7

Acquiring requirements: a domain-specific approach

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 outlines two domain-specific techniques to improve the acquisition of requirements for software-intensive systems. The principles ideas, theoretical models and computational mechanisms needed to implement these techniques are presented. Both techniques exploit results from the ESPRIT III NATURE basic research action which proposes domain-specific requirements engineering environments. NATURE provides a set of problem abstractions which are the basis for intelligent guidance during requirements acquisition. After outlining these abstractions, the paper presents a framework for guiding requirements acquisition which incorporates the two techniques.

Keywords

Requirements engineering, requirements acquisition, domain modelling, scenario generation

1 REQUIREMENTS ENGINEERING

The task of 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. Easterbrook 1993, Sommerville et al. 1993), tools (e.g. Johnson et al. 1992, Nuseibeh et al. 1994) and languages (e.g. Mylopoulos et al. 1990, Yu 1993) which enable formal specification and validation of requirements. In comparison, acquisition of requirements from stakeholders has been neglected. Such acquisition is different because requirements are often expressed as linguistic expressions (Sutcliffe & Maiden 1993) which are not amenable to formalisation or computational reasoning. It is now agreed that new methods and tools are needed.

The ESPRIT 6353 'NATURE' basic research action (Jarke et al. 1993) has identified a large set of problem abstractions to provide domain-specific guidance for requirements engineers. Problem abstractions are semi-formal models of the fundamental behaviours, states, objects, agents, goals, constraints and functions which belong to members of possible categories of requirements engineering problem. In this sense, these abstractions are similar to problem frames (Jackson 1995) or clichés (Reubenstein & Waters 1991). However, NATURE has derived a larger data base of problem abstractions (e.g. Sutcliffe & Maiden 1994), undertaken empirical validation (e.g. Maiden et al. 1995) and constructed tools to exploit the abstractions during activities such as requirements structuring (Maiden & Sutcliffe 1993), critiquing (Maiden & Sutcliffe 1994) and communication, as well as reuse (Maiden & Sutcliffe 1992).

A. Sutcliffe et al. (eds.), *Domain Knowledge for Interactive System Design*

This paper proposes using NATURE's problem abstractions to provide domain-specific guidance for requirements acquisition. Once the requirements engineer has identified the categories of the problem domain under analysis, problem abstractions can be used to generate large numbers of questions about the problem domain and scenarios describing common and unexpected events which the system must handle. Both techniques aid acquisition of both requirements of the software system and knowledge about that system's environment, described by diverse phenomena in the problem domain such as behaviour, events, structure and states. Furthermore these two techniques are embedded in a method framework to guide selection of these and other requirements acquisition methods, thus providing a more complete outline approach for requirements acquisition.

This paper is in four parts. ACRE, the method framework for requirements acquisition is outlined. This is followed by two domain-specific techniques for requirements acquisition in the form of automatic question and scenario generation. The paper ends with future research directions, focusing on further implementation of the work presented in this paper. First however, the next section reviews the set of NATURE's problem abstractions.

2. NATURE'S SET OF PROBLEM ABSTRACTIONS

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 problems. 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 problem abstractions 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 problem abstractions. Alternative means of detecting and validating such problem abstractions, 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. NATURE's emphasis on cognitive validation (e.g. Maiden et al. 1995) is a novel and powerful approach to this research problem.

Each problem abstraction is a composition of several object system models and information system models. Object system models define all features of one class of requirements engineering problem which discriminate it from other classes. These are stored in a hierarchical class structure. An orthogonal set of information system models defines functions which report on states defined in object system models. Object system models are defined in the hierarchical class structure in a systematic manner. 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 1994).

The object system models are structured in 13 hierarchies, see Figure 1. The top-level object system model of each hierarchy is defined using the basic behaviours, states, objects, agents and domain structure 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, higher-order relations between objects and agents, 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. For example, domain simulation models describe complex environment simulations for

human agents. The model at the top of this hierarchy is specialised according to domain structure (e.g. user is inside/outside the environment, number of users in the environment), event triggers (e.g. real-time nature of interaction) and object properties (e.g. physical or conceptual). Furthermore, most 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.

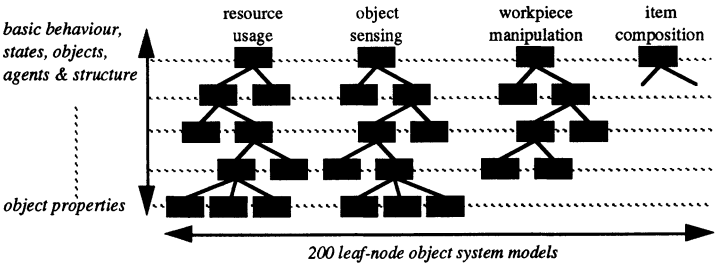


Figure 1 Graphical depiction of the hierarchical class structure of object system models.

3. AIR TOOLKIT

The AIR (Advisor for Intelligent Reuse) tookit is composed of six tools which exploit object system models during requirements engineering tasks. Its current architecture is shown in Figure 2. Several tools have been designed and implemented on a SparcStation IPX under UNIX using the ConceptBase object-oriented deductive data base (Jarke et al. 1994) and ProLog by BIM. Others are being implemented on the same platform. 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 Maiden & Sutcliffe (1994).

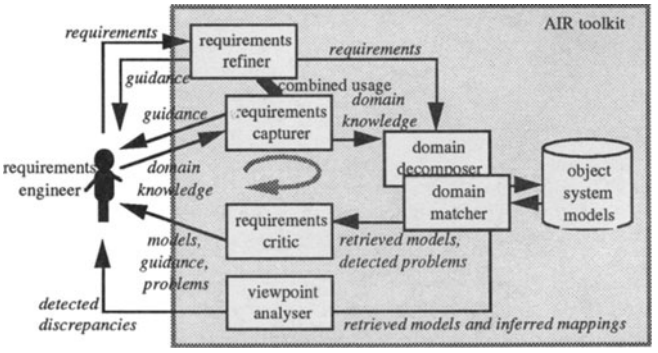


Figure 2 Outline architecture of NATURE's AIR toolkit.

At the heart of the toolkit are two computational mechanisms which retrieve and reason with object system models:

- the domain matcher is a computational analogical reasoning mechanism for retrieving object system models (Maiden & Sutcliffe 1994). 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. Matching also retrieves information system models linked to the retrieved object system models. Another tool, the problem classifier, exploits inferred object-pair and fact-pair mappings to aid detection of problem situations in entered requirements. This is linked to yet another tool called the requirements critic which aids problem understanding and requirements critiquing by explaining retrieved object system models and detected problem situations to the requirements engineer;
- the domain decomposer is the second computational mechanism. It retrieves multiple object system models from a larger requirements problem (Maiden & Sutcliffe 1996). New requirements are neither well-defined or well-structured. Therefore, the mechanism combines domain semantics with pattern matching algorithms to detect fact and requirement patterns for then retrieving single object system models using the domain matcher.

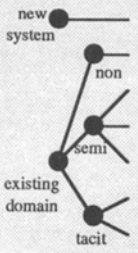
These two computational mechanisms, when combined with the set of object system models, provide a basis for determining the types of knowledge to acquire and generating large numbers of domain-specific questions and scenarios. Next however, the methodological framework for using these and other techniques is outlined.

4. ACRE: METHODS FOR ACQUIRING REQUIREMENTS

A framework called ACRE (ACquisition of REquirements) provides guidelines for method selection based on theoretical models and empirical evidence from cognitive and social science. ACRE offers 12 acquisition methods as a representative sample of the types available. Most of these methods are also comparable in their objectives, duration and manpower needed. ACRE's central premise is that different methods are more effective for acquiring different types of knowledge and requirements. Six facets inform method selection:

- *purpose of requirements*: requirements can be acquired for different purposes such as specification of bespoke systems, selection of software packages and to provide a legal contract for requirements procurement;
- *knowledge types*: requirements modelling languages include semantic primitives such as events, states and agents. ACRE proposes methods for acquiring different types of knowledge;
- *internal filtering of knowledge*: it is often the case that stakeholders are unaware of their own knowledge and its boundaries. Problems can include poor recall and communication of incomplete or incorrect knowledge. Methods are also offered to overcome these limitations;
- *observable phenomena*: some knowledge cannot be communicated by stakeholders but only learned by observing the domain and its environment;
- *acquisition context:* method choice also depends on the context of its use. Complex organisational, political, financial and temporal pressures influence acquisition and method selection in the framework also recognises this;
- *method interdependencies*: an acquisition programme will include a sequence of methods, therefore selection is influenced by other chosen methods. The need for such sequencing and interaction was shown in the case studies with card sorting and laddering.

The first factor to consider is whether the method must acquire knowledge about the existing domain or requirements for the new system. Scenario analysis, prototyping and RAD are better for acquiring requirements for new systems. Scenarios and prototypes can provide good simulations of the future system and its interaction with its environment. Such simulations provide powerful tools for determining requirements.



Internal representation of knowledge	observation	u. interviews	s. interviews	protocols	card sorting	laddering	rep. grid analysis	brainstorming	rapid prototyping	scenario analysis	RAD	ethnographic
Future system knowledge	x	√	√	x	-	√	-	√	√√	√√	√√	x
Non-tacit knowledge	-	√√	√√	√√	√√	√√	√√	√	-	√√	√√	-
Recognised knowledge	x	x	x	-	√	-	-	√	√√	√√	√√	x
TFG knowledge	√√	-	-	√	-	-	-	-	√	√	√	√√
Working memory knowledge	x	x	x	√√	x	x	x	x	x	x	x	x
Compiled knowledge	√√	-	-	√√	-	-	-	-	√	√	√	√√
Implicit knowledge	√√	-	-	√√	-	-	-	-	√	√	√	√√

When considering existing domain knowledge, the framework makes a broad distinction between non-tacit, semi-tacit and tacit knowledge. Recall of non-tacit knowledge is uncontroversial but subject to numerous well-documented biases. For example, even domain experts may selectively recall only examples which fit with their preconceived schemata, and forget examples which do not. ACRE proposes using verbal methods for acquiring non-tacit knowledge.

Semi-tacit knowledge has three factors to consider:

- recognition of non-tacit knowledge differs from recall in the access to knowledge. Recall often accesses only a fraction of the knowledge which can be recognised. For example, most people are able to recall only a few function names of a software package which they routinely use, but are able to recognise all of them if shown a list. Rapid prototyping, scenario analysis and RAD are the most effective methods. Some brainstorming methods also produce models.
- taken-for-granted (TFG) knowledge refers to the filtering process in communication (Grice 1975) which leads to the omission of information which can be taken for granted (e.g. one would not say: "My computer, which is a machine...."). Stakeholders often treat knowledge as TFG when it involves everyday things, forgeting that it may be unknown to outsiders such as requirements engineers. TFG knowledge is best acquired using methods which do not rely on communication such as observation, protocol analysis and ethnographic methods.
- working memory is a limited-capacity, short-duration memory for information used during a task which is forgotten within a few seconds unless there is some reason to commit it to long term memory. Only protocol analysis, which combines observation with verbalisation of thought processes, can capture knowledge in working memory.

Tacit knowledge subsumes compiled and implicit knowledge. For example, compiled knowledge is knowledge which was once non-tacit, but which has since been practised so often that it has become habitualised and speeded up to the point where accurate introspection is no longer possible (Anderson 1990). ACRE promotes use of observational methods (observation, protocol analysis, ethnographic methods) for acquiring both compiled and implicit knowledge.

Figure 3 Example of guidelines provided in the ACRE framework.

The ACRE method is presented as simple steps and look-up tables which guide planning of a requirements acquisition programme and choice of methods for each acquisition session. For

example, consider the facet "internal filtering of knowledge", see Figure 3. Interviewing is a traditional method for acquiring requirements. However, it assumes that the stakeholder has conscious, accurate access to all relevant knowledge. In fact though, this is often not the case. Much knowledge is not accessible to conscious introspection, and of the knowledge which is, much may be missed. Alternative methods are recommended.

The two domain-specific techniques described in the remainder of the paper can be seen as supporting methods in the ACRE framework. Question generation can inform unstructured interviewing and both inform and shorten the preparation time for structured interviewing, laddering, brainstorming and the use of prototypes. Automatic scenario generation has an obvious role for scenario analysis. It provides a more complete set of scenarios and reduces preparation time considerably. It can also provide information for informing evaluation of prototypes. Both techniques are now described in more detail.

5. DOMAIN-SPECIFIC QUESTION GENERATION

Requirements acquisition involves asking a lot of questions. However, a reoccurring problem is that important questions are often unforeseen or overlooked. Furthermore, when the right questions are asked, they are not asked in the order which is most effective. Several commercial tools are available to overcome these problems, for example the GMARC method and tool (CSA 1995). However, a better understanding of which questions to ask to obtain the right answers is needed.

Question generation

The essential but simple idea is to turn NATURE's object system models "inside-out". Each feature of the retrieved model then becomes the topic focus for a set of questions. This topic focus is associated with different questions depending on the type of feature in a model, such as an object, agent, state and behaviour. The result is a large number of questions about generic features. To make these questions more domain-specific, the requirements engineer can be asked to give simple mappings between question objects and domain objects. For example, questions would refer to the 'air traffic controller' rather than the 'controller agent'. One advantage from generating questions in this manner is to enable the acquisition session to be planned. Retrieved object system models suggest the structure of the problem space. The order in which requirements engineers ask questions should reflect this structure. Acquisition planning is also improved because the likely boundaries and key features of this space are defined at the beginning of the acquisition session.

Computational mechanisms

Generating a sequence of questions is, at least in part, clerical and hence amenable to automation. Taking its input from the domain knowledge in object system models, question generation is automated using a simple computational mechanism called the question generator. This mechanism informs question generation using the: (i) topic foci of questions, (ii) question types. A set of predefined questions is asked about each feature in the object system model according to its type. Question templates are filled using domain-specific features and tailored using predefined mappings between object system models and question types.

Question types

Each feature of an object system model is presented to requirements engineers in different styles, although all use natural language to ensure effective communication with end-users. These styles are questions (Potts et al. 1994), summarising statements (Reubenstein & Waters 1991) and simple models (Maiden & Sutcliffe 1994). Several different, system-generated styles of questions can be identified. The system generates questions of the type *what-is*, *how-to*,

who, *what-kinds-of*, *when*, *relationships* and *what-if* (Potts et al. 1994). The type of question asked is driven from the desired answers. The hierarchical structure of the models also informs procedural guidance for this acquisition through prioritisation of questions.

Question presentation

Questions are presented in two basic forms, depending on the context in which acquisition will take place. Often acquisition takes place on the stakeholder's own ground where use of technological support is difficult, if not impossible. In such cases, a booklet of questions is produced to act as a checklist and a guide. If technological support is possible, interactive guidance for questions to ask during the session is provided. Again, guidance for sequencing and prioritising questions can be provided using the structure of models in the data base. Furthermore, the depth and direction of questioning can be constrained depending on external factors such as the time available for acquisition and the level of knowledge of the stakeholder. Such tailorability is seen as critical for effective acquisition guidance.

Example

Consider retrieval of the resource hiring system model, which describes all library circulation control problems such as book lending or car hire. This model has five basic objects: the resource, the borrower, lender, the container (holding the resource) and a controller (who controls circulation). A large number of critical questions can be asked about each of these objects, including: do you have these objects, what are their names and attributes, how do each of these objects behave, who manipulates each of the objects, and what are the relationships between them? The model also includes a number of states, such as whether a book is loaned or not, whether it is reserved or not, as well as the structure of the domain itself. Questions about states can include: when does a state arise, how long does the state last for, what stops the state, and what are attributes of the state? The model also includes events, state transitions, goal states and object properties. Different questions can be generated for all of these phenomena types. The question generator achieves this with a small data base of predefined question types and presentation templates. The result is several hundred questions from just one simple object system model. This provides significant, scaleable and planned guidance for more complete requirements acquisition.

Advantages

The automatic question generation approaches outlined here have at least four clear advantages. First, generating the topic focus of questions from the models ensures greater coverage and hence question completeness. This is because object system models aim to include the key features of all requirements engineering problems. Second, the models provide a context for focusing and scoping problems. Stakeholders often misunderstand acquisition questions, therefore contextualising these questions is important. Indeed models can be presented alongside questions to provide this context in an explicit manner. Third, the hierarchies of object system models provide a basis for both layering and focusing questions, thus introducing more flexibility into the questioning process. This is linked to the fourth advantage, which is in that models enable the requirements engineer to define the order of questions before the acquisition session begins. Planning such acquisition sessions is both difficult and time-consuming, and object system models can provide an important basic structure for ordering questions.

6. AUTOMATIC SCENARIO GENERATION

Scenarios have been recognised as a useful technique for acquiring requirements. A scenario is a description of a sequence of actions or events for a specific case of some generic task which the system is meant to accomplish (Wexelblat 1987). Scenarios can serve as a useful, informal

explanatory device where questions about a desired future state cannot be easily answered. One advantage is that stakeholders do not need to learn any formal syntax. The importance of scenarios is shown by their recent integration into structured and object-oriented methods, such as use-cases in the OOSE method (Jacobson et al. 1992).

Scenario generation

Although benefits of scenarios for requirements acquisition have been shown, generating a large and useful set of scenarios remains a bottleneck (Gough et al. 1995). One solution is to extend the question generation approach outlined above to the generation of scenarios. Question generation is simple. It generates different questions for individual features of an object system model. Scenario generation is different. It identifies different permutations of these model features to generate a set of possible scenarios. Agents, events, transitions, states and goal states are all fundamental components of both object system models and scenarios (Potts et al. 1994). These can be manipulated, as a set, to determine different permutations, or scenarios, for a problem domain. These permutations are extended using a set of exception conditions which define unforeseen situations and events in problem domains. Furthermore, features in object system models are interconnected, thus enabling the imposition of useful constraints on scenario generation. A computational mechanism to generate these permutations has been designed to overcome the scenario generation bottleneck.

It is important to note that scenario generation is more sophisticated than just deriving all possible permutations of all actions, events, agents, state transitions and states in one object system model. The space of these permutations would be just too great. Rather the generation engine examines different permutations of causal chains of actions, events, state transitions and resultant states which are defined in the object system model. A causal chain is shown in Figure 4. For example, a borrower undertakes a sequence of actions which includes the event of loaning books, which causes the state of those books to be loaned to change. Each chain is divided into a series of causal links, each of which has a finite set of permutations. Therefore the set of all permutations for a causal chain is equal to the permutations of all permutations for each causal link, and the complete set of permutations for an object system model is equal to all permutations for all causal chains belonging to that model.

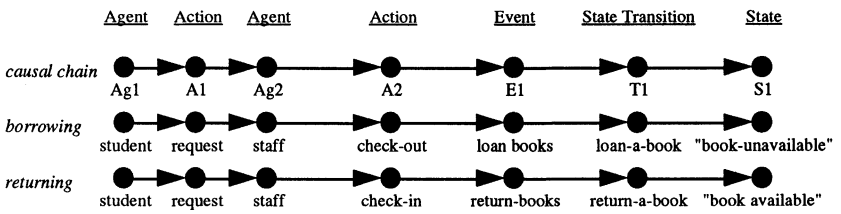


Figure 4 Causal chains as the basis for automatic scenario generation.

The library circulation control example

The object system model for resource hiring defines different causal chains for borrowing resources, returning resources, reserving resources and making resources available. This example focuses on the causal chain for borrowing. In this chain, in its generic form, the borrower agent requests the lender to undertake a sequence of events and actions which trigger a state transition which changes the state of the desired resource from "available" to "borrowed". Assuming once again that a simple mapping process is available to instantiate this causal chain, a student requests check-out staff to check-out a book which then changes state from "available" to "borrowed". However, more importantly, different permutations of this chain can be generated, for example:

- different agents or individuals can undertake the same actions, for example one student tries to borrow a book using the identification number of another student;
- actions might happen in a different sequence to the model sequence, for example the student asks to borrow the book without bringing it to the check-out desk;
- events can trigger different state transitions, for example one student might ask to loan the same book several times during a loan transaction;
- possible states when a state transition occurs, for example the requested book is reserved to either the student requesting it or to a different student.

Given that object system models describe abstractions of phenomena in the problem domain, scenarios also describe the problem domain and the system's interaction with it. Therefore scenarios can generate both direct and indirect requirements for the software system. Indeed, this generative role for scenarios should be exploited. Often one scenario will lead to recognition of a wider set of scenarios and requirements which cannot be generated using the engine described here. However, combining casual chain permutations for one object system model gives a large set of more complex permutations. It provides the baseline for generating a large set of scenarios which are then elaborated using human errors and technical problems.

Human errors

Systematic generation of scenarios needs a research basis to determine a complete and useful set of exception situations. Exceptions can arise from different sources. One is human error. Most problem domains include human agents who can make errors and hence give rise to exceptions. Human errors in requirements engineering were investigated by Bowers et al. (1994). Slips often occur during performance of familiar tasks and include attentional failures (e.g. a librarian failing to see when a borrower has overdue books), memory failures (e.g. forgetting to tag borrowed books for the security system) and selection failures (e.g. tagging books before checking them out). Mistakes occur in the formulation of actions and include bad application of rules (e.g. inappropriate charging of fines for overdue books) and various biases (e.g. confusing borrowers and over-confidence with the system). Human errors are attributable to human agents in object system models, and provide different permutations of this kind. Research into human error has led to a tentative classification scheme of such errors.

Technical problems

Most problem domains also involve computer systems and other technologies which are prone to act unexpectedly. Types of technical problem include power failures (e.g. the library looses power from the national grid or other power sources), "hanging" systems (e.g. the check-out system hangs during different check-out activities) and "overactive" systems (e.g. the security system signals illegal removal of books for all people entering and leaving). As with human errors, a tentative classification of such problems has been derived to inform scenario generation.

Back to the library circulation control example

Both human errors and technical problems enables generation of further permutations of the resource borrowing causal chain. Borrowing involves two human agents (borrower and lender). Lender errors include attentional failures (e.g. stamping with the wrong return date), memory failures (e.g. forgetting to tag borrowed books) and mistakes (e.g. failing to reserve requested books). The borrower is prone to similar errors such as memory errors, for example forgetting to hand over all books for checking out. Technical problems can also complicate borrowing. Power failures and "hanging" systems are all possible during a loan transaction, therefore alternative and repair strategies are needed. Combining these possible exceptions with

permutations of causal chains in object system models generates a large set of possible scenarios. Furthermore, human errors and technical problems are presented in a more useful context. It is possible to present the classification of human errors without scenarios, however scenarios provide a mechanism for both interpreting such errors as well as exploring problems and generating further scenarios.

Constraints on permutations

Combining different permutations of events, activities and states of object systems with large possible of exceptions (both human and technical) can result in a very large set of permutations. Therefore, effective use of scenarios generated in this manner needs to be constrained. Our approach implements several forms of constraint. The most important is the likelihood of the permutation occurring in the problem domain. Object system models define most common patterns of situations and actions. Scenarios often need to include less probable variations on these patterns. One option is to grade the likelihood of permutation on a model-by-model basis. In essence, each object system model is extended to include probable and important exceptions. For example, attentional failures in object sensing system models such as air traffic control and patient monitoring (Maiden et al. 1995) are often safety-critical and hence a priority. Second, business rules are applicable for constraining permutation by defining what should and should not happen, however further research is still needed in this direction. Third, the requirements engineers themselves can select the topic focus of each scenario depending on the agenda and stakeholders present at an acquisition session. Returning to circulation control in the library, an acquisition session with counter staff should use scenarios which involve counter staff, while a session investigating how books are loaned should utilise scenarios which include book loaning. The current algorithm can generate scenarios for specific objects, agents, activities and types of error, for example meetings might investigate safety-critical aspects of a system.

Summarising

The research outlined here represents a pragmatic solution for automatic scenario generation. Such generation needs domain knowledge, such as in the form of object system models, to provide context-specific knowledge for this task. This approach introduces some novel advantages. One is that the generating mechanism provides an indexing mechanism for their retrieval and use. One can envisage the requirements engineer retrieving different scenarios from a data base according to the topic focus or interests of stakeholders, for example during exploration of a specific exception or obstacle. However, there are still some outstanding research questions. The use of genetic algorithms for generation of scenario permutations remains an open issue. Further effort is also needed to extend and validate the taxonomy of human errors and technical problems. In most current cases, scenarios are presented using natural language. Further work is also needed to determine how to present generated scenarios to stakeholders. However, the main unanswered question is constraining the space of possible permutations to make it manageable.

7. FUTURE RESEARCH AND EXPLOITATION

This paper has described two domain-specific guidance guidelines for requirements acquisition. Requirements acquisition has received little attention from researchers, let alone with domain-specific guidance in mind, despite the fact that several problems, such as the scenario generation bottleneck, have been identified. However, acquisition also needs stronger methodological guidance, hence the inclusion of ACRE in this paper. This is ongoing work to be taken further at a theoretical level and a practical level through method and tool design.

There are several issues, despite those outlined so far, which warrant further exploration. First, domain-specific guidance first necessitates retrieval of relevant object system models. NATURE implemented computational mechanisms for retrieving object system models for partial problem descriptions, however earlier retrieval is needed. Two options are available. First, more

facilities for browsing and asking questions about object system models can be provided, to enable easier access. This approach seems feasible during multi-session bouts of acquisition. After a first, high-level analysis to scope the problem, the requirements engineer can select relevant object system models as basis for further, more detailed acquisition sessions. The second option is to place an emphasis on training requirements engineers about the data base of object system models, so that models can be quickly retrieved and used during acquisition sessions. Successful reuse programmes place an emphasis on training about code libraries: there is no reason why such training will not help retrieval of object system models.

The other area for further research is to extend and evaluate the classification of human errors and technical problems to enable generation of more complete and more realistic scenarios. This will involve both further studies of the literature and direct empirical studies. The author looks forward to reporting the results of this and other work outlined in this paper in the near future.

Acknowledgements

Part of this research was funded as part of the European Commission's ESPRIT III 6353 'NATURE' basic research action. Special thanks goes to Alistair Sutcliffe.

References

Anderson J.R., 1990, 'The Adaptive Character of Thought', Hillsdale NJ, Erlbaum.

Bowers J., Viller S. & Rodden T., 1994, 'Human Factors in Requirements Engineering', Technical Report REAIMS/WP1.2/LU004, Lancaster University, UK.

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.

CSA, 1995, 'Getting the Requirements Right - A Professional Approach', Computer Systems Architects Internal Document.

Easterbrook S., 1993, 'Domain Modeling With Hierarchies of Alternative Viewpoints', *Proceedings 1st IEEE Symposium on Requirements Engineering*, IEEE Computer Society Press, 65-72.

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.

Grice H.P., 1975, 'Logic and Conversation', in Cole, P. & Morgan, J.L. (eds.) *Syntax and Semantics* **3**, New York: Academic Press.

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.

Jarke M., Eherer S., Gallersdörfer R., Jeusfeld M. & Staudt M., 1994, 'ConceptBase - A Deductive Object Manager for MetaData Bases', *Journal of Intelligent Information Systems*, 1994.

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.

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., 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.

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

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

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

Nuseibeh B., Kramer J. & Finkelstein A., 1994, 'A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification', *IEEE Transactions on Software Engineering* **20(10)**, 760-773.

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.

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.

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., 1994, 'Domain Modeling for Reuse', *Proceedings 3rd International Conference on Software Reuse*, IEEE Computer Society Press, 157-164.

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.

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

Yu E.S.K., 1993, 'Modelling Organisations for Information Systems Requirements Engineering', *Proceedings IEEE Symposium on Requirements Engineering*, IEEE Computer Society Press, 34-41.