February 1999

CREWS: Towards Systematic Usage of Scenarios, Use Cases and Scenes

Matthias Jarke
RWTH Aachen, jarke@informatik.rwth-aachen.de

CREWS : Towards Systematic Usage of Scenarios, Use Cases and Scenes

Matthias Jarke
RWTH Aachen (jarke@informatik.rwth-aachen.de)

Contents

Electronic Business Engineering / 4. Internationale Tagung Wirtschaftsinformatik 1999.
Hrsg.: August-Wilhelm Scheer; Markus Nüttgens. – Heidelberg: Physica-Verlag, 1999

Abstract

In the wake of object-oriented software engineering, use cases have gained enormous popularity as tools for bridging the gap between electronic business management and information systems engineering. A wide variety of practices has emerged but their relationships to each other, and with respect to the traditional change management process, are poorly understood. The ESPRIT Long Term Research Project CREWS (Cooperative Requirements Engineering With Scenarios) has conducted surveys of the research literature and of the industry practice in scenario-based requirements engineering as a basis to develop a framework of approaches and research issues in the field. In two demonstrator prototypes, one based on textual scenario representations, the other on multimedia scenes, solutions to some of the most critical open problems from these surveys are being explored. The project results, besides being integrated in leading commercial software engineering environments, feed into a component-oriented method server on the Internet.

1 Introduction

Object-oriented programming and design have gained great popularity in the 1990s. At the programming level, languages such as C++ and Java demonstrate this trend, supported by the introduction of object-oriented and object-relational databases. At the design level, this has been further strengthened by the effort of a group of leading researchers and practitioners to establish an object-oriented Unified Modelling Language (UML), in order to avoid fragmentation of notations and to limit learning efforts. Despite the complexity of the overall UML formalism, its uniformity -- meanwhile accepted by standardisation bodies as well as by most leading vendors -- gives some hope that it might actually become used as widely as the predecessor structured development methodologies.

UML is essentially a language for design, to be used by developers. It hardly satisfies the demand for an adequate communications medium between users, developers, and other stakeholders. Such a communications medium is not only needed in requirements engineering at the start of the project, but also for maintaining user and stakeholder involvement throughout the systems lifecycle, as requirements, solutions, and the environment change. Projects that have, for example, started directly with developing class definitions often get lost in the complexity of these definitions, leading to costly failures.

Among the early object-oriented approaches, only OOSE has addressed this issue in a promising manner (Jacobson 1995). Consequently, Jacobson’s use-case approach has been recently included in UML (Fowler and Scott 1997). *Use cases* are graphical depictions which group collections of interaction scenarios between users and systems around one typical usage. In practice, these *scenarios* are usually written as textual narratives or, more formally, as message sequence diagrams. In domains where situations are difficult to describe by text, people also use multimedia *scenes*, in the form of videos, virtual reality, or interaction games. These practices have their roots in early experiences by management science and human-computer interaction, and are broadly summarised under the label of *scenario-based requirements engineering*.

However, as discussed below, the practice of scenario usage and management differs widely and little advice is available what techniques to use when, how to support them, and how to integrate them with the rest of the object-oriented software development process. In 1996, the European Community has therefore started a Long-Term Research project called CREWS (Cooperative Requirements Engineering With Scenarios) intended to address these problems. In order to accomplish these goals, the project undertook a number of activities whose results will be summarised in the remainder of this paper. These activities are listed below, together with pointers to publications in which individual details of the project are described:

- Section 2: Through an interdisciplinary workshop held at Dagstuhl Castle in early 1998, a general framework for integrating scenario-based techniques in model-based processes of continuous change management has been established (Jarke et al. 1998).
- Section 3: A comprehensive survey of the state-of-the-art in research has been conducted (Rolland et al. 1998) and contrasted with the analysis of about 25 industrial projects in which use cases, scenarios, and scenes were used with varying degrees of success (Arnold et al. 1998, Weidenhaupt et al. 1998).
- Section 4: Two prototypical requirements engineering environments addressing key problems identified in the empirical studies have been designed, implemented, and empirically evaluated (the evaluation is still ongoing at the time of this writing). One addresses requirements elicitation and validation using textual scenarios, the other requirements traceability and change envisioning based on multimedia scenes. Details of these tools and of related developments by other research groups world-wide can be found in recent Special Issues of the *IEEE Transactions on Software Engineering* (Jarke and Kurki-Suonio 1998) and of the *Requirements Engineering Journal* (Jarke 1998).

More guidance for scenario-based requirements engineering and for the integration with object-oriented software engineering is being encoded in an internet-based server during the current, final phase of the project (section 5).

2 The Role of Scenarios in Change Management

Change management research in fields such as management science, software engineering, and human-computer interaction has traditionally followed a *model-based re-engineering cycle* :

(a) formally re-construct the concepts and rationale behind the current system,

(b) specify the desired change at the conceptual level,

(c) implement the changed concepts to reach the new system while

(d) taking the legacy context into account.

In the context of object-oriented software engineering, the conceptual models are represented in UML. The “system” usually comprises both humans and computerised components interacting with each other and with their environment.

With the deeper immersion of IT usage and impact in everyday life, formal models often prove clumsy to develop and hard to understand, especially when multiple stakeholders are involved who have little IT expertise and who have difficulties to imagine how their life might change due to the planned system. Even where initial shared understanding exists, the above procedure describes but one step in a *continuous change process* which is hard to trace without strong linkage to reality.

A *scenario* describes (textually or graphically) a possible set of events that might reasonably take place; a *scene* captures the same in some form of multimedia. Its purpose is to *stimulate and document thinking* about current problems, possible occurrences, assumptions relating these occurrences, action opportunities and risks. Results from cognitive psychology (Carroll 1995) indicate that scenarios offer a *middle-ground abstraction* between models and reality, serve as a universally understood medium for *participatory design,* and *facilitate reuse* of design knowledge:

1. Scenarios focus design efforts on *use first and foremost.* What people can do with the old/new system, and the consequences for themselves and for their organisations, is described and analysed prior to detailing the system functions and features that enable this use.
2. Scenarios *suspend commitment* but support *concrete progress*: They vividly explain why a system is needed by showing what it is used for, but they also facilitate an analysis of design alternatives how it is used.
3. Scenarios provide a task-oriented design decomposition that can be used from *many perspectives*, including usability trade-off’s, iterative development, and manageable software design object models.

Consistent with these observations, *information systems engineering* employs scenarios as intermediate design artefacts in an expanded change process, as shown in figure 1 (Jarke et al. 1998). During early requirements elicitation, scenarios focus on problems with the current system. They thus help to discover change goals and elaborate them into more detailed requirements. Once requirements for a future system have been specified, future-state scenarios can be generated to validate requirements against reality and higher-level goals, but also help refine requirements for the handling of exceptional situations.

The number of possible scenarios for a change situation is even greater than the number of possible conceptual models. The choice and elaboration of scenarios and scenes must therefore be guided by the *change goals* expressed by the users and other stakeholders. Conversely, stakeholders obtain an elaboration of goals into more detailed *requirements* through the analysis and discussion of scenarios and scenes. In other words, we claim that a scenario-based approach, at least for large projects, is inextricably linked to explicit capture of a goals/requirements hierarchy. The actual conceptual models (class diagrams etc.) are then derived by considering both the elaborated scenarios and the goal/requirements hierarchy. Indeed, some of the more advanced requirements engineering tools, such as Rational's Requisite Pro, support hierarchical structuring of (textual) requirements. More sophisticated goal modeling techniques, together with approaches to map goals to process and object models, can be found in the research literature (e.g. Kaindl 1998; Mylopoulos et al. 1992).

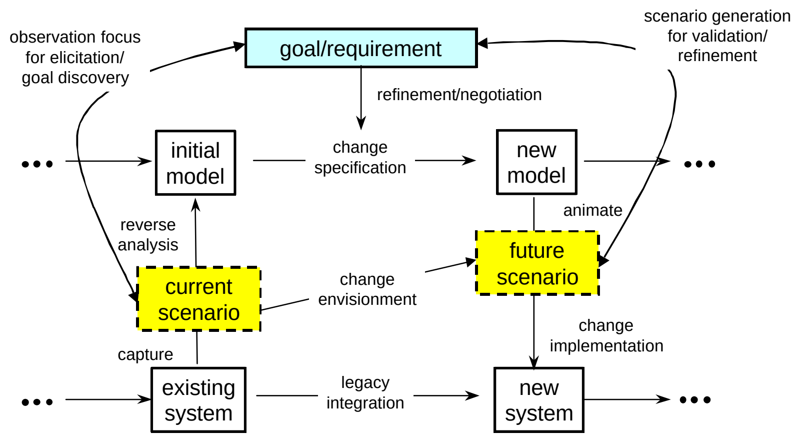

Figure 1: Change process with goals and scenarios

3 Scenario Research and Practice

The CREWS project has conducted surveys of scenario research and practice, with an emphasis on the requirements engineering task within software and systems engineering. To structure the analysis, the project followed an approach which perceives an information system to comprise four interacting basic perspectives or ”worlds” (Jarke et al. 1992). As a product (figure 2), an information system can be modelled as a human-machine *system* which provides *users* information or control over a *subject* domain (often called Universe of Discourse) which is denoted by the information objects. Users can be studied in two complementary roles: as individuals with cognitive problems of understanding, and as social organisations exploiting the information system as a communication and coordination medium to support their tasks, interests, formal roles, etc.

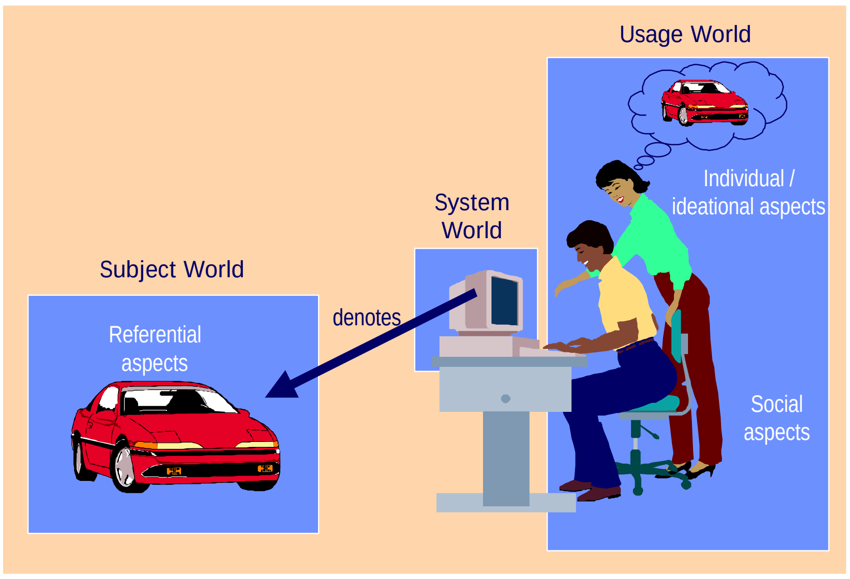

Figure 2: Conceptualization of an information system

This product triple <*system world*, *usage world*, *subject world*> is subject to an evolutionary change process in the *development world*. The development world is best understood as a meta-level *change information system* (figure 3). It controls the product information system as its subject domain, has the development team as its users and the development environment with its intermediate artefacts as the system itself. Scenarios are a particular kind of design artefact in the development world, intended to facilitate shared understanding of the target system, its interaction with users and subject domain, and its larger context.

A review of the scenario literature (Rolland et al. 1998) showed that this framework also provides a good starting point for classifying scenario-based approaches. Looking at the work activity as the subject domain and scenarios as one kind of development system artefact, we obtain four views (figure 4):

- What part of the work activity is captured in a scenario (content view) ?
- How is it represented in the development environment (form view) ?
- For what usage in the design process is it captured (purpose view) ?
- How is it developed and evolved (life-cycle view) ?

This framework also serves as a basic structure to manage knowledge about scenario-based approaches in a method repository (cf. section 5, below). In (Rolland et al. 1998), each of these four basic views is further elaborated into detailed facets. The framework has been applied to classify more than a dozen well-known proposals in the literature, including, for example, Jacobsen's initial Use Case approach and various proposed extensions.

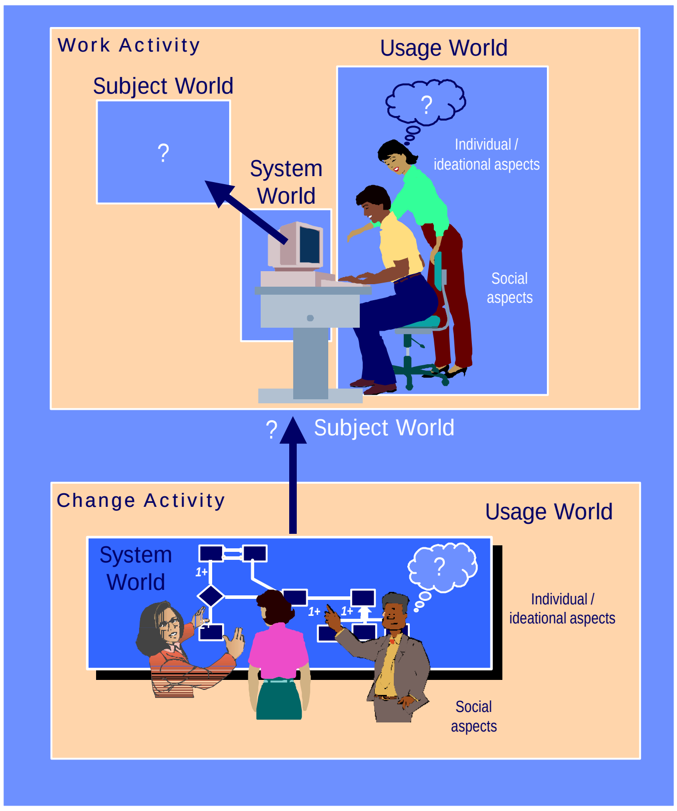

Figure 3: Change management as a meta information system

The framework of figure 4 was also elaborated in a set of questionnaires and semi-structured interviews used to determine the *state-of-practice* in scenario-based software engineering (Weidenhaupt et al. 1998). More than 25 projects, studied in part by the CREWS partners themselves, in part jointly with the RE group within the German GI (Arnold et al. 1998), were investigated this way, covering a variety of project sizes and application domains. The results show insufficient overlap between research and practice, asking for re-orientation on both sides.

While researchers, focusing on the *form view*, investigate scenarios as formal mediators between detailed traces and class-level specifications (Hsia et al. 1994), practitioners rarely use formal scenario representations. However, they would like to treat textual scenarios more formally and complain about a lack of guidance in authoring text scenarios.

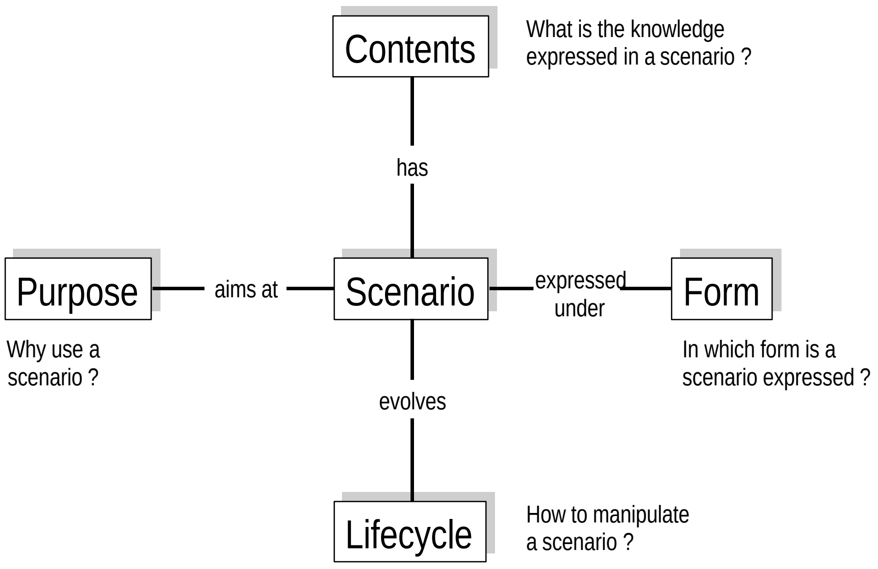
Contents
What is the knowledge expressed in a scenario ?
has
Purpose
aims at
Scenario
expressed under
Form
Why use a scenario ?
In which form is a scenario expressed ?
evolves
Lifecycle
How to manipulate a scenario ?

Figure 4: The CREWS framework for describing scenario-based approaches

Figure 5 gives an overview concerning the *content view*. Scenarios can address an organisational *work context* (C), they can represent the internal *interplay of components* within the current or future system (A), or – the most frequent case – they can focus on the *interaction* between the system and its environment (B). Interaction scenarios, in turn, can be studied in an in-bound direction (*what constraints does the environment place on the system?)* or in an outbound direction (*what impact has the system on its environment?).* Inbound interaction scenarios are called *blackbox scenarios* if they do not consider system internals; combinations of interaction with internal scenarios are called *whitebox scenarios*.

Scenario purpose and impact showed much more variation than expected from the research literature. While researchers discuss the application of scenarios for making abstract models understandable, to reach partial agreement and consistency, practitioners in the survey also reported scenario usage for task decomposition in complex projects, as a linkage between development phases, and as design aids and boundary conditions for object models.

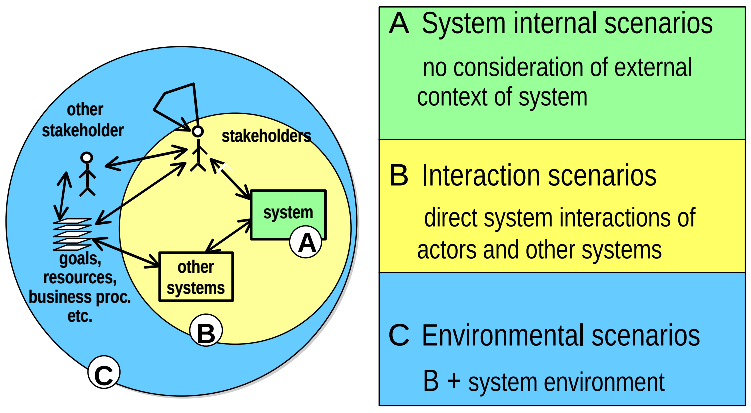
other stakeholder
stakeholders
system
A
goals, resources, business proc. etc.
other systems
B
C
A System internal scenarios
no consideration of external context of system
B Interaction scenarios
direct system interactions of actors and other systems
C Environmental scenarios
B + system environment

Figure 5: Types of scenario content

Consequently, *the life-cycle* of scenarios found in practice is much more involved than addressed by current research. The framework in figure 1 covers a broad variety of possible methodologies. Many software companies follow an informal development cycle that contains just general goals and future scenarios, but no conceptual models. On the other extreme, formal scenario techniques in management science often abstract reality to the values of a few key variables and strategic events. In between, UML has adapted Jacobsen's (1995) approach, which groups a *collection of inbound interaction scenarios* (expressed as message trace diagrams or collaboration diagrams) into a *use case* for manageability. However, as figure 2 shows, this definition of scenarios is clearly too narrow. For example, practitioners also employ use cases for managing internal scenarios of technical systems, e.g. in telecommunications.

Many large projects consider scenario selection, structuring and evolution as key unresolved issues. Multiple views on scenarios (e.g. developer, user and manager view on the same scenario) and the traceability of scenarios across project phases (e.g. interplay between scenarios and prototypes, elaboration of scenarios into test cases) still await solid solutions. Finally, methodological advice when to embed what kind of scenario technique into traditional methods, based on sound cost-benefit analysis of scenario usage, is one of the most crucial topics to be addressed when the vision of scenario-integrated methodologies such as promoted by UML is to become a reality. In the remainder of this paper, we discuss how the CREWS project has addressed these issues.

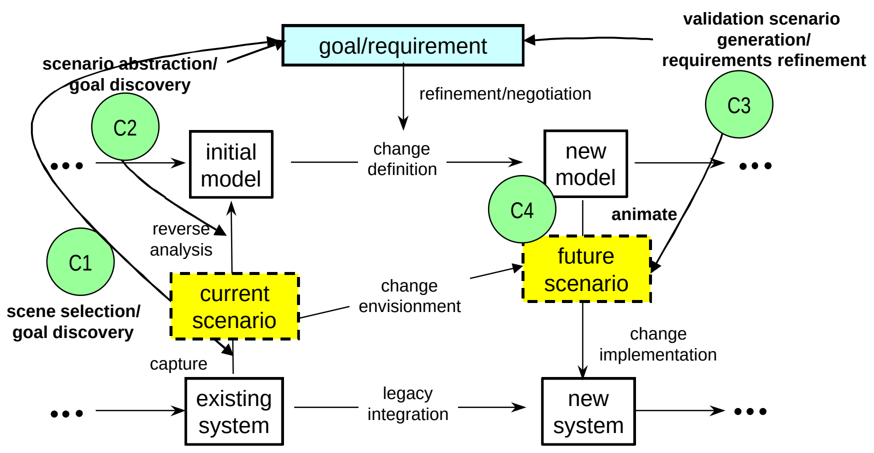

Figure 6: Positioning the CREWS method/tool components in the framework of figure 1

4 The CREWS Tools

Following the empirical studies, the CREWS project has developed prototypical solution components for some of the most pressing issues identified in industry. Four such components have been developed. The first two focus on requirements elicitation, the last two on requirements validation (cf. figure 6 for an overview):

1. **Scenario-extended traceability**: In model-based approaches such as UML, the background information on which modeling decisions are based is quickly forgotten. Moreover, traceability of scenarios throughout the lifecycle was identified as a critical issue by almost all industry projects. The C1 component extends traceability support in a process-integrated development environment back to the requirements sources. It includes an editor for real-world scenes captured in multimedia, and links the capture of these scenes to the goal/requirements hierarchy in a kind of FMEA infrastructure (Haumer et al. 1998). At the product level, the linkage from scenes to more formal scenario descriptions is provided by an advanced editor for message sequence diagrams.
2. **Authoring support for text scenarios**: Users of object-oriented approaches complain about lack of authoring guidelines (content guidelines as well as style guidelines) for writing use cases and scenarios. By a combination of authoring patterns and interactive natural language understanding, the C2 component (nick-named *L'Ecritoire* after the famous student cafe at Sorbonne university) supports not only the structured text presentation of scenarios but also their content analysis. This process leads to a formal

knowledge base of models, but also to the discovery of new goals in the text of the scenario (Rolland and Ben Achour 1997).

3. **Systematic generation of validation scenarios**: Another critical issue is the question of coverage: how many scenarios are enough to characterise the requirements to a system? This question can only be answered with respect to reference domain knowledge. However, typical reference models focus on the normal case, plus maybe a few most likely exceptions (e.g. Scheer 1994). In contrast, the exploration of relevant exception scenarios is one of the most important tasks of risk analysis in requirements engineering. The CREWS-SAVRE toolkit (component C3) has synthesised the literature on non-functional requirements (performance, reliability, security, user-friendliness, ...) into small reusable patterns of possible exception scenarios which can perturb normal-case scenarios, thus stimulating new requirements. The collections of possible exception scenarios, each linked to typical recovery mechanisms, are delivered as Excel spreadsheets which allow what-if analysis concerning the occurrence and treatment of each exception type. The resulting requirements are then stored in standard RE tools such as Requisite Pro. This way, not only inbound scenarios of system usage and system environment but also outbound analyses of system impact can be conducted (Sutcliffe et al. 1998).
4. **Cooperative animation as a validation tool for distributed systems**: In complex distributed systems with many overlapping scenarios, traditional scenario delivery mechanisms do not give an adequate overall picture of system behavior and impact since they neglect the interference effects. The C4 component offers a management-game like animation for the specification of distributed work scenarios defined in the formal requirements modeling language Albert. Albert can be understood as an agent-oriented extension to UML. The cooperative animation helps stakeholder groups identify problems in their specification and better envision the planned change (Dubois and Heymans 1997). Guiding the animation requires, in principle, complex theorem-proving based on temporal logic. However, careful domain analysis enabled us to replace general theorem provers largely by pattern-based checking mechanisms which focus specifically on problems typically found in distributed cooperative work applications.

Industrial evaluation of all four components is underway. In addition, the four tools are being grouped into two larger-scale demonstrators. One shows a text-based requirements elicitation and validation cycle based on the components C2 and C3 – a direct extension of the Use Case approach in OOSE (Jacobson 1995). The other demonstrates a multimedia-based cooperative elicitation and validation approach, based on components C1 and C4. In the following subsections, we briefly sketch both these demonstrators.

4.1 Requirements Elicitation and Validation Based on Text Scenarios

This demonstrator closes a gap in the object-oriented software engineering approach initially proposed by Jacobson (1995) and now being brought into the UML effort. This gap is fourfold:

- The process by which use cases are selected, and by which scenarios are developed from them, is only vaguely defined.
- Systematic guidance what specific scenarios to elaborate within a given use case is completely missing.
- There is no systematic procedure how to validate the use cases and scenarios against the requirements, and how to feed the results back in order to expand the scenarios, and to refine or correct the requirements.
- Use cases and scenarios are hardly supported by present UML-oriented tools.

Our solution to these issues is shown in figure 7. It relies heavily on several hundred knowledge patterns of different kinds. Coherent with the framework in figure 1, it starts with a set of high-level goals/requirements represented in a hierarchical structure such as offered by standard RE tools such as Requisite Pro.

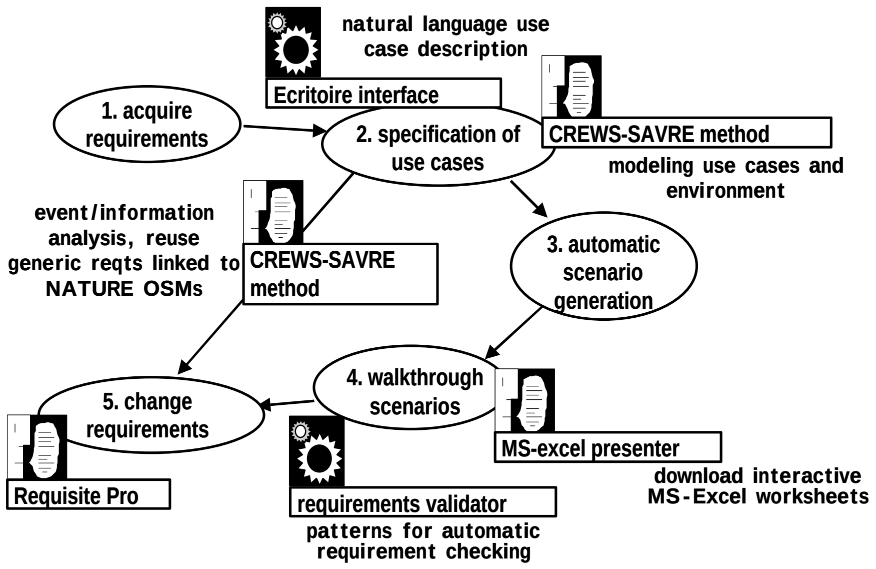
natural language use case description
Ecritoire interface
1. acquire requirements
2. specification of use cases
CREWS-SAVRE method
modeling use cases and environment
event/information analysis, reuse generic reqts linked to NATURE OSMs
CREWS-SAVRE method
3. automatic scenario generation
4. walkthrough scenarios
5. change requirements
MS-excel presenter
download interactive MS-Excel worksheets
Requisite Pro
requirements validator
patterns for automatic requirement checking

Figure 7: Text-oriented scenario-based requirements elicitation and validation

Following the guidelines provided by *L'Ecritoire*, initial use cases are developed from the initial goals in natural language. The integration of pre-existing use case texts is possible with some more effort; yet another option would be the adoption of standard scenarios from a business process reference model in the domain in question. The analysis of these texts is guided by patterns of structured natural language (case frames), as well as of domain knowledge (object system models and goal models). The result is not only an initial set of normal-case base scenarios but also their internal representation in the mentioned formalisms.

This internal representation of a scenario, typically comprising a few dozen steps, is now reflected against patterns of exception types gained from the literature on non-functional requirements such as performance, reliability, user-friendliness, and the like. Based on the user's judgement, and possibly further stored domain knowledge, about the relevance of specific non-functional requirements in the given application, a family of exception scenarios is generated for the use case, and stored for (a) validating and refining the present set of requirements and (b) regression testing during future requirements changes.

For the validation walkthroughs, a compact representation of the resulting scenario family can be downloaded onto an Excel spreadsheet, annotated with a suggested walkthrough method (again based on patterns) which helps the developer compare the scenarios systematically with the existing requirements. The resulting requirements changes are stored back into the requirements tool, together with traceability to the scenarios that caused these changes.

This approach has been successfully applied to requirements processes within several companies including DHL and GEC-Marconi.

4.2 Requirements Traceability and Change Envisionment Based on Multimedia Scenes

In technical or media-oriented domains with complex system interfaces such as found in computer-integrated engineering and manufacturing, white-box or even black-box interaction scenarios involve several different users and system components. It is very hard to understand the interplay between these agents from a formal specification or even from the interaction scenarios of a single user with the system, as described in the use case approach.

An animation of the specification, as supported by component C4 above, is an important first step to improve understanding and facilitate validation in the stakeholder group. However, unless such an animation is very much tailored and thus expensive, it will still be rather abstract with respect to the present reality experienced by stakeholders. The CREWS-EVE demonstrator therefore links the animation environment with the C1 component that provides traceability back to real-world scenes captured in multimedia.

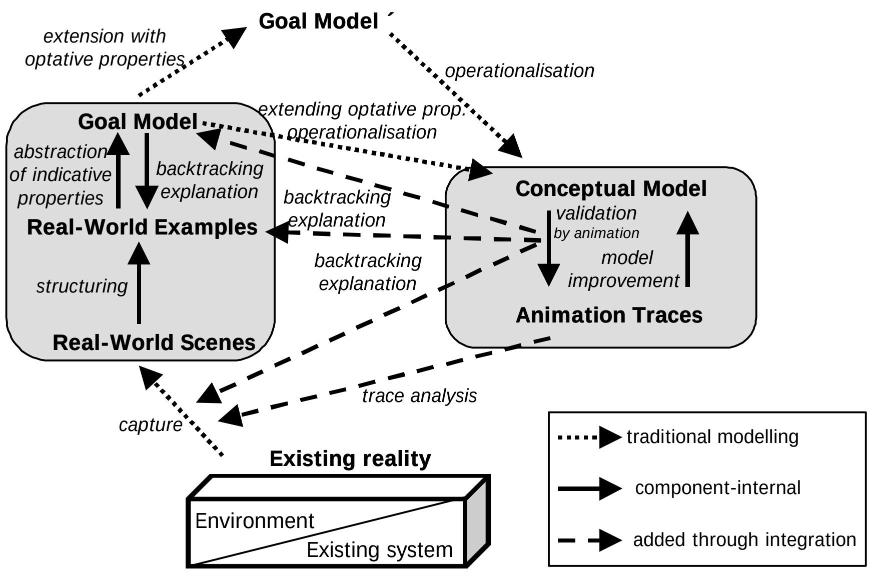

Figure 8: Functionalities of the CREWS-EVE scene-oriented demonstrator

Figure 8 displays the services gained from this integration in the context of the framework of figure 6. In the left dark box, the C1 components assists in the goal-oriented, cooperative and traceable abstraction of requirements and scenario structures from captured real-world scenes. In the right box, C4 offers animation of cocneptual models. In between, existing goal-oriented modelling techniques such as proposed by (Mylopoulos et al. 1992) are used for mapping goals to conceptual solution models, and for evolving the current-system goals towards the new system goals by adding desirable properties and deleting unwanted aspects. The dashed lines indicate the specific, explanatory services offered by the integration of both components. Positive experiences with a process along these lines (with somewhat simpler hypertext tool support) have recently also been reported in (Kaindl 1998).

The screendump in figure 9 offers an illustration how CREWS-EVE actually supports these options, taken from a trial application in the ADITEC gear factory at RWTH Aachen. The task is to validate one of the goals in the annotated goal tree (upper left of the figure) by animating the conceptual model derived from it (a distributed system specification partially visible in the right upper box). When several stakeholders cooperatively play this animation, they hit a surprising new feature they do not remember from their experience with the old system. They therefore invoke the traceability to the real-world scenes underlying the development of this system feature (via the goals). One of the three relevant real-world example fragments (video clips excerpted from the real-world scenes during the initial analysis) reveals that a worker actually missed this feature in the system, and lost valuable time through a manual work-around. This justifies the surprising extension of requirements whose consequences became visible in the animation.

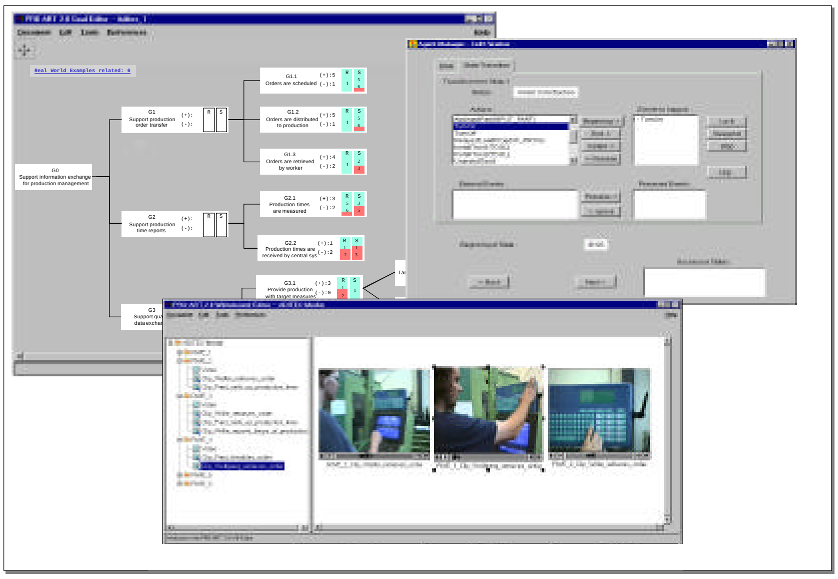
Real World Examples related: 6
G0
Support information exchange for production management
G1
Support production order transfer
(+):
(-):
R
S
G1.1
Orders are scheduled
(+):5
(-):1
G1.2
Orders are distributed to production
(+):5
(-):1
G1.3
Orders are retrieved by worker
(+):4
(-):2
G2
Support production time reports
G2.1
Production times are measured
(+):3
(-):2
G2.2
Production times are received by central sys.
(+):1
(-):2
G3.1
Provide production with target measures
(+):3
(-):0
G3
Support qua
data exchar

Figure 9: Screendump of the CREWS-EVE demonstrator including multimedia editor, goal editor, and animation guidance tool

The status of stakeholder agreement about goals, the availability of real-world scene evidence for and against goals, and the process status of the goal itself (proposed, reviewed, agreed, ...) are all visible in the coloured annotations shown in the individual goals of the goal editor in the upper left. This focuses management attention on those goals where either further work or explicit decision-making is needed. The linkage of scenes and scenarios directly to goals, rather than indirectly via conceptual models, has proven more flexible and work-efficient in our experiences and once more highlights the need to complement object-oriented IS engineering techniques by a systematic requirements engineering approach which goes beyond the simple listing of use cases and interaction scenarios.

5 Summary and Outlook

In this paper, we hope to have demonstrated that the applicability of scenario techniques extends far beyond what is covered by the standard use case approach in object-oriented systems engineering. We presented a framework in which these different uses can be positioned at the rough level. The CREWS project has used this framework for a comprehensive study of literature and practice in scenario-based requirements engineering even though space restrictions forced us only to sketch the main results here.

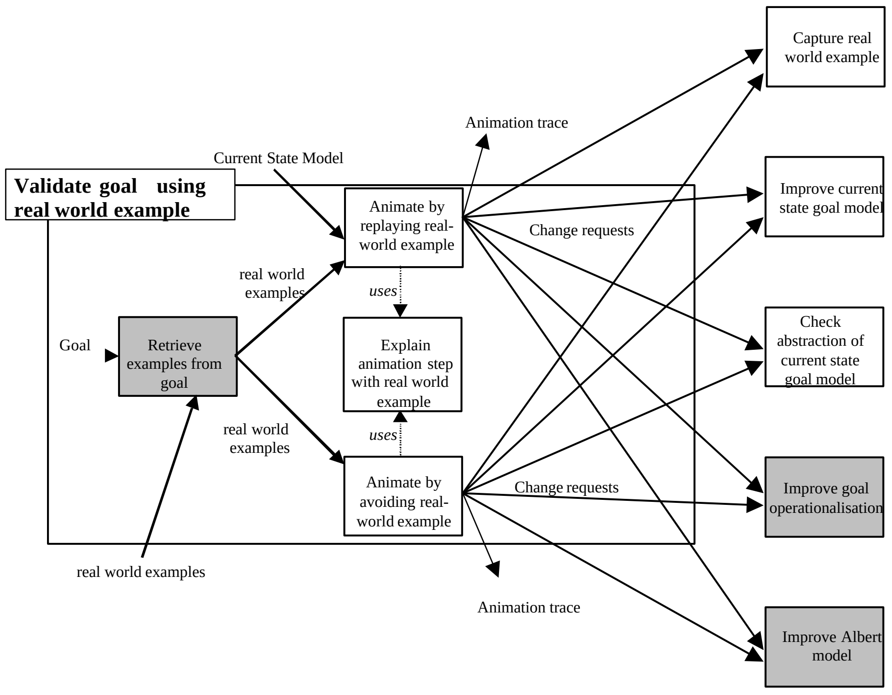
Validate goal using real world example
Current State Model
Animation trace
Capture real world example
Improve current state goal model
Check abstraction of current state goal model
Improve goal operationalisation
Improve Albert model
Animate by replaying real-world example
Change requests
real world examples
uses
Goal
Retrieve examples from goal
Explain animation step with real world example
real world examples
uses
Animate by avoiding real-world example
Change requests
real world examples
Animation trace

Figure 10: Composite method chunk describing the usage setting demonstrated in figure 9

For some of the most critical unsolved problems, the CREWS project has developed prototypical methods and tools which are currently undergoing industrial trials. Early experiences indicate that significant progress over the present state-of-the-art is indeed possible with reasonable effort, and there is strong commercial interest in some of the tools, as indicated above.

To make the methodological part of the experiences available beyond individual demonstration or industrial uptake, an additional effort is required. Using the requirements engineering process model developed in the NATURE project (NATURE Team 1996), an Internet-based method server is set up which describes important chunks of scenario-oriented process experience and linka them to the basic process model offered in OOSE textbooks. A first set of about 40 such process chunks is reported in (Plihon et al. 1998). Figure 10 shows a simplified version of such a process chunk, describing the method behind the example in figure 9. This method server, following a component-based approach, will also be open to other groups doing research in scenarios, thus assisting cumulative and experience-based systematisation of our knowledge how scenarios can used effectively and efficiently in information systems engineering.

Acknowledgement

This work was supported by the European Commission via ESPRIT Long Term Research project 21.903 (CREWS). Thanks are due to the task leaders Klaus Pohl, Colette Rolland, Pierre-Yves Schobbens, Alistair Sutcliffe and to numerous co-workers and members of the industrial advisory board.

References

Arnold, M. et al. (1998): Survey on the scenario use in twelve selected industrial projects. Aachener Informatik Berichte 98-7 (submitted for publication).

Carroll, J.M. (ed.) (1995): Scenario-based design: Envisioning work and technology in system development. New York: John Wiley and Sons.

Dubois, E./ Heymans, P. (1998): Scenario-based techniques for supporting the elaboration and the validation of formal requirements. CREWS Report 98-15, Universite de Namur, Belgium.

Fowler, M./Scott, K., (1997): UML Distilled – Applying the Standard Object Modeling Language. Addison-Wesley.

Haumer, P./Pohl, K./Weidenhaupt, K. (1998): Requirements elicitation and validation with real-world scenes. IEEE Transactions on Software Engineering, Special Issue on Scenario Management, December 1998.

Hsia, P./Samuel, J./Gao, J./Kung, D./Toyoshima, Y./Chen, C. (1994): Formal approach to scenario analysis. IEEE Software, March 1994, 33-41.

Jacobsen, I. (1995): The use-case construct in object-oriented software engineering. In J.M. Carroll (ed.), Scenario-based design: Envisioning work and technology in system development. New York: John Wiley, pp. 309-336.

Jarke, M. (ed.) (1998): Special Issue on Interdisciplinary Uses of Scenarios, Requirements Engineering Journal 3, 3/4.

Jarke, M./Bui, X.T., Carroll, J. (1998): Scenario management – an interdisciplinary perspective. Requirements Engineering Journal 3, 3/4.

Jarke, M. Kurki-Suonio, R. (eds.) (1998): Special Issue on Scenario Management, IEEE Transactions on Software Engineering 24, 12.

Jarke, M./Mylopoulos, J./Schmidt, J.W./Vassiliou, Y. (1992): DAIDA – an environment for evolving information systems. ACM Trans. Information Systems 10, 1, 1-50.

Kaindl, H. (1998): Combining goals and functional requirements in a scenario-based design process. To appear, Proc. HCI 98.

Mylopoulos, J./Chung, L./Nixon, B. (1992): Representing and using non-functional requirements: a process-oriented approach. IEEE Trans. Software Eng.18, 6, 483-497.

NATURE Team (1996): Defining visions in context: models, methods and tools for requirements engineering. Information Systems 21, 5, 566-592.

Plihon, V. et al. (1998): A reuse-oriented approach for the construction of scenario-based method chunks. Proc. Intl. Conf. Software Process, Chicago 1998.

Rolland, C./Ben Achour, C. (1997): Guiding the construction of textual use case specifications. Data & Knowledge Engineering, to appear.

Rolland, C. et al. (1998): A proposal for a scenario classification framework. Requirements Engineering Journal 3, 1, 23-47.

Scheer, A.-W. (1994): Business Process Engineering – Reference Models for Industrial Enterprises. Springer-Verlag.

Sutcliffe, A.G., Maiden, N.M./Minocha,, S./Manuel, D. (1998): Supporting scenario-based requirements engineering. IEEE Transactions on Software Engineering, Special Issue on Scenario Management, December 1998.

Weidenhaupt, K./Pohl, K./Jarke, M./Haumer, P. (1998): Scenario usage in software development: current practice. IEEE Software, March 1998, 34-45.