 8 types of action-link rules which are described later in more detail.

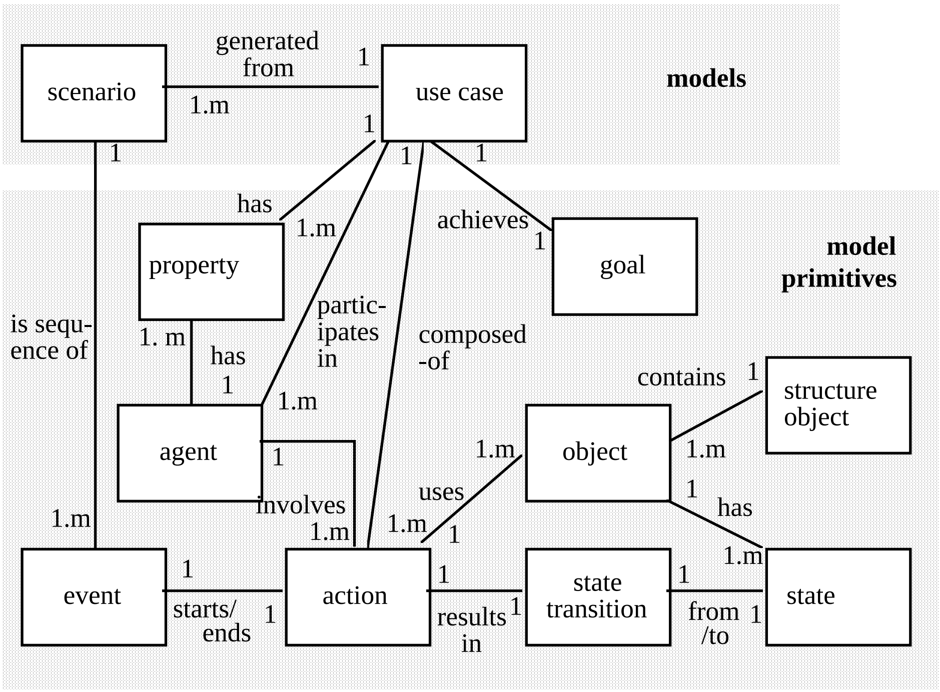


Figure 2 Meta-schema for modelling use cases and scenario based knowledge.

Each action involves one or more agents. Each agent can be either human (e.g. a dealer), machine (e.g. a dealing-system), composite (e.g. a dealing-room) or an unspecified type. Agents have user-defined properties that describe their knowledge and competencies. Action - agent links are specified by an involvement relation that can be subtyped as {performs, starts, ends, controls, responsible}. Each action uses nil, one or many object instances denoted by a use relation, subtyped as {accesses, reads, operates}. Each action can also result in state transitions which change the state of objects. States are aggregations of attribute values which characterise an object at a given time in both quantitative and qualitative terms. Structure objects are persistent real world objects that have spatial properties and model physical and logical units of organisation. Structure objects are important components in reusable generic domain models (called object system models and described in section 3.5), but also allow use case models to be extended to describe the location of objects, agents and their activity.

Introduction to the case study

The case study is based on a security dealing system at a major bank in London. Securities dealing systems buy and sell bonds and gilt-edged stock for clients of the bank. The bank's dealers also buy and sell securities on their own account to make a profit. Deals may be initiated either by a client requesting transactions, or by a counterparty (another bank or stockbroker who acts as a buyer) requesting a quotation, or by the dealer offering to buy or sell securities. Quotes are requested over the telephone, while offers are posted over electronic wire services to which other dealers and stockbrokers subscribe. In this case study we focus on deals initiated by a counterparty from the viewpoint of the dealer. The main activities in this use case include agreement between the dealer and buyer on the deal price and number of stock, entering and recording deal information in the computer system, and checking deal information entered by the dealer to validate the deal.

The case study contains several use cases; however, as space precludes a complete description, we will take one use case ' prepare-quote' as an example, as illustrated in figure 3. The sequence is initiated by a request event from the counterparty agent. The dealer responds by providing a quotation which the counterparty assesses. If it is suitable the counterparty agrees to the quotation and the deal is completed. Inbound events to the system are the deal which has to be recorded and updated, while outbound events are displays of market information and the recorded deal. System actions are added to model the first vision of how the dealing support system will work. The dealer agent is linked to the dealing room structure object that describes his location of work. The dealer carries out the "prepare quote" use case which is composed of several actions and involves the "trade" object.

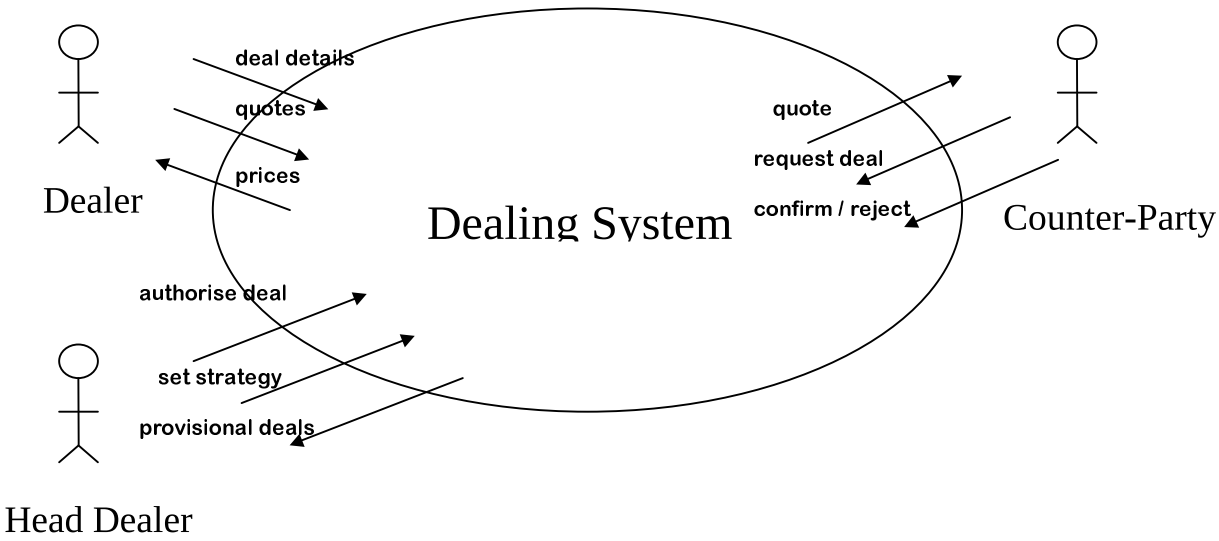


Figure 3 Upper level use case illustrated as an Agent-interaction showing tasks, agents and actions.

3.2. Tool support

The method is supported by Version 2.1 of the CREWS-SAVRE tool that 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 and computer-aided software engineering software tools. It supports 6 main functions which correspond to the architecture components shown in Figure 4.

1: incremental specification of use cases and high-level system requirements (the domain/use case modeller supports method stage 1)
2: automatic generation of scenarios from a use case (scenario generator supports stage 3);
3: manual description of use cases and scenarios from historical data of previous system use, as an alternative to tool-based automatic scenario generation (use case/scenario authoring component supports stage 1);
4: presentation of scenarios, supporting user-led walkthrough and validation of system requirements (scenario presenter supports stage 4);
5: semi-automatic validation of incomplete and incorrect system requirements using commonly occurring scenario event patterns (requirements validator supports stage 4). CREWS-SAVRE is loosely coupled with RequisitePro's requirements database so it can make inferences about the content, type and structure of the requirements.

Another component, which guides natural language authoring of use case specifications, is currently under development. The component uses libraries of sentence case patterns (e.g. [14]) to parse natural language input into semantic networks prior to transforming the action description into a CREWS-SAVRE use case specification. Space precludes further description of the component in this paper.

The CREWS-SAVRE tool permits the user to develop use cases which describe projected or historical system usage. It then uses an algorithm to generate a set of scenarios from a use case. Each scenario describes a sequence of normal or abnormal events specified in the original use case. The tool uses a set of validation frames to detect event patterns in scenarios, thereby providing semi-automatic critiquing with suggestions for requirements implied by the scenarios.

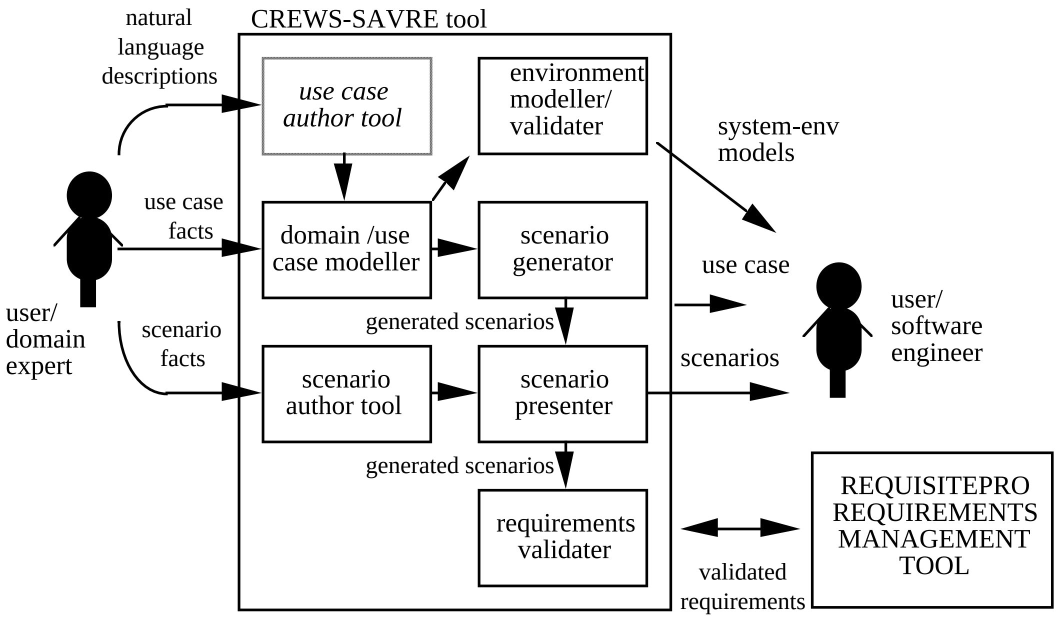


Figure 4 Overview of the CREWS-SAVRE tool architecture

3.3 Use Case Specification

The user specifies a use case using CREWS-SAVRE's domain and use case modeller components. First, for a domain, the software engineer specifies all actions in the domain, defines agents and objects linked to these actions, assigns types {e.g. communicative, physical, etc.} to each action, and specifies permissible action sequences using action-link rules. From this initial domain model, the user can then choose the subset of domain actions which form the normal course of a use case. This enables a user to specify more than one use case for a domain, and hence generate more scenarios for that domain. As a result, a use case acts as a container of domain information relevant to the current scenario analysis. As scenarios are used to validate different parts of the domain's future system, different use cases containing different action descriptions are specified. Consequently, the domain model enables simple reuse of action descriptions across use cases, because two or more use cases can include the same action.

Eight types of action-link rule are available in CREWS-SAVRE:

strict sequence (A then B): ev(endA)<ev(startB);
part sequence (A meanwhile B): (ev(startA)<ev(startB)) AND ((ev(endA)>(ev(startB)) AND ((ev(endA)<(ev(endB)) ;
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)).

where event ev(X) represents the point in time at which the event occurs. Actions are defined by ev(startA) < ev(endA) and ev(startB) < ev(endB), that is all actions have a duration and the start event of an action must occur before the end event for the same action. These link rules types build on basic research in temporal semantics [2] and the formal temporal semantics and calculus from a real-time temporal logic called ALBERT-CORE which underpins the ALBERT II specification language [23]. The current version of CREWS-SAVRE does not provide all of Allen's [2] 13 action-link types. Rather, it provides a set of useful and usable semantics based on practical reports of use case analysis (e.g. [1], [29]).

For the dealing system domain, two actions are *Buyer requests deal from the dealer* and *Dealer retrieves price information from the dealer-system*. Selection of the action-link rule MEANWHILE indicates that, in general, the dealer begins to retrieve price information only after the buyer begins to request the deal. A third action, this time describing system behaviour- *Dealer-system displays the price information to the dealer*- can be linked through the THEN rule to specify a strict sequence between the two actions, in that the first action must end before the second action starts. Part of the dealing system domain model is shown in Figure 5. It shows a subset of the current domain actions (shown as (A) in Figure 5), the specification of attributes of a new action (B), current action-link rules (C) and agent types (D). Other parts of the domain modeller are outside of the scope of this description.

One or more use cases are specified for a domain such as the dealing system. Each use case is linked to one high-level requirement statement, and each system action to one or more system requirements. Each use case specification is in 4 parts. The first part specifies the identifiers of all actions in the use case. The second part specifies action-link rules linking these actions. The third part contains the object-mapping rules needed to handle use of synonyms in action descriptions. The fourth part specifies exception-types linked to the use case, as described in section 3.6.

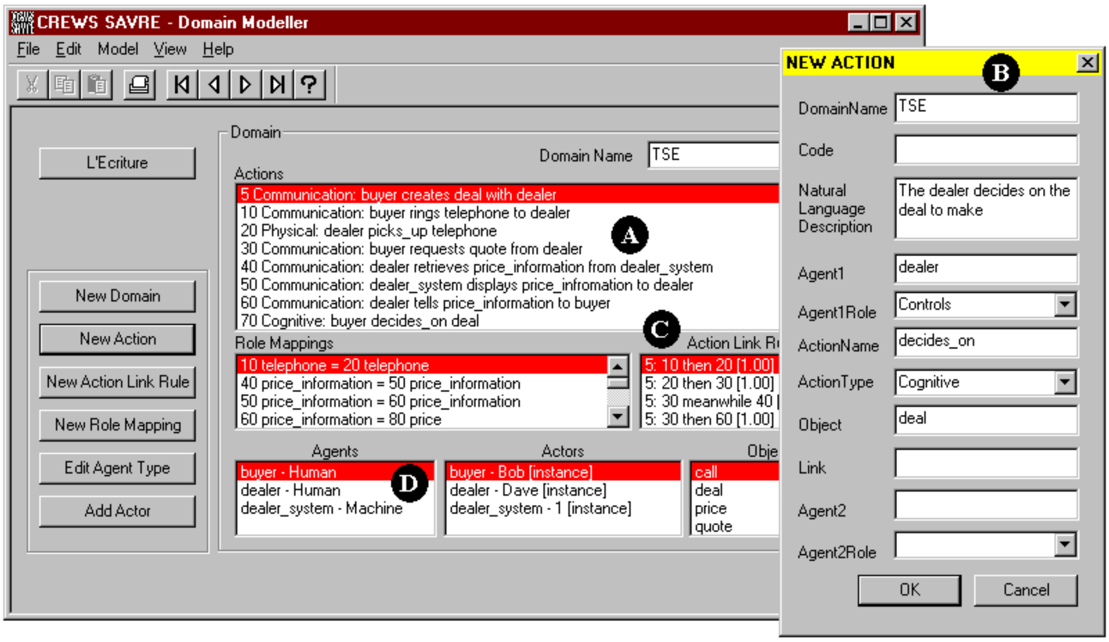


Figure 5 Domain Modeller screen dump.

3.4 Checking the use case specification

Validation rules check the integrity of the relationships specified in section 3.1, represented in tuple format < model component1, relationship-type, model component2>. The tool checks the integrity of use case models to ensure they conform to the schema, but in addition, the tool is parameterised so it can check use cases in a variety of ways. This enables user defined validation of properties not defined in the schema. Validation checks are carried out by clusters of rules, called validation-frames which are composed of two parts. First a situation that detects a structural pattern (i.e. a combination of components connected by a relationship) in the use case. The second part contains requirements that should be present in the part of the model detected by the situation. Frames are used for validating the consistency of use case models against the schema, and for detecting potential problems in use cases as detailed in section 3.5. In the former case the frames detect inconsistencies in a use case, the latter case frames detect event patterns in scenarios and suggest appropriate, generic requirements. Two examples of consistency checking frames are as follows:

(i) Checks for agents connected by a specific relationship type.

validation-frame {detects dyadic component relationships}

situation:

model(component(x), relationship (y), component (z));

schema requirements:

(component type (i) component (x), mandatory) ;
(component type (j) component (y), mandatory);
(relationship type (k) relationship (z), mandatory)

end-validation-frame

Example: The user decides to check that all actions are controlled by at least one agent. First the tool finds all nodes with a component type = agent, then it finds all the nodes connected to that agent which are actions and finally whether the involvement relationship that connects the two nodes is of the correct type = control. As the tool is configurable the input parameters, (i) and (j) can be any schema primitives, and the relationship (k) is defined in the schema. The tool detects untyped or incorrect relationships using this 'structure matching' algorithm. In the ‘prepare quotes’ use case (see figure 3), this validation check should detect that the dealer controls the give-quote action.

(ii) Validates whether two components participating in a relationship have specific properties.

validation-frame {detects components in a relationship have the correct properties}

situation:

model(component(x), property(w), relationship (z), component (y),
property (v));

schema requirements:

(component type (i) component (x), mandatory);
(component type (j) component (y), mandatory);
(relationship type (k) relationship (z), not specified);
(component (x) property (w), mandatory);
(component (y) property (v), mandatory);

end-validation-frame

Example: The user wishes to validate that all agents that are linked to a use case with a <decision> property have an <authority> property. The properties (v,w) to be tested are entered in a dialogue, so any combination may be tested. In this case the tool searches for nodes with a type = agent, and then tests all the relationships connected to the agent node. If any of these relationships is connected to a node type = use case, then the tool reads the property list of the use case and the agent. If the agent does not have an 'authority' property and the use case has a 'decision' property then warning is issued to the user. As properties are not sub types, this is a lexical search of the property list. This frame will test that the dealer has authority for the use case ‘evaluate choice’ - which has a decision property.

The system is configurable so any combinations of type can be checked for any set of nodes and arcs which conform with the schema described in figure 2. Our motivation is to create requirements advice that evolves with increasing knowledge of the domain, so the user can impose constraints beyond those specified in the schema, and use to tool to validate that the use case conforms to those constraints.

3.5 Using Application classes to identify Generic requirements

This stage takes the first cut use case(s) and maps them to their corresponding abstract application classes. This essentially associates use cases describing a new system with related systems that share the same abstraction. A library of application classes, termed Object system models (OSMs) has been developed and validated in our previous work on computational models of analogical structure matching [34].

Object system models (OSMs) are organised in 11 families and describe a wide variety of applications classes. The families that map to the case study application are object supply (inventory control), accounting object transfer (financial systems), object logistics (messaging), object sensing (for monitoring applications), and object-agent control (command and control systems). Each OSM is composed of a set of co-operating objects, agents and actions that achieve a goal, so they facilitate reuse at the system level rather than as isolated generic classes. Taking object supply (see Appendix A) as an example, this OSM models the general problem of handling the transaction between buyers and suppliers. In this case study, this matches to the purchase of securities which are supplied by the bank to the counter party who acts as the buyer. The supplier giving a price maps to the dealer preparing a quotation. Essentially OSMs are patterns for requirements specification rather than design solutions as proposed by [19]. Each OSM represents a transaction or a co-operating set of objects and are modelled as class diagrams, while the agent's behaviour is represented in a use case annotated with high level generic requirements, expressed in design rationale diagrams.

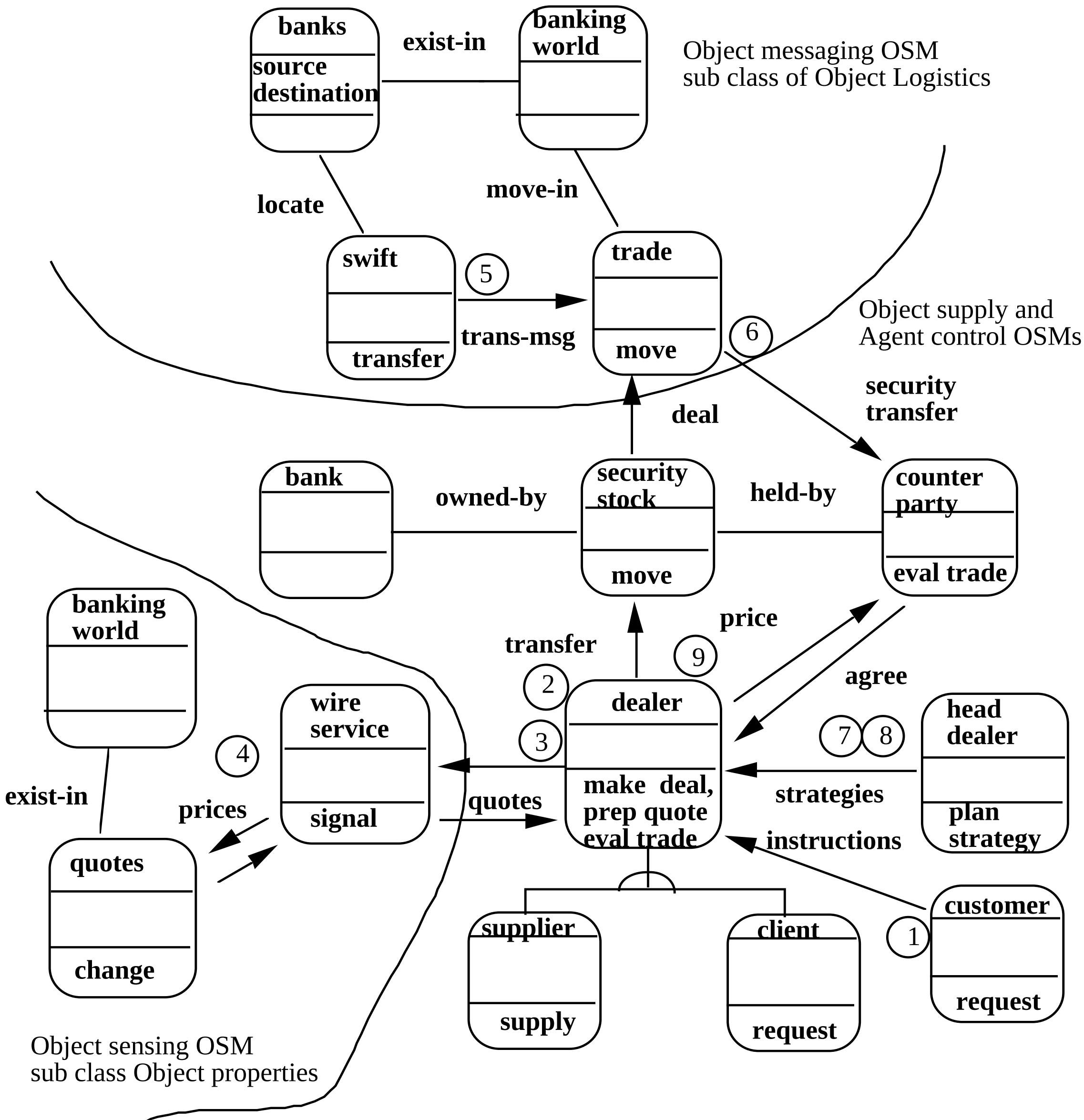
A separate set of generic use case models are provided for the functional or task-related aspects of applications. Generic use cases cause state changes to objects or agents; for instance, a diagnostic use case changes the state of a malfunctioning agent (e.g. human patient) from having an unknown cause to being understood (symptom diagnosed). Generic use cases are also organised in class hierarchies and are specialised into specific use cases that are associated with applications. Currently seven families of generic use cases have been described; diagnosis, information searching, reservation, matching-allocation, scheduling, planning, analysis and modelling. An example of a generic use case is information searching which is composed of subgoals for: articulating a need, formulating queries, evaluating results, revising queries as necessary. In the dealing domain this maps to searching for background information on companies in various databases, evaluating company profit and loss figures and press releases, then refining queries to narrow the search to specific companies of interest. For a longer description of the generic use cases and their associated design rationale see [53].

Specific use cases in a new application may be associated with object system models either by browsing the OSM/use case hierarchy and selecting appropriate models, or by applying the identification heuristics - see Appendix A, or by using a semi-automated matching tool that retrieves appropriate models given a small set of facts describing the new application [51]. These heuristics point towards OSM models associated with the application; however, identification of appropriate abstractions is complex and a complete description is beyond the scope of this paper. In this stage, mapping between use case components and their corresponding abstractions in the OSMs are identified so that generic requirements, attached to the OSMs, can be applied to the new application. Unfortunately, the mapping of problems to solutions is rarely one to one, so trade -offs have to be considered to evaluate the merits of different solutions. Design rationale [11] provides a representation for considering alternative designs that may be applied to the requirements problems raised by each OSM. Non-functional requirements are presented as criteria by which trade offs may be judged. The software engineer judges which generic requirements should be recruited to the requirements specification and may adds further actions to the use case thereby elaborating the specification.

Case study

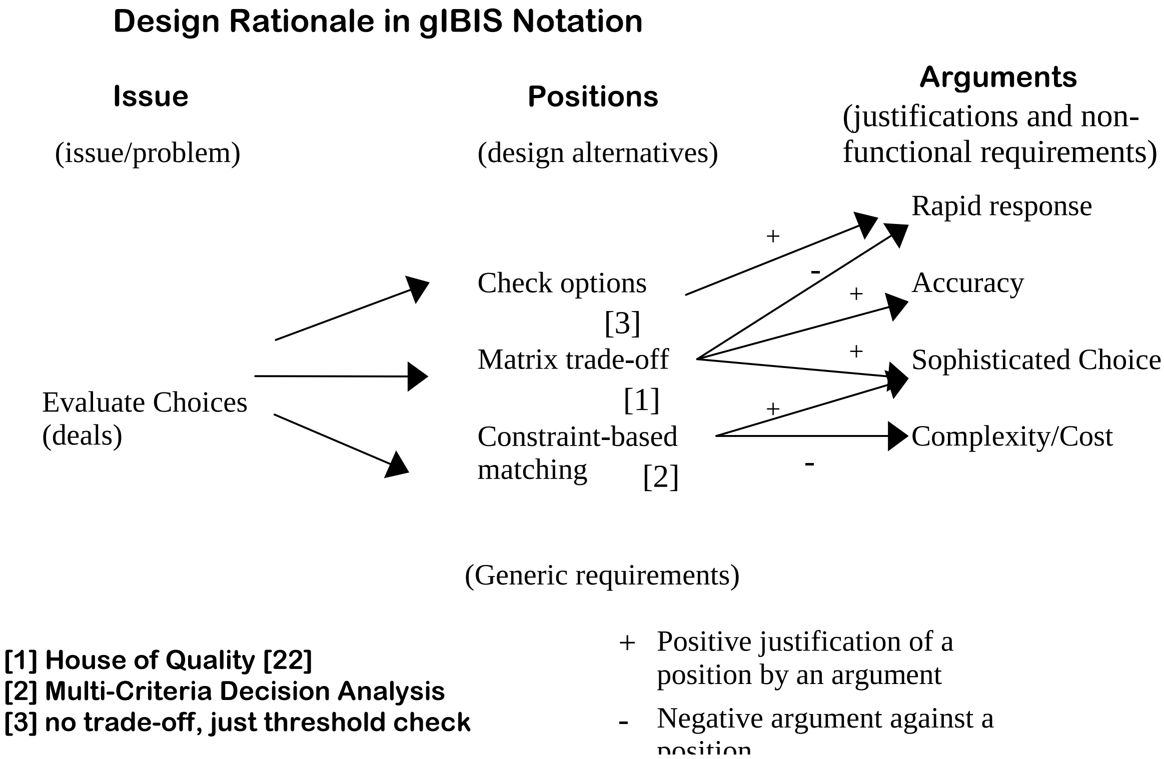
The security trading system involves five OSMs; Object Supply that models securities trading; Account Object Transfer which models the settlement part of the system (payment for securities which have been purchased) and Object messaging to describe the communication between the dealer and counterparties. Other OSMs model sub systems that support trading, such as Object Sensing that detects changes in security prices, markets and the dealer's position, and Agent Control which describes the relationship between the head dealer and the other dealers. In addition to the OSMs, the dealing system contains generic use cases (evaluate purchase and plan strategy) that describe the dealer's decision making and reasoning. These map the domain specific use cases of evaluating a deal that has been proposed and for planning a trading strategy. A further specific use case 'prepare quote' is mapped to the generic use case (price item) associated with the Object Supply OSM. A generic model of the security trading system, expressed as an aggregation of OSMs, is given in figure 6.

The settlement part of the system (Accounting object transfer OSM) has been omitted. The OSM objects have been instantiated as dealing system components. Clusters of generic requirements, represented as design rationale are associated with appropriate OSM components.



Generic requirements clusters (GRs in Design rationales)
1. Checking customer preferences (limits on dealing)
2. Calculating prices (prepare quotations)
3. Evaluating choices (deals)
4. Sampling changes to object properties (stock prices)
5. Message transmission protocols (deal notification)
6. Message encryption (deal security)
7. Communicating commands (head dealer strategy)
8. Reporting compliance (strategy obeyed/followed)
9. Calculating replenishment

Figure 6 Aggregation of OSMs that match to the security dealing system, represented in object oriented analysis notation [9].



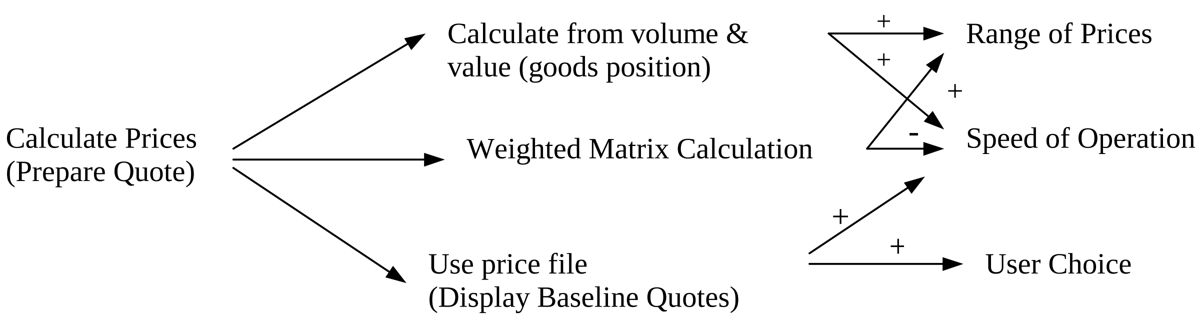


Figure 7 Design rationale for high level generic requirements from clusters 2 and 3 in Figure 6 attached to 'evaluate choice' and 'prepare quote' use cases. The instantiated requirement derived from the generic version is given in brackets.

The functional requirements that could be applied to support two use cases 'evaluate deal' and 'prepare quote'. For evaluate deal the rationale is taken from the generic use case 'evaluate choice', which is a sub-class in the matching/allocation family. This proposes three options: to assess the purchase against a set of reference levels, to prioritise several purchase options by a simple House of Quality style matrix [22], and finally to use a sophisticated multi-criteria decision making algorithm. Hypertext links from the rationale point to reusable algorithms for each purpose. The options for 'prepare quotes' are to automate quotation with simple calculations based on the dealer's position, the desirability of the stock and the market movement; or to choose a weighted matrix technique for quoting according to the volume requested and the dealer's position, or to leave quotation as a manual action with a simple display of the bank's baseline quotations. Since the first two options may be too time consuming for the dealers, the third was chosen. For evaluate deal, the simple calculation is taken as optional facility, leaving the dealer in control. Two high level generic requirements are added to the requirements specification and actions to the use case which elaborates the system functionality.

3.6 Scenario Generation

This stage generates one or more scenarios from a use case specification. Each scenario is treated as one specific ordering of events. The ordering is dependent on the timings of start- and end- events for each action and the link rules specified in the originating use case. Entering timings is optional, so in the absence of timed events, the algorithm uses the ordering inherent in the link rules. More than one scenario can be generated from a single use case if the action-link rules are not all a strict sequence (i.e. A then B).

The space of possible event sequences is large, even within a relatively simple use case. The scenario generation algorithm reduces this space by first determining the legal sequences of actions. The space of permissible event sequences is reduced through application of action-link and user-constraints. The user can enter constraints that specify which agents and actions should be considered in the scenario generation process, thus restricting the permissible event sequences to sequences (es) to include an event (ev) that starts the action (A) and involves a predefined agent (ag) and which has at least a given probability (p):

UC: (ev(startA) in es) AND (ev starts A) AND (A involves ag) AND (probability(A)$\geq$p))

For example, each generated scenario must include the event that starts action 20, it must involve the agent "dealer" and action 20 must have at least a 10% likelihood of occurrence according to probabilities calculated from information in the use case specification.

The 'prepare quote' use case definition leads to the 3 possible scenarios shown in figure 8. The difference between each is the timing of event E40 which ends action 40, and whether action 40 or action 45 occurs. This depiction ignores the application of the constraint on the likelihood of an event sequence occurring. Scenario-1 and -2 differ in the timing of event E40 (end of request for price information from the dealer-system) while scenario-3 describes a different event sequence when the dealer is unable to offer a quote for the deal.

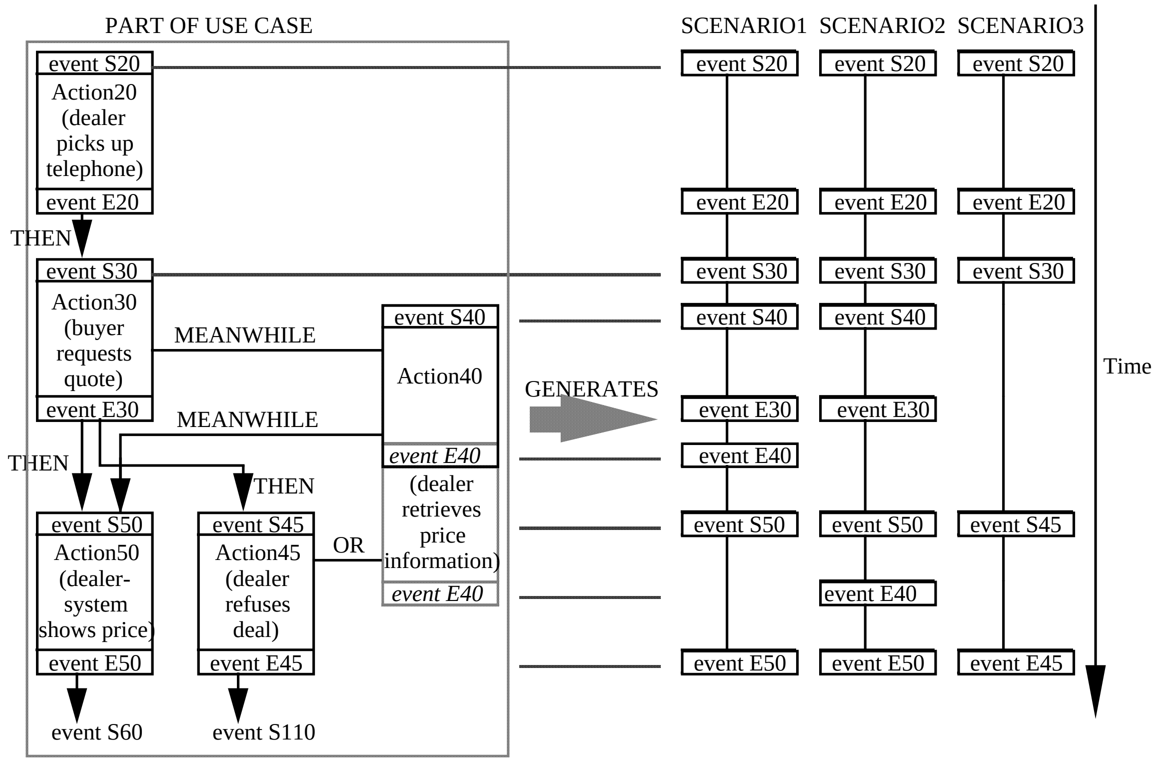


Figure 8 Diagram illustrating 3 normal scenario paths generated from a use case fragment

The generation mechanism is in two stages. First it generates each permissible normal course scenario from actions and link rules in the use case, then it identifies alternative paths for each normal sequence from exception types, as summarised in Table I. They are divided into two groups. First abnormal events drawn from Hollnagel's [24] event 'phenotypes' classification and secondly, information abnormalities that refer to the message contents(e.g. the information is incorrect, out-of-date etc.) that follow validation concepts proposed by Jackson [27]. Each exception is associated with one or more generic requirements that propose high level solutions to the problem. The exception types are presented as "what-if" questions so the software engineer can choose the more probable and appropriate alternative path at each action step in the scenario.

Table I Summary of Exception types for events originating in the system and generic requirements to deal with abnormal patterns.

Exception	**Generic Requirement**
event does not happen - omitted	time-out, request resend, set default
event happens twice (not iteration)	discard extra event, diagnose duplicate
event happens in wrong order	buffer and process, too early - halt and wait, too late - send reminder, check task
event not expected	validate vs. event set, discard invalid event
information - incorrect type	request resend, prompt correct type
incorrect information values	check vs. type, request resend, prompt with diagnosis
information too late (out of date)	check data integrity, date/time check, use default
information too detailed	apply filters, post process to sort/group
information too general	request detail, add detail from alternative source

A set of rules constrain the generation of alternative courses in a scenario using action and agent types. These rules are part of an extensible set which can be augmented using the agent types and influencing factors described below. Two example rules are:

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

which ensures that only human exception types and influencing factors (see next section) are applied to a cognitive action for event (ev1), action (ac1) and exception type (ex), and

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 (ex(type=machine-machine-communication) applies-to ev1).

which ensures that only machine-machine communication failures are applied to communication actions where both agents are of type 'machine' for event (ev1), action (ac1), agents (ag1 and ag2) and exception type (ex). The rules identify particular types of failure that may occur in different agent-type combinations so that generic requirements can be proposed to remedy such problems. For instance, in the first rule, human cognitive errors that apply to action1 can be counteracted by improved training or aid memoir facilities (e.g. checklists, help) in the system. In the second rule which detects network communication errors, generic requirements are suggested for fault tolerant designs sand back-up communications.

The algorithm generates a set of possible alternative paths according to the agent-action combination. The use case modeller allows the user to select the more probable abnormal pathways according to their knowledge of the domain. To help the software engineer anticipate when exceptions may occur and assign probabilities to abnormal events, a set of influencing factors are proposed. These describe the necessary preconditions for an event exception to happen and are sub divided into 5 groups according to the agents involved:

- human agents: Influencing factors that give rise to user errors and exceptions are derived from cognitive science research on human error [42], Norman's model of slips [37] and Rasmussen's three levels of human-task mismatches [41]. However, as human error cannot be adequately described by only cognitive factors; we have included other performance affecting properties such as motivation, sickness, fatigue, and age; based on our previous research on safety critical systems [49].

- machine agents: failures caused by hardware and software, e.g. power supply problems, software crashes, etc.

- human-machine interaction: poor user interface design can lead to exceptions in input/output operations. This group draws on taxonomies of interaction failures from human-computer interaction [46] and consequences of poor user interface design (e.g. [36]);

- human-human communication: scenarios often involve more than one human agent. Communication breakdowns between people have important consequences. Exceptions have been derived from theories from computer-supported collaborative work [46]. Examples include communication breakdowns and misunderstandings;

- machine-machine communication: scenarios often involve machine agents, and exceptions specific to their communication can also give rise to alternative paths.

The interaction between influencing factors that give rise to human error is described in figure 9. Four outer groups of factors (working conditions, management, task/domain and personnel qualities) effect four inner factors (fatigue, stress, workload and motivations). These in turn effect the probability of human error which is manifest as an event exception of type <human-machine action or human action>. Human error can be caused by environment factors and qualities of the design, so two further outer groups are added. Personnel/user qualities are causal influences on human operational error, whereas the system properties can either be treated as causal explanations for errors or viewed as generic user interface requirements to prevent such errors. Requirements to deal with problem posed by influencing factors are derived from several sources in the literature, e.g. for task design and training [3], workplace ergonomics [45] and for Human Computer Interface design [46], [47] and standards (e.g. ISO 9241 [25]).

Ultimately, modelling event causality is complex, moreover, the effort may not be warranted for non-safety critical system, so three approaches are offered. First is to use the influencing factors as a paper-based 'tool for thought'. Second, the factors are implemented as a hypertext that can be traversed to explore contextual issues that may lead to errors and hence to generic requirements to deal with such problems. However, many of these variables interact, e.g. high stress increases fatigue. Finally as many combinations of influencing factors are possible and each domain requires a particular model, hence we provide a general modelling tool that can be instantiated with domain specific information. The tool allows influencing factors to be entered as a rating on a five point scale (e.g. high task complexity = 5, low complexity = 1) and then calculates the event probability from the ratings. The combination of factors and ratings are user controlled. he factors described in figure 9 may be entered into the tool with simple weightings to perform sensitivity analyses. A set of default formulae for inter-factor weights are provided, but the choice depends on the user's knowledge of the domain. The tool can indicate that errors are more probable given a used defined subset of influencing factors, but the type of exception is difficult to predict, i.e. a mistake may be more likely but whether this is manifest as a event being omitted or in the wrong order is unpredictable. Where more reliable predictions can be made new 'alternative path' rules (see above) are added to the system.
The tool is configurable so more validation rules can be added so the system can evolve with increasing knowledge of the domain. The current rules provide a baseline set that recommend generic requirements for certain types of agent e.g. untrained novices need context sensitive help and undo facilities, whereas experts require short cuts and ability to build macros. The influencing factors may be used as agent and use case properties and validated using the frames described in section 3.4.

3.7 Scenario Validation

CREWS-SAVRE is loosely-coupled with Rational's RequisitePro requirements management tool to enable scenario-based validation of requirements stored in RequisitePro's data base. CREWS-SAVRE either presents each scenario to the user alongside the requirements document, to enable user-led walkthrough and validation of system requirements, or it enables semi-automatic validation of requirements through the application of pattern matching algorithms to each scenario. Each approach is examined in turn.

Figure 10 shows a user-led walkthrough of part of one scenario for the dealing system use case, and the RequisitePro requirements document being validated. The left-hand side of the screen shows a normal course scenario as a sequence of events. On the right-hand side are alternative courses and generic exceptions generated automatically from the requirements engineer's earlier selection of exception types. For each selected event the tool advises the requirements engineer to decide whether each alternative course is (a) relevant, and (b) handled in the requirements specification. If the user decides that the alternative course is relevant but not handled in the requirements specification, s/he can retrieve from CREWS-SAVRE one or more candidate generic requirements to instantiate and add to the requirements document. Each exception type in CREWS-SAVRE's data base is linked to one or more generic requirements which describe solutions to mitigate or avoid the exception. Thus, CREWS-SAVRE provides specific advice during user-led scenario walkthroughs.

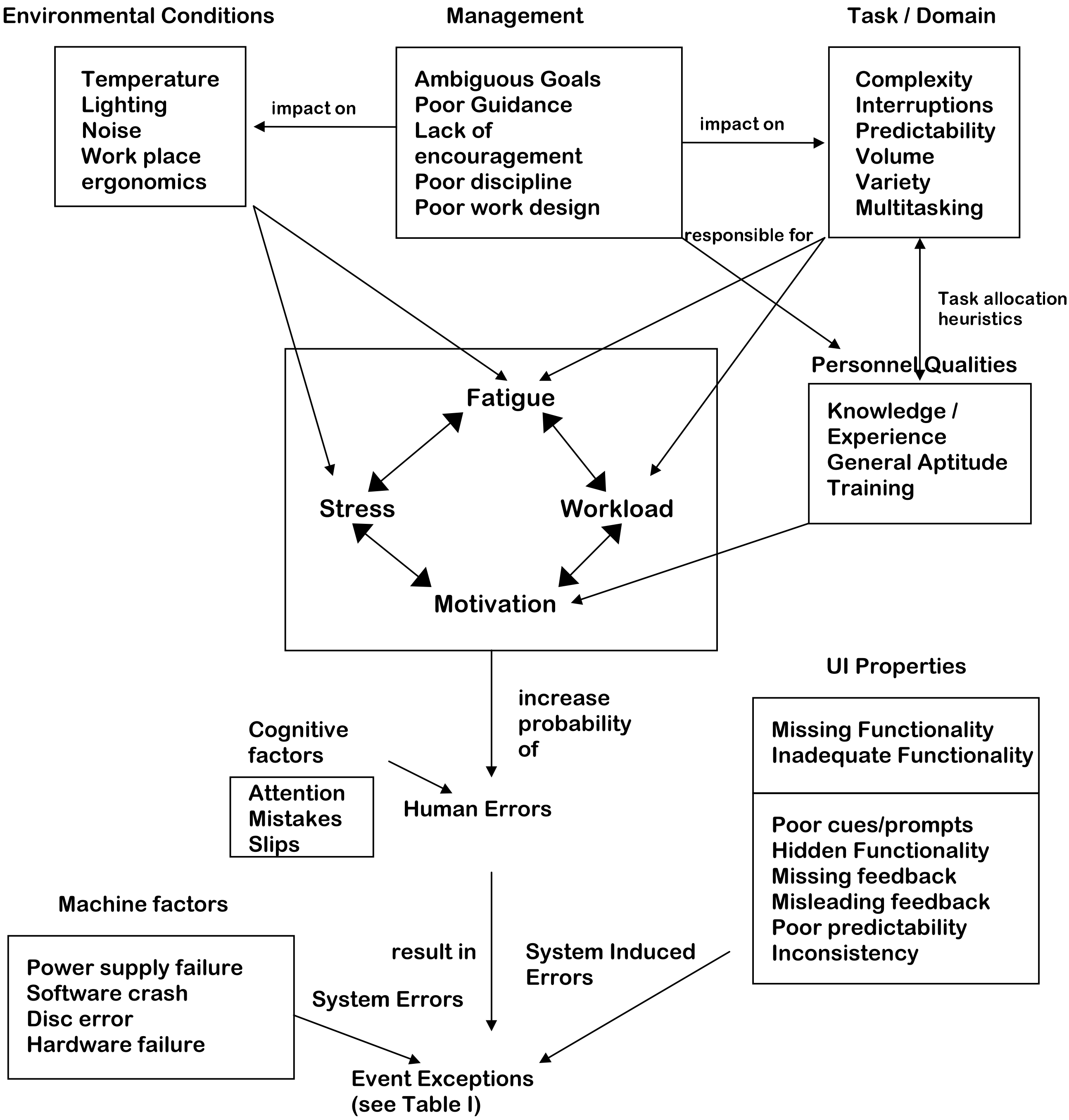


Figure 9 Influencing Factors for Exceptions and their interrelationships.

Consider the example shown in Figure 10. The user is exploring normal course event 90, the start of the communication action 'the dealer enters deal information into the dealer-system' ,(shown in (A) in Figure 10s), and the alternative course GAc6, the information is suspect (B). The user, having browsed candidate generic requirements (C), copies and pastes the requirement 'the system shall cross-reference the information with other information sources to ensure its integrity' into RequisitePro's requirements document (D). This figure also shows the current hierarchical structure of the requirements held in RequisitePro's data base (E).

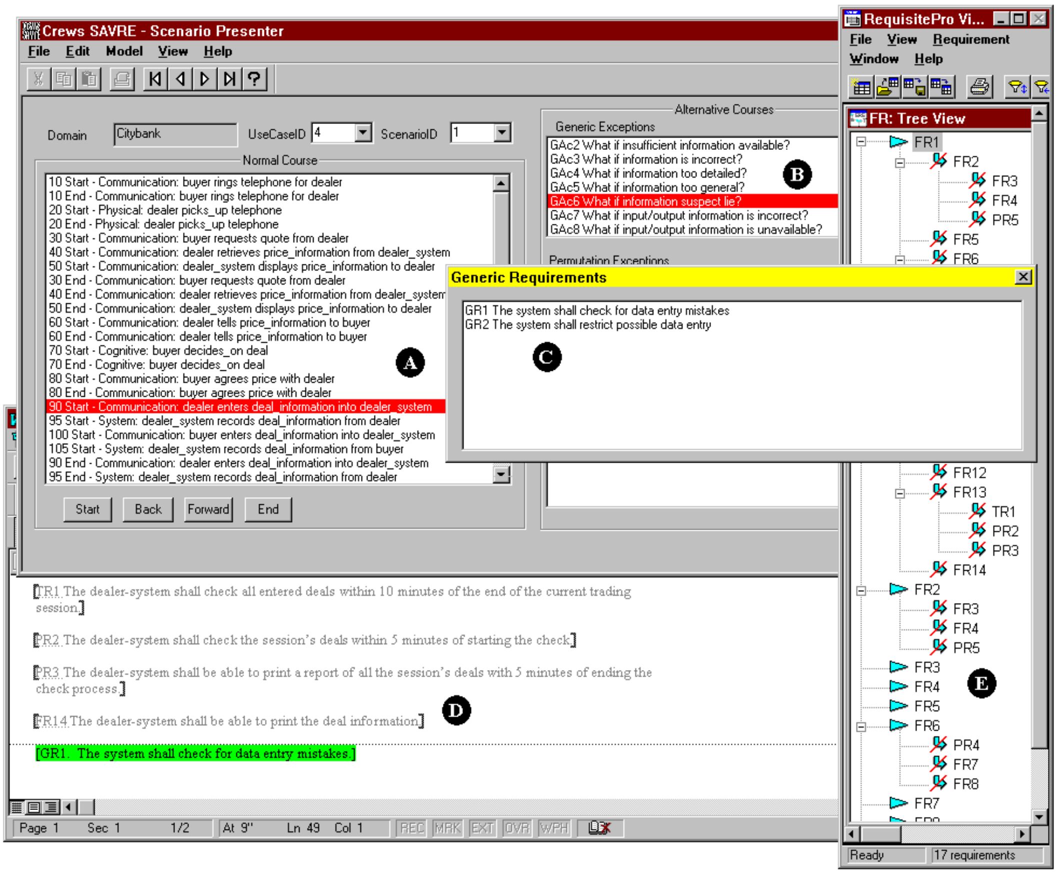


Figure 10 Validation Frames for Adding Generic Requirements

The second approach automatically cross-checks a requirements document and a scenario using a collection of patterns which encapsulate 'good' socio-technical system design and requirements specification. To operationalise this, the CREWS-SAVRE tool applies one or more validation frames to each event or event pattern in a user-selected scenario to determine missing or incorrect system requirements. Each validation frames specifies a pattern of actions, events and system requirements that extend KAOS goal patterns [12] describing the impact of different goal types on the set of possible system behaviours. A validation frame has two parts. The first defines the situation, that is, the pattern of events and actions expressed in the form <identifier, action-type, agent-types> where agent-types are involved in the action. Each event is expressed as <identifier, event-type, action-identifier> where event type defines whether the event starts or ends the action. The second part of the frame defines generic requirements needed to handle the event/action pattern. The frames start from the PS055 standard [35] and type each requirement as a functional, performance, usability, interface, operational, timing, resource, verification, acceptance testing, documentation, security, portability, quality, reliability, maintainability or safety requirement. Hence, automatic requirements-scenario cross-checking is possible using patterns of event, agent and action types in the scenario and requirement types in the requirements document.

An example validation frame is:

validation-frame {detect periods of system inactivity}
    situation:
        event(evA,_,acA) AND
        event(evB,_;acB) AND
        event(evA)≠event(evB) AND
        action(acA,_,agC) AND
        action(acB,_,agC) AND
        agent(agC,machine) AND
        not consecutive(agC,evA,evB);
    requirements:
        requirement(performance, optional, link);
        requirement(function <time-out/re-send>, optional, link);
end-validation-frame

This frame detects the absence of a reply event after a set time period from a human agent (implicitly) and signals the requirement to ask for resend or set a time out. An instantiation of this is the requirement to request a price to be entered by the dealer within 30 seconds of accessing the ‘prepare quote’ option. This exception deals with inbound event delays, so if the time is longer than a preset limit (i.e. the system is not used for that period), the tool recommends reusable generic requirements, for example to warn the user and log-out the user after a certain period of time.

Validation frames for alternative course events provide generic requirement to handle an event exception (ev) linked to an event 'ev' as follows:

validation-frame

```
        situation:
                event(evA,_,acA) AND
                action(acA,_,agA) AND
                agent(agA,machine) AND
                exception(acA,human-attention);
        requirements:
                requirement(functional, mandatory, link),
                generic-requirement(GR1, system shall check for data entry mistakes),
                generic-requirement(GR2, system shall restrict possible data entry);
end-validation-frame
```

Such attention failures are possible when entering dealer information. The tool proposes generic requirements, and the requirements engineer chooses requirement GR1 "the system shall check for data entry mistakes". In Figure 11, the requirements engineer is again examining event S90 [the dealer enters the deal information into the dealer-system] (A) and the causal influencing factor [agent pays poor attention to detail] (B). Again the above validation frame detects the need for functional requirements to handle the alternatives linked to event S90. As a result, the user is able to add new requirements to the requirements document to handle this alternative course.

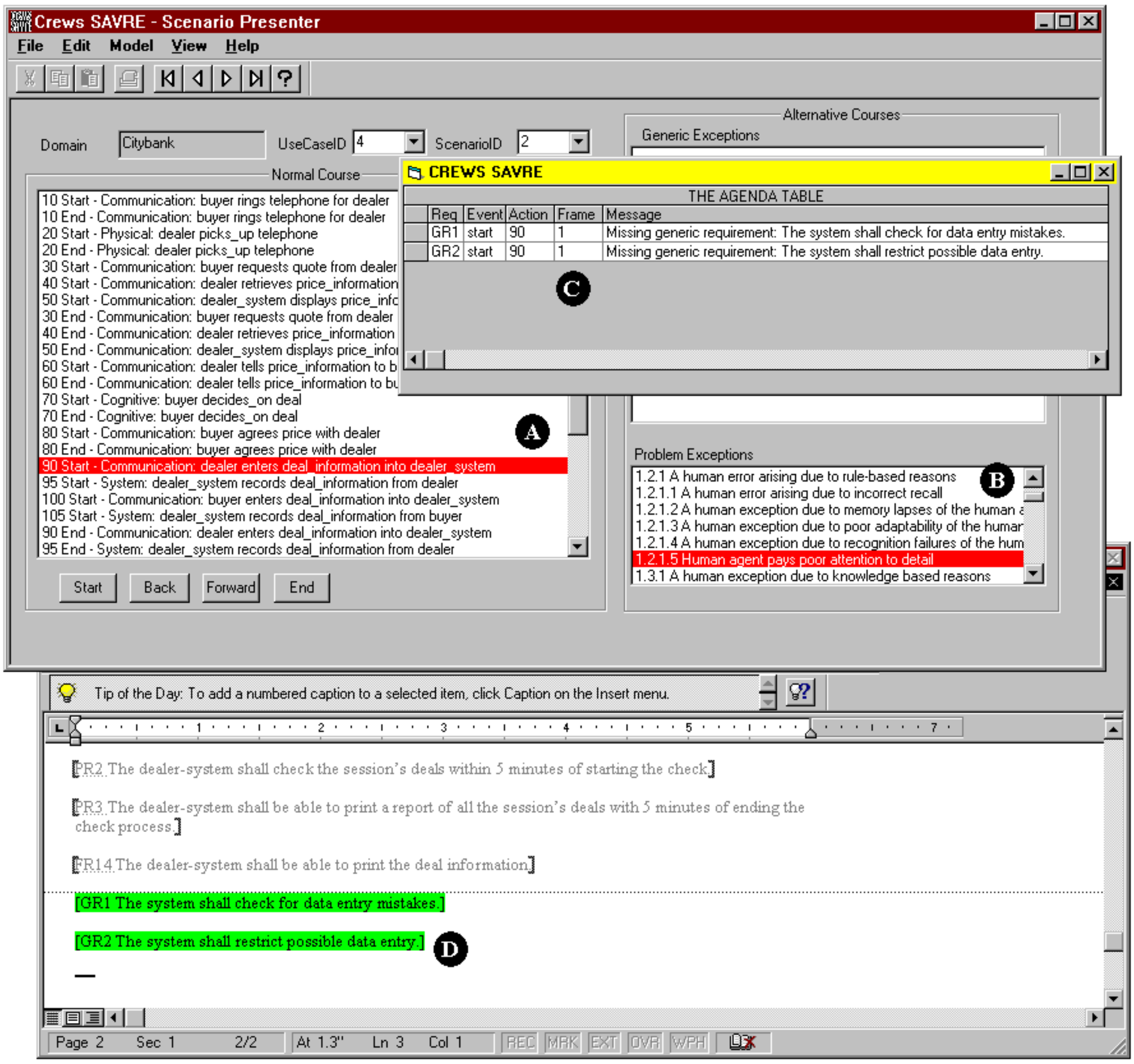


Figure 11 Validation Frames for Exceptions

Finally the tool uses validation frames for applying OSM-specific generic requirements in a similar manner. In section 3.4, high level generic requirements were recruited as design rationale trade offs, whereas when frames are used to detect OSM specific patterns in scenarios, more detailed requirements can be indicated. The computerised dealing system is an instantiation of the object supplying, object messaging and agent-object control system models. Consider a validation frame linked to the 'send' communication action in the object messaging system model. The situation (scenario) specifies an event which starts a communication action that involves a machine agent and which matches one of the actions in the object messaging OSM (i.e. sending or returning messages). For this situation, the validation frame identifies at least 5 generic system requirements:

validation-frame
comment: generic requirements for send action, object messaging OSM situation:
event(evA,_,acA) AND
action(acA,communication,[agA,agB]) AND
[agent(agA,machine) OR agent(agB,machine)] AND match(acA,acX,osm(acX,object-messaging));
requirements:
requirement(functional, mandatory, link), generic-requirement(GOM1, system shall support identification/retrieval of receiver agent's address);
generic-requirement(GOM2, system shall enable a user to enter the receiver agent's address);
generic-requirement(GOM3, system shall enable a user to enter the content of a message);
generic-requirement(GOM4, system shall enable a user to send a composed message to the receiver agent);
generic-requirement(GOM5, system shall maintain a sender-agent log of all messages which are sent from and received by the sender-agent);
end-validation-frame

For example, the validation frame is applicable to event S30 which starts the action 'buyer requests quote from dealer'. Each of the generic requirements is applicable to computerised support for this action, whether or not the action is undertaken by telephone or e-mail, for example retrieving the receiver agent's telephone number or e-mail address, and maintaining a sender log of all telephone calls or e-mail messages. Such generic requirements can be instantiated and added to the requirement specification.

4. Discussion

The contributions to RE we have reported in this paper are threefold. First, extensive reuse of requirements knowledge is empowered via decision models and generic requirements. Secondly a means of semi-automatic requirements validation is provided via frames. Frames extend type checking by recognising patterns of agents' behaviour to which appropriate validation questions, and possible design solutions, may be applied. Third, we have described the use of scenarios as test pathway simulations, with novel tool support for semi-automatic scenario generations. The current status of development is that the scenario generator-validator tool has been implemented and industrial trials are commencing. Clearly coverage in terms of the number of validation frames and generic requirements contained in the tool database is a key issue effecting the utility of our approach. Our approach is eclectic and depends on knowledge in literature such as from ergonomics, human resource management and user interface design. The contribution we have made to implement and integrate this knowledge in an extensible architecture. The advice currently contained retrieved by the validation frames provides requirements knowledge at a summary level. Although this may be criticised as lacking sufficient detail initial industrial reaction to the tool has been encouraging, in particular the value of raising design issues which are not addressed by current methods, e.g. DSDM [15]. So far we have demonstrated proof of concept in terms of operation. This will be followed by further testing of utility within a small scale, but industrially realistic application.

In many senses the strength of the method we have proposed lies in integration of previous ideas. We have brought concepts from safety critical system assessment [24], [42] to bear on requirements analysis, and integrated these with scenario based approaches to RE. We acknowledge the heritage of the Inquiry Cycle [39]; however, our research has contributed an advanced method and support tool that give more comprehensive guidance for solving RE problems. Specification of requirements to deal explicitly with the implications of human error is a novel contribution where we have broken ground beyond the previous approaches [30], [16]. Furthermore, the influencing factors that bear on causes for potential error are useful heuristics to stimulate debate about many higher level requirements issues, such as task and workplace design. However, we acknowledge it is difficult to provide prescriptive guidance from such heuristics. While some may contend that formalising analytic heuristics can not capture the wealth of possible causes of error in different domains, we answer that some heuristics are better than none and point out that the method is incremental and grows by experience. Failure to formalise knowledge can only hinder RE.

Parts of the scenario based method reported in this paper are related to the enterprise modelling approach of Yu and Mylopoulos [54] and Chung [7]. They create models of the system and its immediate environment using similar semantics for tracing of dependencies between agents, the goals and tasks with limited reasoning support for trade-offs between functional requirements and non-functional requirements (referred to as soft goals). However the i* method does not contain detailed event dependency analysis such as we have reported. Scenarios have been used for assessing the impact of technical systems by several authors [6], [30], [16]. However, these reports give little prescriptive guidance for analysis, so the practitioner is left with examples and case studies from which general lessons have to be extracted. For instance, the ORDIT method [16] gives limited heuristics that advise checking agent role allocations, but these fall far short of the comprehensive guidance we have proposed.

Dependencies between systems and their environment have been analysed in detail by Jackson and Zave [28] who point out that input events impose obligations on a required system. They propose a formalism for modelling such dependencies. Formal modelling is applicable to the class of systems they implicitly analyse, e.g. real time and safety critical applications, but it is less clear how such models can deal with the uncertainties of human behaviour. To deal with uncertainty in human computer interaction, we believe our scenario based approach is more appropriate as it focuses on eliciting requirements to repair problems caused by unreliable human behaviour. Another approach is the KAOS specification language and its associated GRAIL tool [32], [33] a formal modelling that refines goal-oriented requirements into constraint based specifications. Van Lamsweerde et al [33] have also adopted problems and obstacles from the Inquiry cycle [39]; furthermore, they have also employed failure concepts from the safety critical literature in a similar manner to CREWS-SAVRE. Their approach is anchored in goal-led requirements refinement and does not use scenarios explicitly. In contrast, CREWS-SAVRE covers a wider range of issues in RE than KAOS but with less formal rigour, representing a trade-off in RE between modelling effort, coverage and formal reasoning.

So far the method has only partially dealt with non functional requirements. Scenarios could be expressed in more quantifiable terms, for instance by the Goal- Question-Metric approach of Basili et al. [4], or by Boehm's [5] quality-property model. Scenarios in this sense will contain more contextual information to represent rich pictures of the system and its environment [29]. The description could be structured to include information related to the NF goal being investigated and metrics for benchmark testing achievement of the goal. Validation frames may be extended for assessing such rich picture scenarios for non functional and functional requirements. For instance, each inbound/ outbound event that involves a human agent will be mediated by a user interface. Usability criteria could be attached to this event pattern with design guidelines e.g. ISO 9241 [25]. Performance requirements could be assessed by checking the volume and temporal distribution of events against system requirements. Elaborating the scenario based approach to cover non functional requirements is part of our ongoing research [52].

In spite of the advances that scenario based RE may offer, we have still to demonstrate its effectiveness in practice. There is evidence that the approach is effective in empirical studies of earlier versions of the method which did use scenarios but without the support tool [50]. Further validation with industrial case studies is in progress.

Acknowledgements

This research has been funded by the European Commission ESPRIT 21903 long term research project 'CREWS' - Co-operative Requirements Engineering With Scenarios. The project partners include RWTH-Aachen (project co-ordinator), City University, London, University of Paris I, France, FUNDP, University of Namur, Belgium.

References

[1] C. B. Achour and C. Rolland, '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, 1997.

[2] J. Allen, 'Maintaining knowledge about temporal intervals', Communications of the ACM, vol. 26. No. 11, 1983.

[3] R. W. Bailey, Human Performance Engineering, Prentice Hall, 1982.

[4] V. R. Basili and D. M. Weiss, 'A methodology for collecting valid software data', *IEEE Transactions on Software Engineering*, vol. 10, no. 3, pp. 728-738, 1984.

[5] B. Boehm and Hoh In, 'Identifying Quality-Requirement Conflicts', IEEE Software, pp. 25-35, March 1996.

[6] C. M. Carroll, 'The scenario perspective on system development', in J. M. Carroll(ed.), Scenario-based design: Envisioning work and technology in system development', John Wiley, New York, 1995.

[7] L. Chung, 'Representing and Using Non-Functional Requirements: A Process-Oriented Approach', *Technical Report*, DKBS-TR-93-1, University of Toronto, June 1993.

[8] P. Coad, D. North and M. Mayfield, 'Object Models: Strategies, Patterns and Applications', Englewood Cliffs, Prentice-Hall, 1995.

[9] P. Coad and E. Yourdon, Object Oriented Analysis, Yourdon Press, Englewood Cliffs, New Jersey, 1991.

[10] A. Cockburn, 'Structuring Use Cases with Goals', http://members.aol.com/acockburn/papers/usecase.htm, 1995.

[11] J. Conklin and M. L. Begeman, 'gIBIS: A hypertext tool for exploratory policy decision', Transactions on Office Information Systems, vol. 6, no. 4, pp. 303-331, 1988.

[12] A. Dardenne, A. van Lamsweerde and S. Fickas, 'Goal-directed requirements acquisition', Science of Computer Programming, vol. 20, pp. 3-50, 1993.

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

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

[15] Dynamic Systems Development Method (DSDM) Version 3.0, DSDM Consortium, Tesseract Publishing, September 1997.

[16] K. D. Eason, S. D. Harker and C. W. Olphert, 'Representing social-technical system options in the development of new forms of work organisation', European Journal of Work and Organizational Psychology, vol. 5, no. 3, pp. 399-420, 1996.

[17] S. R. Faulk, 'Software Requirements: A Tutorial, in Software Engineering, M. Dorfman and R. H. Thayer (eds.), pp. 82-103, 1996.

[18] M. Fowler, Analysis Patterns: Reusable Object Models, Addison-Wesley, 1997.

[19] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of reusable object-oriented software, Addison-Wesley, Reading, Mass., 1994.

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

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

[22] R. Hauser and D. Clausing, 'The House of Quality', *Harvard Business Review*, May-June 1988, pp. 63-73.

[23] P. Heymans and E. Dubois, 'Some Thoughts about the Animation of Formal Specifications written in the ALBERT II Language', CREWS Technical Report, Computer Science Department, University of Namur, Belgium, 1997.

[24] E. Hollnagel, 'Human Reliability Analysis Context and Control', Academic Press, 1993.

[25] ISO 9241, Ergonomic requirements for office systems visual display terminals, Parts 10, 11, 16 International Standards, parts 1-9, 12-15, 17, draft standards; International Standards Organisation, Switzerland, available from National Standards Organisation.

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

[27] M. A. Jackson, 'System Development', Prentice Hall, 1983.

[28] M. Jackson and P. Zave, 'Domain descriptions', in IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, pp. 56-64, 1993.

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

[30] M. Kyng, 'Creating contexts for design', in J. M. Carroll(ed.), Scenario-based design: Envisioning work and technology in system development', John Wiley, New York, 1995.

[31] W. Lam, J. A. McDermid and A. J. Vickers, 'Ten steps towards systematic requirements reuse', Proceedings 3rd IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, pp. 6-13, 1997.

[32] A. van Lamsweerde and E. Leiter, 'Integrating obstacles in goal-driven requirements engineering', Technical Report, Dept. of Computer Science, Catholic University of Louvain, 1997.

[33] A. van Lamsweerde, R. Darimont and P. Massonet, 'Goal directed elaboration of requirements for a meeting scheduler: Problems and lessons learned', Proceedings 2nd IEEE Symposium on Requirements Engineering, IEEE Computer Society, pp. 194-203, 1995.

[34] N.A.M. Maiden and A. G. Sutcliffe, 'Requirements Critiquing Using Domain Abstractions', Proceedings IEEE Conference on Requirements Engineering, IEEE Computer Society Press, pp. 184-193, 1994.

[35] C. Mazza, J. Fairclough, B. Melton, D. De Pable, A. Scheffer and R. Stevens, 'Software Engineering Standards', Prentice Hall, 1994.

[36] J. Nielsen , 'Usability Engineering', Academic Press, 1993.

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

[38] J. D. Palmer, 'Traceability', in Software Engineering, M. Dorfman and R. H. Thayer (eds.), pp. 266-276, 1996.

[39] C. Potts, K. Takahashi and A. I. Anton, 'Inquiry-Based Requirements Analysis', IEEE Software, vol. 11, no. 2, pp. 21-32, 1994.

[40] C. Potts, K. Takahashi, J. Smith and K. Ora, 'An evaluation of inquiry based requirements analysis for an Internet service', Proceedings 2nd IEEE Symposium on Requirements Engineering, IEEE Computer Society, pp. 27-34, 1995.

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

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

[43] S. Roberston at the Atlantic Systems Guild, United Kingdom, personal communication.

[44] T. Royer, 'Using Scenario-based designs to review user interface changes and enhancements', Proceedings DIS95: Designing Interactive Systems, Ann Arbor, 237-246.

[45] M. S. Sanders and E. J. McCormick, Human Factors Engineering and Design, McGraw-Hill, 1992.

[46] B. Shneiderman, 'Designing the User Interface: Strategies for Effective Human-Computer Interaction', Addison-Wesley, Third Edition, 1997.

[47] A. G. Sutcliffe, Human Computer Interface Design, 2nd Edition, Macmillan, London, 1994.

[48] A. G. Sutcliffe, 'Requirements rationales: integrating approaches to requirements analysis', in Proceedings of Designing Interactive Systems (DIS'95), ACM Press, New York, pp. 33-42, 1995.

[49] A. G. Sutcliffe and G. Rugg, 'A taxonomy of error types for failure analysis and risk assessment', Centre for HCI Design, City University, London, Centre Report HCID/94/17, 1994.

[50] A. G. Sutcliffe, 'A Technique Combination Approach to Requirements Engineering', Proceedings 3rd IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, pp. 65-74, 1997.

[51] A. G. Sutcliffe and N.A.M. Maiden, 'The Domain Theory for Requirements Engineering', IEEE Transactions in Software Engineering, November 1998.

[52] Alistair Sutcliffe and Shailey Minocha, “Scenario-based Analysis of Non-Functional Requirements”, Workshop on ‘Requirements Engineering For Software Quality’ (REFSQ’98) at CAiSE’ 98 in Pisa, Italy, June 1998.

[53] A. G. Sutcliffe, ‘Domain Analysis for Software Reuse’, to appear in Journal of Systems and Software.

[54] E. Yu and J. Mylopoulos, ‘Towards modelling strategic actor relationships for information systems development - with examples from business process reengineering’, Proceedings 4th workshop on Information Technology and Systems (WITS’94), Vancouver, B.C. Canada, December 17-18, 1994.

Appendix A

Selection of the Object System Models used in the case study- for more details see Sutcliffe and Maiden [51]

Level-1 class: Object Supply

This class is characterised by the unidirectional transfer of key objects from an owning structure to a client agent who generated the request for the object. Transferred objects are moved physically in a different container and are consequently owned by a different agent. Familiar examples are sales order processing systems and inventory applications. The model has one structure object, two agents (client and resource owners), one key object, one event that models the request for the transaction and one state transition for delivery of the object. The common purpose of this class is the orderly transfer of resource objects from an owner to a client who requires the resource. The goal state asserts that all requests must be satisfied by transfer of key objects to the requesting client. Specialisations of this class distinguish between transfer of physical objects, while a separate abstraction is used for the monetary aspects of sales.

Goal state	to satisfy all resource requests by supply of resource objects
Agents	client (requester), supplier
Key object	resource
Transitions	request, supply-transfer

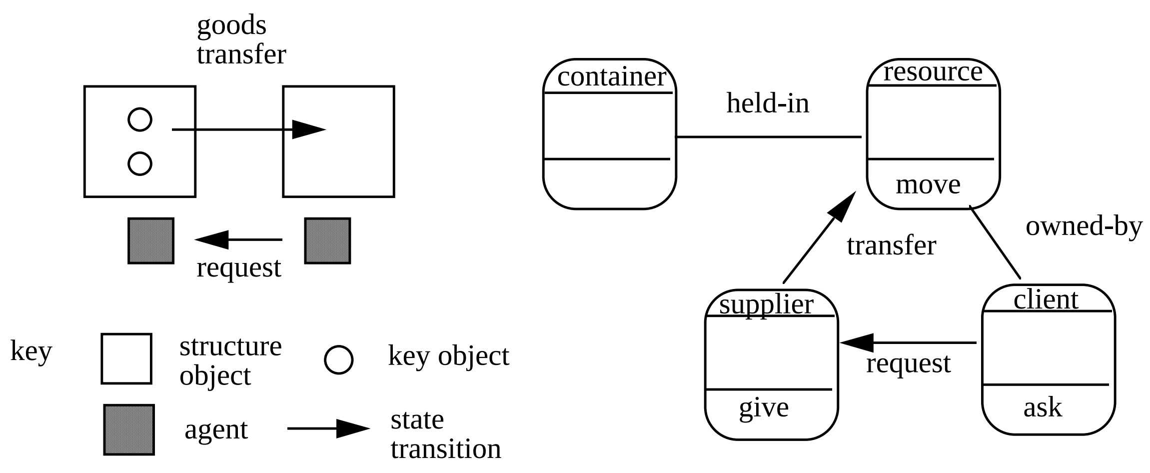


Figure A1. Object Supply OSM. Each OSM is illustrated as an informal diagram using the metaschema semantics and a corresponding model in Object-Oriented Analysis notation . This shows classes, inheritance, aggregation and instance relationships with message connections. Only key attributes and methods are included.

Level-1 class: Agent Control

This OSM family models command and control applications. The class is characterised by a controlling agent and one or more controlled agents contained in an object structure which models the real world environment, e.g. airspace in air traffic control. The transitions represent messages passed from the controlling agent to the controllee to change its behaviour in some way.

Goal state: for one agent to control the behaviour of another; all commands issued by the commanding agents have to be responded to in some manner by the controlled agent

Agents: controller, controllee

Key objects: as for agents

Transitions: command, respond

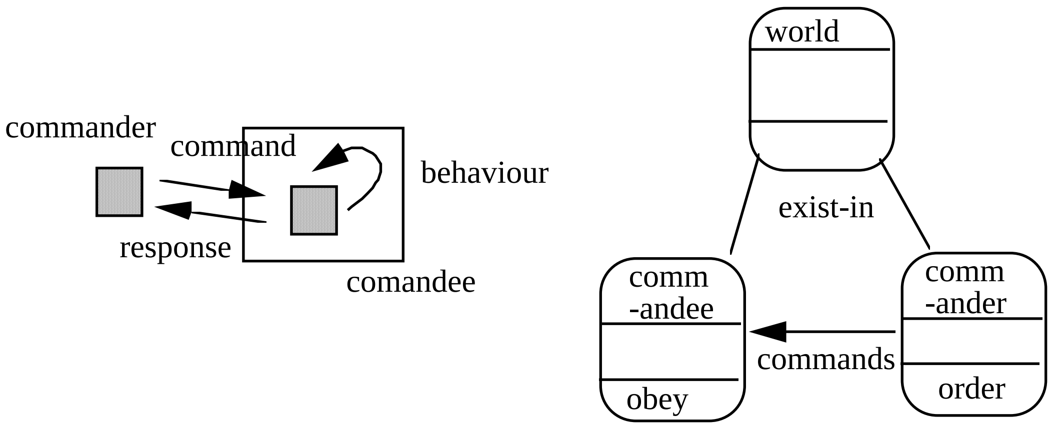


Figure A2. Agent Control OSM which has two agents instead of key objects.

Level-1 class: Object Sensing

The purpose of this class is to monitor the physical conditions or movement of objects in the real world and record their relative state with respect to a spatial/physical structure object. This class has a sensing agent which reports on the key object. Sub-classes in this family are determined according to whether it is a property of the key object which is to be detected or spatial behaviour, and whether the sensing agent is active or passive, e.g. spatial sensing by radar detection of air traffic; or property sensing applications, e.g. monitoring temperature, pressure in process control applications. Object Sensing models are often aggregated with the Agent-Object Control class.

Goal:	to detect any state change in a key object and report that change to an external agent
Agents:	sensor, client
Key objects:	real world objects
Transitions:	detect, report

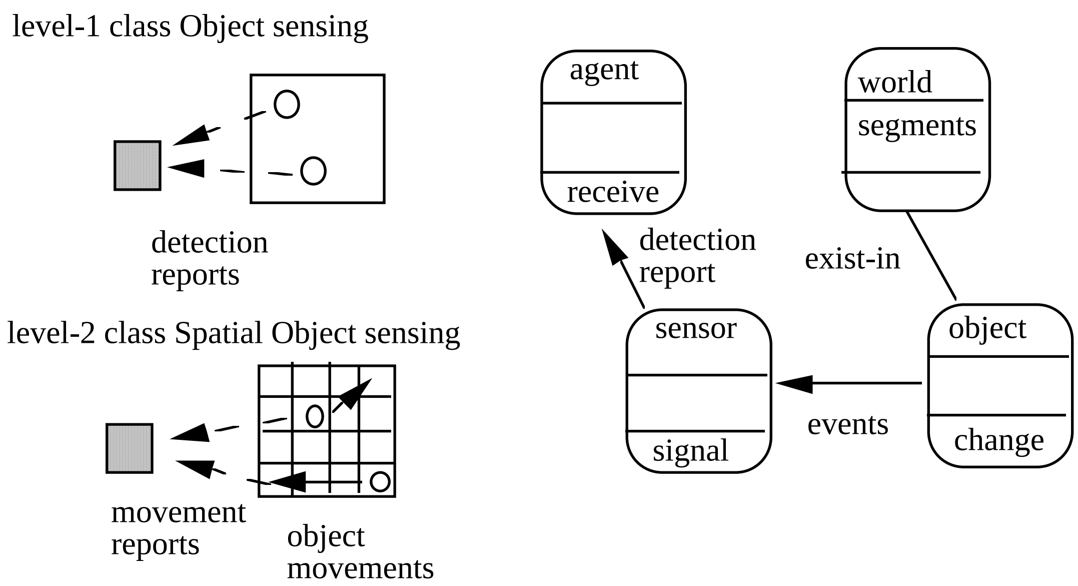


Figure A3. Object Sensing OSMs; the sensor agent may be a means of communicating events or an active detection device. The OOA model is the same for both the level-1 and level-2 classes, although the event and use of segment attributes in the Spatial Object Sensing will be different.

Table AI Decision Table for Location OSMs. Each column is linked to a set of keywords that identify the generic models in a variety of applications.

	Hiring or renting	repair & maintenance	broking & matching	supplying goods	manufacture	command & control	Monitor world	Real time &safety critical	models & games	communication & transport
Object supply				●						
Object hiring	●									
Object service		●								
Object alloc			●							
Object Comp					●					
Object Decomp		○			○					
Object sensing						○	●	●		
Agent control					○	●		●	○	
Object logistics				○						●
Object Synth					●					
Object Sim									●	

Key: ● Strong association

○ Weaker association