CREWS Report 98-28

An Integration of Scenario-Based Requirements Elicitation and Validation Techniques

by

Peter Haumer*, Patrick Heymans**, Klaus Pohl*

(*): Lehrstuhl Informatik V, RWTH Aachen
Ahornstraße 55, 52056 Aachen, Germany
+49 (0)241 80 21 501
{haumer,pohl}@informatik.rwth-aachen.de

(**): Institut d'Informatique, University of Namur
Rue Grandgagnage 21, B-5000 Namur, Belgium
phe@info.fundp.ac.be

An Integration of Scenario-Based Requirements Elicitation and Validation Techniques

Peter Haumer*, Patrick Heymans** and Klaus Pohl*

(*) Lehrstuhl Informatik V, RWTH Aachen
Ahornstraße 55, 52056 Aachen, Germany
{haumer,pohl}@informatik.rwth-aachen.de

(**): Institut d'Informatique, University of Namur
Rue Grandgagnage 21, B-5000 Namur, Belgium
phe@info.fundp.ac.be

Abstract

Scenarios are widely used in industry to support requirements elicitation and validation. Tool support is a prerequisite for effective use and management of scenarios.

In this paper we propose to integrate two scenario-based techniques developed in the ESPRIT Project CREWS. One supports the elicitation of a goal model from real-world scenarios. The other allows to validate a formal system specification by playing instance-level scenarios against it. Interestingly, the two of them complement each other in various respects.

We outline the benefits obtained by integration and present how we performed it at the conceptual and technical levels. The resulting integrated environment has been validated on a small case study in the manufacturing domain.

1 Introduction

Scenario-based approaches are widely used within industry [28]. Research literature offers an increasing number of scenario-related methods, models and notations which highlight the consideration of concrete examples during requirements engineering. The most popular form of scenarios is probably Jacobson's OOSE approach [16] and various extensions, e.g. [24] or [6]. To get a better understanding of the diversity of scenarios and their usage, the ESPRIT Reactive Research Project CREWS[1] has developed a scenario classification framework [25] based on a comprehensive literature survey. The framework was used to classify eleven prominent approaches. As the classification indicates, the scenario-based approaches vary in form, purpose, content, and lifecycle. To complement the more research-oriented classification framework, the CREWS team has undertaken 15 site visits to industrial projects in four European countries. As the results indicate the usage of scenarios during requirements engineering is much richer than anticipated from the literature survey [28].

One main shortcoming identified in the survey was the lack of adequate tool support for scenario applications and management. In the CREWS project we have developed scenario-based techniques and tools for requirements elicitation and validation.

[1] CREWS: Cooperative Requirements Engineering with Scenarios, ESPRIT Reactive Long Term Research 21.903

In this paper we present an integration of two scenario-based techniques. One focuses on requirements elicitation based on captured real world scenes, the other focuses on requirements validation through the animation of formal specifications.

In Sect. 2 we outline our integration and describe the advantages gained. The individual techniques are sketched in Sect. 3 (elicitation) and Sect. 4 (validation), respectively. The conceptual and technical integration of the two existing tool environments is described in Sect. 5. Besides the data integration achieved through an IRDS-based repository and the definition of several reusable integrated process parts, an integration of the environments was achieved by adapting the PRIME (*PR*ocess-*I*ntegrated *M*odelling *E*nvironment) framework proposed in [22].

We finally report on a trial application of the integrated environment in the manufacturing domain (Sect. 6) and provide conclusions and outlook on future work (Sect. 7).

2 Overall Approach

Following Jackson [15], we consider that the components of a composite system are the elements of the environment into which a new *machine* has to be introduced. In this respect, Jackson attributes two kinds of properties to a composite system:

- ♦ properties that the environment intrinsically possesses, *in spite of the machine* to be introduced (the so-called *indicative* properties) and
- ♦ properties that we hope the environment will satisfy after the introduction of the machine (called *optative* properties). Properties of both kinds are expressed in terms of the components of the composite system, that is the environment of the future system.

Consequently, a model always has an indicative portion, referring to properties of the sole environment, and an optative portion, referring to properties of the environment that the system should enforce.

The RE team typically starts by defining a model of the environment, which comprises a description of the existing system. They then progressively and repeatedly change the model by identifying new indicative and optative properties (future requirements) and improving their definitions.

2.1 Overview on the Integration

In the following, we outline the integration of techniques for improving the modelling of indicative and optative properties (see Fig. 1 for an overview). Each individual technique is depicted as a box being able to perform its own tool- and method-supported activities.

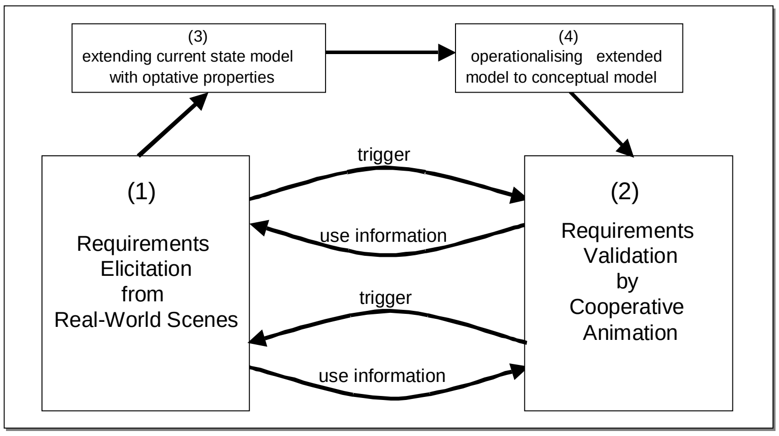

(3)
extending current state model with optative properties
(4)
operationalising extended model to conceptual model
trigger
use information
trigger
use information
(1)
Requirements Elicitation from Real-World Scenes
(2)
Requirements Validation by Cooperative Animation

Fig. 1: Overview of Integrated Approach

The elicitation technique (box ① of Fig. 1; see Sect. 3 for details) identifies indicative properties of existing systems and their environments based on examples observed and recorded in real world usage situations. It provides support for producing hierarchical goal models from the examples capturing the rationale behind the observed systems. While doing this, fine-grained traceability interrelations between the abstract goals and fragments of the real world examples are established. These will be used in the elicitation as well as in the integrated technique.

Due to the fact that conceptual models are often too complicated and written using notations stakeholders like domain experts rarely understand, we also support cooperative validation of conceptual models (box ②; see Sect. 4 for details). This is performed by role-playing stakeholders who elaborate possible behaviours of the model as interactive animations. During the animation several behavioural paths can be explored in an experimental manner. The result is a set of animation traces representing examples for the conceptual model enacted during the animation.

During the ongoing RE process performed by various stakeholders, the indicative properties identified during elicitation will be extended and changed to fulfil additional optative properties (box ③). This process, which is not considered to be supported by the work presented in this paper, leads from the current state to a desired state model.

The desired state goal model resulting from this process is then concretised by operationalisation (box ④). The results from this phase are (i) the indicative properties which remain the same with respect to the existing reality (i.e. those pertaining to the environment and to the subset of the current system's properties that should not change) and (ii) the optative properties identified by goal-oriented analysis. How to go from goals to a more fine-grained representation of properties in a conceptual model is not a topic of this paper as well. For this, we rely on other research and, in particular, to what is reported in [7]. More specifically, if the target model is to be written using the Albert II language (see Sect. 4.1) we refer to [9].

The requirements engineering process however, is not as streamlined as described so far. Rather, during elicitation the requirements engineering team has, for example, the desire to animate parts of the current state model to check and improve the accuracy of the abstractions. During validation, the team might want to backtrack to real world scenes from the elicitation process to better understand and explain indicative properties during animation. Therefore, the integration must provide ways for one technique to trigger activities of the other. When activities of the other technique are triggered, these can either be performed locally, i.e. using actions the standalone technique offers or they can use newly defined integrated activities in which one technique uses information provided by the other. (E.g., elicitation uses animation traces, animation uses real world examples etc.) Triggering validation from elicitation and vice versa is what we call elicitation-validation cycles.

2.2 Expected Benefits of the Integration

The integration of the two techniques outlined above improves the support offered to the requirements engineering team for scenario-based elicitation and validation in four main ways:

Real world scenes provide explanation during animation. Real world examples can be retrieved for explanation purposes of abstract concepts in the animation. For instance, during animation a client user has to make a decision about which action to perform next. The descriptions of the offered actions are unclear to him. In a situation like this, he is able to inspect real world scenes (e.g. captured on videotape) which show him example situations in the existing system for each action.

Animation of current state model allows to check accuracy and consistency of abstractions. Although Abstraction Guides provide support for the conceptualisation of real world observations, it still depends on the requirements engineer's *subjective* interpretation of the real world aspect he wants to capture. To check and discuss his abstractions with other stakeholders he can, on the one hand, use real world scene fragments which have been linked to the concepts by the Abstraction Guides, but on the other hand he is able to animate the current state model. While Abstraction Guides provide direct access to recorded example situations of the real world for a concept the animation additionally allows to explore different variants of the example not performed during observation. This can be useful for the analysis team to uncover forgotten cases but also when it was not possible (too expensive, time consuming etc.) to perform and/or observe them. It is also easier to just *play around* with the current state model in the animation to explore if those use cases have been captured. People who might be distributed over wide areas in the real world application can perform the animation with their normal roles allocated in a single room and discuss their actions right away.

Animation initiates focused elicitation from real world scenes. Traceability relationships can also be used for further analysis of the real world scenes. E.g., when problems are discovered, like missing data or functionality in the current state model, tracing back the originating real world scenes leads to a focused elicitation of the missing knowledge if it was captured.

Animation initiates focused capture of real world scenes. As stated above, animation can be used to validate the accuracy and completeness of the current state model. The real world scene based creation of the abstract models can be driven by several elicitation and validation cycles for evolutional model improvement. One aspect of these cycles is that animation can lead to the discovery of open issues of the current state model, which have to be *re-tackled*, for instance by another site visit and recording of information. The discovered problems in the animation can be used as a starting point for a focused site visit for resolving the unclear issues.

3 Elicitation of Conceptual Models Based on Real World Scenes

McMenamin and Palmer [19] argue that one has to consider the history and functionality of the existing system when building a new system (see also Gause and Weinberg [11]). There are two main reasons for this: a) the new system has to provide to a large degree the functionality of the old system; b) one can learn a lot from the success stories and pitfalls of the existing system; thereby to make failures twice could be avoided. The quality of conceptual models (and this is especially valid for the quality of current state models) heavily depends on the successful stakeholder involvement in the requirements engineering process. To achieve better involvement of different stakeholders the use of rich media (e.g. video, pictures, screen dumps, speech etc.) to record and discuss current system usage is proposed, e.g. by participatory design techniques [2], [4], [17].

Because the material gathered during an observation session can contain information about many system usages, we propose to pre-structure this material into what we call coherent *real world examples (RWE)*. A real world example is a collection of all the different material collected using different types of media that represents **one** system usage. However, several different examples of system usages can represent different incarnations of one task fulfilling one specific goal. We therefore elicit and relate goals to the examples (and parts of the examples) because goal models focus on a more abstract level why certain system properties are constructed rather than how [1]. When the *whys* behind the existing system are understood one can go on modelling critical or problematic details using more concrete modelling languages (e.g. UML Static Structure Diagrams [23], Message Sequence Charts [14] etc.).

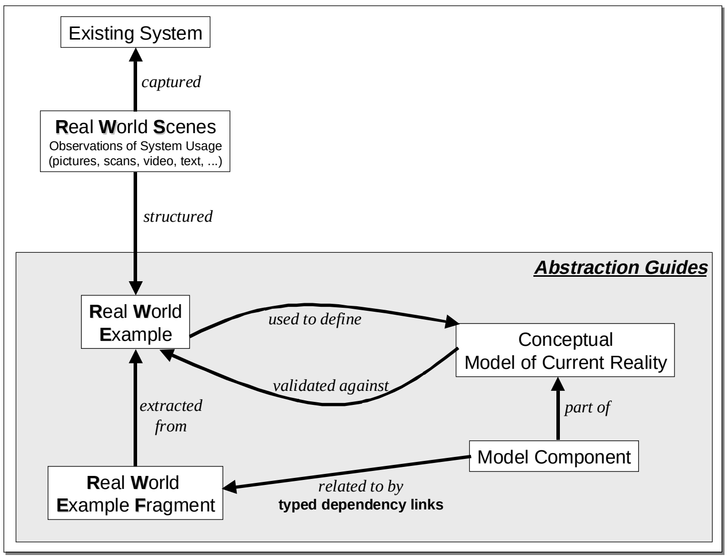

Existing System
captured
Real World Scenes
Observations of System Usage (pictures, scans, video, text, ...)
structured
Abstraction Guides
Real World Example
used to define
Conceptual Model of Current Reality
validated against
extracted from
part of
Model Component
Real World Example Fragment
related to by typed dependency links

Fig. 2: Abstraction Guides

To systematically support these activities we introduced so called *Abstraction Guides* (cf. Fig. 2 for an overview), which support the requirements engineer in the definition of a current state model based on real world examples. For Abstraction Guides real world examples are used for two main purposes: First, new concepts are elicited from them. Further, the elicited concepts are validated against other real world examples. In both cases, the Abstraction Guides support the typed interrelation of concepts of the current state model with corresponding fragments of the RWE which caused their definition respectively which were used for their validation.

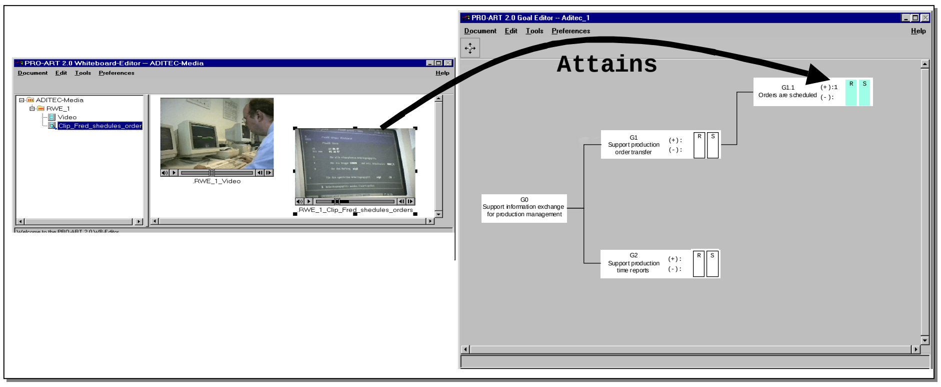

PRO-ART 2.0 Whiteboard-Editor – ADITEC-Media
Document Edit Tools Preferences Help
ADITEC-Media
RWE_1
Video
Clip_Fred_shedules_order
RWE_1_Video
RWE_1_Clip_Fred_shedules_orders
PRO-ART 2.0 Goal Editor -- Aditec_1
Document Edit Tools Preferences Help
Attains
G1.1 Orders are scheduled (+):1 (-): R S
G1 Support production order transfer (+): (-): R S
G0 Support information exchange for production management
G2 Support production time reports (+): (-): R S

Fig. 3: Interrelating an RWEF with a Model Component

Fig. 3 shows a simple example of our tool environment. In the left window of the screen dump one can see two videos from which the left one represents an RWE showing a complete example captured solely on videotape. The right video represents a fragment (RWEF) of the left one. It has been cut out of the example by marking a temporal beginning and ending point and is now displayed as an individual object. The RWEF has been linked to a model component; in this case a goal of a hierarchical goal model, using a dependency of type `Attains`. The link type `Attains` expresses that the analyst interprets the fragment as an example of how the related goal is fulfilled. Likewise, we support for goal models the link type `Fails`, which expresses the direct opposite to `Attains`, namely that the example shows how a goal is failed (cf. [18] for discussion on further link types).

As explained above, the `Attains` relationship of Fig. 3 could have been established during two possible activities following two different objectives: (1) the linked goal has been created because of the example (elicitation objective); or (2) the goal has been compared against the example (validation objective). In a typical analysis session however, validation and elicitation are often heavily intertwined. Whenever the analyst encounters problems in validating a conceptual model against the RWEs it leads to the elicitation of new or alternative concepts.

Consequently, the established structure imposes on the one hand access paths upon the real world examples; i.e. a set of real world example fragments typed by the incident links is gained from initially totally unstructured multimedia material. On the other hand, the goal model is annotated by a set of real world evidences that are close to the perception of the involved stakeholders. This structure can now be used for[2]

- explanation of the current state model with the real world examples, which caused their creation in a navigational manner, for people not trained in conceptual modelling or to ease the training of people joining the project, but also to assure a better model understanding for all stakeholders during the whole system development lifecycle;
- comparison of different real world scenes in respect to the goals they fulfil. We are able to support comparison of different real world examples using the goal model which describes the objectives to be achieved by the system;
- reviewing of conceptual models by other stakeholders in a structured way helping the reviewer in justifying and commenting on the abstraction made by the model builder and comparison of multiple viewpoints created during reviewing and further refinement by different stakeholders.

[2] see [18] for details on all topics as well as further applications

4 Cooperative Animation of Formal Models

Because a model is usually too abstract, it is almost impossible to get a detailed understanding of the model by simply reading it, even extremely carefully. This situation is reinforced if the application domain is the one of complex/critical systems. Having a way for the stakeholders to experience and visualise behaviours defined by the model is therefore a crucial issue in validation. This is what animation is intended to provide. The language supported by our animator (Albert II) and the animator itself are in turn briefly presented below.

4.1 The Albert II Language

An Albert II model is used to express (indicative and/or optative) properties of a composite system. The adopted modelling paradigm is an agent-oriented one: classes of components of the composite system are modelled as *agents* which can be hierarchically grouped into *societies*. Agents can perform *actions* and have *state components* denoting either physical or informational characteristics. Actions can be used to denote exchanges of information between components and/or how components can affect each other's state components. An Albert II model is also made of a series of constraints aiming at restricting the number of admissible behaviours to the intended ones (depending on the indicative and optative properties that one wants to model). In order to provide more guidance to the analyst, these constraints are written by instantiating predefined *constraint patterns*. Each of the latter is useful in expressing a specific kind of property: preconditions, effects of actions, actions compositions, invariants, visibilities, etc. The editing of an Albert II formal description is supported by the CAT-Edit Tool (which has been designed within the CAT project). It basically provides a graphical editing environment, syntax checking facilities and a client-side interface for invoking more elaborate checks.

In order to be reasoned on automatically (amongst other things, to be animated), the language is given its formal semantics by an established mapping of its constructs to an underlying real-time temporal logics called Albert-KERNEL. For a detailed description of the Albert II language, see [10].

Regarding method support for writing Albert II descriptions, we have investigated progressive formalisation using intermediate semi-formal description like MSCs [8] and E-R diagrams [29], [25].

4.2 Animation of an Albert II Model and Tool Support

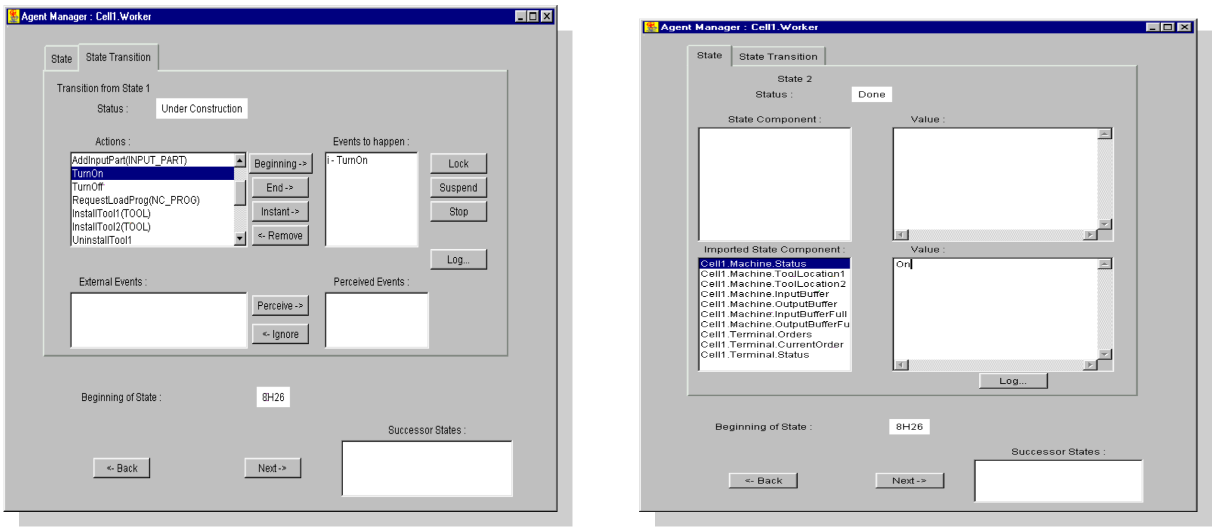
Albert II specifications are class-level descriptions: they describe abstract properties of classes of components of a system. The Albert II animator [12] helps to construct instance-level behaviours step-by-step in a distributed and co-operative way. Each participant is assigned some *agent instances* he will manipulate during the animation (each stakeholder animates the components he is the most competent to validate). The simulation of the overall behaviour of the composite system results from tasks that the stakeholders perform in cooperation, allowing to validate interactions between components. The manipulation of agent instances is performed by the participants through a graphical client-side application, called *agent manager*. Each such application allows to browse through the already constructed behaviours to create and observe the action occurrences taking place at definite steps; similarly, agent managers allow to create and observe state component values of the agent instances[3] in the various steps (see Fig. 4). Browsing through steps is done using the backward and forward arrows.

The role of the animation server (which supports the task of the coordinator user) is to conduct an animation by centralising the various requests issued by the stakeholders and to build a new global state of the system (made of all the local instances' states). Noteworthy is that the resulting state may be inadmissible with respect to the specification. This results in a dead-end branch which cannot be further investigated. Such states are especially marked both in the *agent managers* and on the co-ordinator's display. The other possible origin of a dead-end branch is when some stakeholder has decided to stop building the current state transition either because he was prevented to perform some event or because some events he performed in previous stages now have unforeseen consequences. Any time an anomaly is discovered, an error message is displayed in the *agent managers*. In order to facilitate revisions, traceability within the animator allows such error messages to refer directly to the

[3] Actually, only initial states can be constructed by the users. Since Albert II's underlying formal semantics has a *frame axiom* [3], changes in states can only be due to events which have happened in previous transitions.

violated statements. It is also possible for a user to attach to any component of a state or state transition, built or being built, a comment, question or a change request. This additional information is recorded and added to the animation traces, which can then be a basis for discussion and model improvements.

An additional advantage of animation is that, since operations are carried out by stakeholders over intangible objects (the representations of the composite system's components inside the animator tool), some restrictions about the kind and the amount of the tested operations disappear. For example, testing how operations are carried out in a manufacturing environment might be very costly in the real-world if real parts have to be produced. Another advantage of animation comes from the settings in which an animation session usually takes places (distributed but in the same room) and is that informal communication between stakeholders is possible. This is often not possible when trying out a real system where operations are performed in different places. Finally, over prototyping, animation has the advantage that it avoids to create an implementation and that it keeps therefore validation at the requirements level (since no design or implementation concerns interfere).



Transition from State 1
Status : Under Construction
Actions :
AddInputPart(INPUT_PART)
TurnOn
TurnOff
RequestLoadProg(NC_PROG)
InstallTool1(TOOL)
InstallTool2(TOOL)
UninstallTool1
Events to happen : i - TurnOn
Beginning of State : 8H26

State 2
Status : Done
Imported State Component :
Cell1.Machine.Status
Cell1.Machine.ToolLocation1
Cell1.Machine.ToolLocation2
Cell1.Machine.InputBuffer
Cell1.Machine.OutputBuffer
Cell1.Machine.InputBufferFull
Cell1.Machine.OutputBufferFu
Cell1.Terminal.Orders
Cell1.Terminal.CurrentOrder
Cell1.Terminal.Status
Value : On
Beginning of State : 8H26

Fig. 4: Transition and state view of one animation step

5 Technical and Methodological Integration

Both techniques presented in the last two sections define complex methods for the elicitation as well as the validation of requirements. Consequently, integration of both methods as outlined in Sect. 2.1 has to be performed on the methodological level, i.e. by defining integrated method parts. This means that a data integration of both environments would not be sufficient. In addition to data integration a composite methodical technique has to be defined and supported. By the methodical integration, each existing environment is empowered to make use of support offered by the other environment. This allows complex tool spanning activities with branches, backtracks, loops etc.

To realise the methodical integration we used an existing framework introduced in [22]. The framework provides mechanisms for data and method integration.

In the following sections, the integration is outlined. In Sect. 5.1 we describe how data integration is achieved. Sect. 5.2 describes the method integration part. Finally, in the Sect. 5.3 we describe the realisation of the process-integrated tool support.

5.1 Data Integration

We achieved the data integration by applying the meta-modelling repository framework developed in the ESPRIT project NATURE which had been defined based on the IRDS standard [13] (see [20] for detailed descriptions of the IRDS standard as well as the formal NATURE meta-models). The resulting IRDS hierarchy is outlined in Fig. 5. We defined our conceptual modelling languages, like the Albert II specification language or the goal modelling language, on the IRD Definition Level as instances of a common IRD Definition Scheme Level Model. Concrete models are represented as instances of the language constructs on the next level the IRD Level (e.g. the ADITEC Machine Terminal (MT) specification or the goals identified for the ADITEC system). Performing an animation will again instantiate the model components with concrete, as we call them, model lives; e.g., in Fig. 5 the concrete set-up performance by agent Michael.

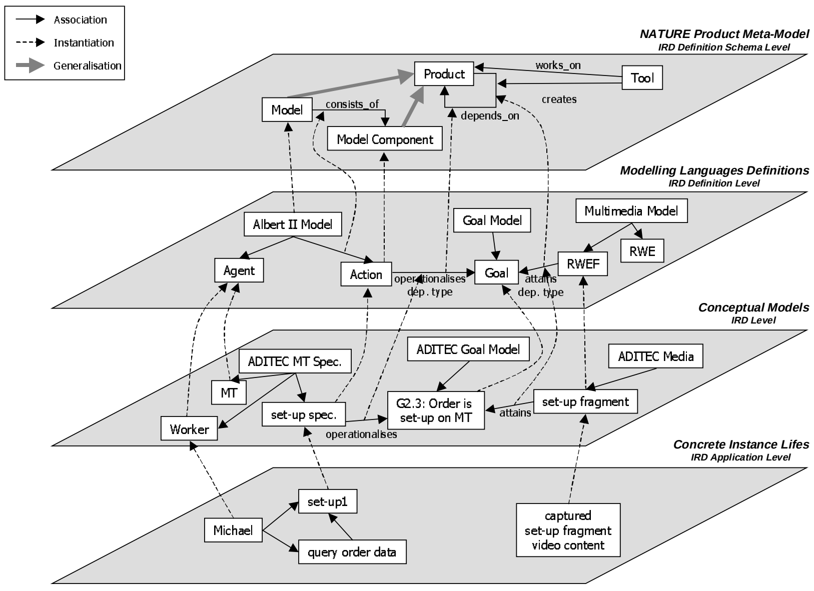


Fig. 5: IRDS-Based Product Integration for Method Chunks

For integration of parts of the different modelling languages, we defined a set of dependency relations between specific product types. Thus, the method chunks described in the following subsections will primarily use instantiations of the IRD Definition Schema Level concept `depends_on` defining these relations between `Products` (on the IRD Level!). For instance, the dependency relation type `Attains` already introduced in Sect. 3 as the integration mechanism between goal models and real world examples is defined on the IRD Definition Level and instantiated on the IRD Level linking a goal concept (Goal G2.3) with a real world example fragment (set-up fragment) showing an example for the attainment of this goal. However, different types of dependencies are defined for the integration between Albert II concepts and goals. E.g., as we outlined for the overall RE process in Sect. 2, goals are operationalised to concrete specification concepts. These concepts are linked to those goals by the `operationalises` dependency type, which we see instantiated in Fig. 5 for example between the Albert II `set-up specification` and `G2.3`. A third group of dependency types are defined between current state models and extended models to be able to trace adoption and changes of resp. on concepts.

The dependency links are heavily employed by the method chunks described below. For instance, the `operationalises` link of Fig. 5 is used by the *explain* method chunk in the following way: During the animation of the existing system, the stakeholder who controls the animation of the client representing worker Michael wants to obtain more information about the operation `set-up`. To display a real world example from the existing system of a set-up the method chunk explanation-support follows the dependency link of the set-up specification object (IRD Level) of type `operationalises` leading to goal G2.3 which in turn is linked to a real world example fragment containing a video clip showing a worker at the set-up procedure.

5.2 Integrated Method Chunks

The basic building block for method integration is the notion of method chunk. A method chunk is a modular reusable set of activities. They are introduced because of the fact that it is not possible to predefine methodical process guidance for the whole RE process, but only for specific, well-understood parts [21], [27]. Thus, a method chunks provides guidance for the well-understood parts. Consequently, we defined our individual as well as the integrated guidelines and processes using these method chunks and meaningful combinations of them.

As presented in Sect. 2, we make the individual techniques able to trigger each other when necessary (going from validation to elicitation and going from elicitation to validation), making the overall requirements engineering process look like a set of elicitation-validation cycles. More precisely, a chunk defining how to carry out an operation with one technique might request that a whole set of operations is performed with the other (a new elicitation session or a new animation session). Performance of the requested operations is improved if information is drawn from what was done in the calling technique. Therefore, we have improved the individual techniques with chunks that are able to make use of information provided in the other one. We will start this subsection by presenting these chunks. Then, we will describe a larger chunk showing an example of how the elementary chunks can be combined in a useful manner. This more coarse-grained method chunk will illustrate (i) when there can be a need for an approach to call the other one and (ii) how to combine lower-level chunks with each other, with chunks already defined in isolation within the techniques and with external chunks to obtain an effective method support.

5.2.1 Elementary Method Chunks

In the following we describe and motivate the chunks.

Animation chunks that make use of elicitation information:

- **EMC 1: Explain an animation step by using a real-world example.** At some point during an animation, a client user might wonder what is the right action to perform. The name of the actions and of their parameters at his disposal might not be very meaningful to him/her. Similarly, what Albert state components denote might not be so easily understandable from their name. If such elements are part of the existing reality, if they were captured on videotape and if traceability links were established between the Albert model and real world example (via goals), then the integrated environment supports explanation of Albert concepts by visualisation of the corresponding captured real world example. Guidance on how to reach the right real world example from the animation (i.e. how to follow the right traceability links) is provided by this chunk.
- **EMC 2: Animate by replaying real-world example.** In order to test if the current model accepts some behaviour that occurs in the current real system, benefit can be taken from the real world example that captures such a behaviour (if there is one). Going through the real world example as one progresses through the animation is supported by this chunk. The chunk makes use the explanation chunk presented above to go through traceability links.
- **EMC 3: Animate by avoiding real-world example.** It is also useful to animate behaviours that were not captured during elicitation. These might have been omitted during the elicitation or maybe they were taken into account but it was not possible to film them. Another possibility is that they are not behaviours of the current system (either unwanted or wanted for the future system). In any case, to elaborate such 'alternative' behaviours, one may find help in referring to the other (captured) ways of performing the same task (accomplishing the same goal). Such a support is provided by this chunk during the animation and also makes use of the explanation chunk above.

Elicitation chunks that make use of animation information.

- **EMC 4: Capture real-world example by using animation traces.** Performing real-world observation in early elicitation can only be done in a non-focussed way. But, as a requirements model evolves, more focussed observation sessions can be carried out. Such focussed elicitation can be initiated if during animation one discovers that some behaviours existing in the current system were not captured. Animation traces may therefore contain requests for new observations. The goal of this chunk is to help the requirements engineer go through the traces and perform observation when needed.
- **EMC 5: Improve current state goal model by using animation traces.** Initially, the current state goal model is abstracted from real world observations using the Abstraction Guides. Some of

the abstracted behaviours are the observed ones and some others have to be inferred. Unfortunately, it may happen that the requirements engineers have not (correctly) inferred behaviours of the latter type. On the other hand, when doing animation, behaviours that should have been inferred can be played by the users. The impossibility to animate them reveals a problem in the abstraction process. Therefore, traces of the animation where such situations happen are a precious indication to improve the requirements model.

- **EMC 6: Check abstraction of current state goal model by using animation traces.** One could also decide to take all the animation traces and check goals of the current state goal model against them. Depending on whether the animated behaviour exists in the current real system or not and depending on whether the animation failed or not, one can decide whether the abstraction of the goal model is a faithful representation of the existing system. In case something has to be changed in the abstraction the chunk "Improve current state goal model by using animation traces" should be invoked.

The method chunks have been formally described using the notation defined in [25]. For the sake of brevity, we have only described them informally here.

5.2.2 A Combined Method Chunk: Validation of Current State Goal Using RWE

The more elementary method chunks depicted above can be combined to larger chunks within methodological support to validation and elicitation. As an example of this, we present a complex combined chunk providing support for performing the validation of a goal from the current state goal model with the animator. The chunks introduced in this paper are represented with white boxes while the grey boxes represent the chunks that already existed in the individual techniques or *external* ones.

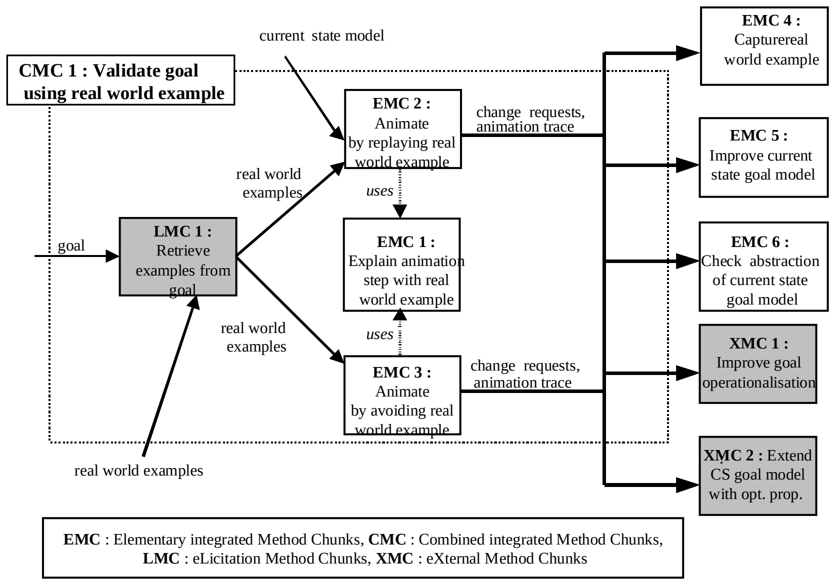

current state model
CMC 1 : Validate goal using real world example
EMC 2 : Animate by replaying real world example
change requests, animation trace
real world examples
uses
goal
LMC 1 : Retrieve examples from goal
EMC 1 : Explain animation step with real world example
real world examples
uses
EMC 3 : Animate by avoiding real world example
change requests, animation trace
real world examples
EMC 4 : Capturereal world example
EMC 5 : Improve current state goal model
EMC 6 : Check abstraction of current state goal model
XMC 1 : Improve goal operationalisation
XMC 2 : Extend CS goal model with opt. prop.
EMC : Elementary integrated Method Chunks, CMC : Combined integrated Method Chunks,
LMC : eLicitation Method Chunks, XMC : eXternal Method Chunks

Fig. 6: Process chunk for validating goal using real world example

Triggering a goal-directed animation (see Fig. 6) comes from the need to validate a goal from the goal model. It is with the satisfaction of this goal in mind that the users of the animation will then try to elaborate behaviours within the tool. But in order to know more precisely which behaviours to try out, they have to retrieve the real world examples associated to this goal. To support this latter task, we chose to reuse the LMC1 chunk that initially only supported browsing in the elicitation approach. Then, one has to consider one of the following alternatives: (1) animate the retrieved examples or (2) animate other behaviours than the captures ones, that is, trying out alternative ways of satisfying the goal. Executions of both animation chunks can result in change requests that trigger one or more of the EMC4, EMC5 and EMC6 elicitation chunks. Finally, it may be that the change request can only be adequately handled using external chunks like XMC1 or XMC2.

5.3 Tool Integration

In this section we sketch our integrated prototype CREWS–EVE (Cooperative Requirements Engineering With Scenarios – Elicitation and Validation Environment) which comprises the method integrated modelling environment consisting of the elicitation tools described in Sect.3 and the validation tools presented in Sect. 4.

To be able to integrate the existing prototypes related to the two individual techniques and to support the method chunks described above, we needed an implementation platform which supports the dynamic definition and adaptability of the integrated chunk as well as the repository described in Sect. 5.1. The PRIME framework (*PR*ocess *I*ntegrated *M*odelling *E*nvironments, see [22] for a detailed description), which defines method knowledge in explicit process models, fulfils these requirements. The process integration provided by PRIME empowers the user to initiate the enactment of predefined method chunks guiding him through the activities of the chunks, even across tool boundaries.

The general PRIME framework is divided into three distinguishable conceptual domains. The *modelling domain* comprises all activities for defining and maintaining process models, i.e. method chunks, using a formal language with underlying operational semantics enabling the mechanical interpretation of the models. In addition, the modelling domain stores the repository definitions described in Sect. 5.1. The interpretation of the process models is done by the *enactment* domain which then guides and controls the *performance domain* comprising the set of actual activities conducted by the tools. The interactions between these domains characterise the way in which model-based method guidance is provided. Process chunks are first instantiated within the modelling domain and passed to the enactment domain. Based on the interpretation of the instantiated model, the enactment domain supports, controls, and monitors the activities of the performance domain by sending and receiving predefined message types between the domains. Thus, the performance domain has to provide feedback information about the current process status to the enactment domain as a prerequisite for adjusting process model enactment to the actual process performance and enabling branches, backtracks, and loops in the process model enactment.

Technically, the tools of the performance domain must be fully process-integrated. This can be achieved by implementing a tool by reusing the generic PRIME tool architecture (see [22] for a detailed description). The elicitation tools (the goal editor, the multimedia management tool, the dependency management tool) have been build based on the PRIME architecture. Thus, this tools are process adaptable. The validation tools (the Albert II editor and the animation tool) did not have the process-integration features required. Thus, they must be adapted to enable their integration according to the method chunks defined above.

To avoid a re-implementation of the validation tools, we focused on a partial process-integration first. This was achieved by implementing a wrapper that provides the validation tools with an API (Application Programming Interface) and empowers them to send and receive service calls from the enactment domain. Moreover, the wrapper provides the functionality for storing and retrieving data out of the common RDBMS based repository by which data integration is achieved. In addition, an extension to the PRIME environment was required to support the wrappers. Details on this can be found in [5].

However, it turned out that the partial integration had two major shortcomings. First, changes of the method definitions caused a re-programming of parts of the wrapper and the validation tools to enable method invocation and method-conform tool performance. Second, the adaptation of the tools interface of the validation tools according to the actual process situation could only be achieved by changing the implementations. Both drawbacks required a lot of re-design and consequently re-programming effort of the validation, which were in some cases almost impossible to achieve. In contrast, changes in the method definitions required no code change in the elicitation tools.

Due to the experiences gained, we are now convinced that a partial re-implementation of the validation tool would require less effort than their continuous adaptation. We will therefore re-implement part of the validations tools according to the PRIME architecture to make them fully process-integrateable in the future.

6 Example from Case Study

The ADITEC manufacturing company produces machine gears for various kinds of industrial devices. In the example, we show the system from the point of view of workers located at production cells. These have to fetch production orders displayed at the cell's machine terminal according to the planning-dispatching system which they have to feed back by reporting on the progress of the orders' production. Because of problems with inconsistent and erroneous data reporting, the management of ADITEC wanted the information flow from the planning-dispatching system to the production cell and backwards to be analysed and improved. We used the CREWS-EVE environment to analyse the indicative properties of the existing system, created current state models of critical parts and animated the current state model.

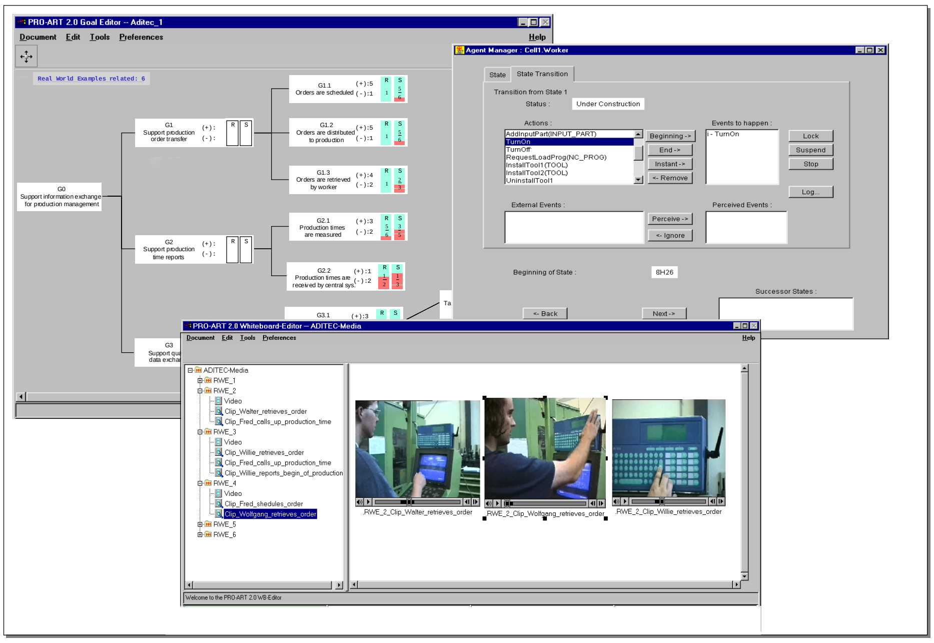

PRO-ART 2.0 Goal Editor -- Aditec_1
Document Edit Tools Preferences Help
Real World Examples related: 6
G0 Support information exchange for production management
G1 Support production order transfer (+): (-): R S
G1.1 Orders are scheduled (+):5 (-):1
G1.2 Orders are distributed to production (+):5 (-):1
G1.3 Orders are retrieved by worker (+):4 (-):2
G2 Support production time reports (+): (-): R S
G2.1 Production times are measured (+):3 (-):2
G2.2 Production times are received by central sys. (+):1 (-):2
G3 Support qua data exchar
G3.1 (+):3
Ta
Agent Manager : Cell1.Worker
State State Transition
Transition from State 1
Status : Under Construction
Actions :
AddInputPart(INPUT_PART)
TurnOn
TurnOff
RequestLoadProg(NC_PROG)
InstallTool1(TOOL)
InstallTool2(TOOL)
UninstallTool1
Beginning -> End -> Instant -> <- Remove
Events to happen :
i - TurnOn
Lock Suspend Stop Log...
External Events :
Perceive -> <- Ignore
Perceived Events :
Beginning of State : 8H26
Successor States :
<- Back Next ->
PRO-ART 2.0 Whiteboard-Editor – ADITEC-Media
Document Edit Tools Preferences Help
ADITEC-Media
RWE_1
RWE_2
Video
Clip_Walter_retrieves_order
Clip_Fred_calls_up_production_time
RWE_3
Video
Clip_Willie_retrieves_order
Clip_Fred_calls_up_production_time
Clip_Willie_reports_begin_of_production
RWE_4
Video
Clip_Fred_shedules_order
Clip_Wolfgang_retrieves_order
RWE_5
RWE_6
RWE_2_Clip_Walter_retrieves_order
RWE_2_Clip_Wolfgang_retrieves_order
RWE_2_Clip_Willie_retrieves_order
Welcome to the PRO-ART 2.0 WB Editor

Fig. 7: Explaining Actions in an Animation Using Goal Models and Real World Scenes

Fig. 7 shows a screen dump of using CREWS-EVE to perform the chunk CMC1 described in Sect. 5.2.2. In the goal editor (on the upper left side), the G1.3 goal has been selected: "Worker retrieves orders". Using the LMC1 chunk, the repository's dependency links between the selected goal and any real world example media objects were queried. The results are displayed in the lower window. We assume that operationalisation of the goal model has been carried out and resulted in an Albert II model which is now animated in the upper window. The analyst using the tool now asks a worker to assist him in performing animation. The analyst first decides to trigger the chunk EMC2. The Albert worker agent's *RetrieveOrder* action is performed within the animator. Consulting the next animation step reveals that the worker agent perceives a list of orders displayed on the terminal. No problem is therefore detected by executing this chunk. The analyst decides then to trigger the other animation chunk EMC3. The worker is requested to look at the examples and to tell if there is no alternative way by which he retrieves orders in the current system. He admits that he usually asks the foreman which order to do next. The lower windows shows that captured real world example do not exist for this situation. This means that if the situation was taken into account in the goal model and in the Albert model it was only by inferring it. However, no action corresponding to this situation is found in the action list of the worker's agent animation window. To be sure of that, explanation was used for all actions with unclear names. The chunk allowed to consult the observations corresponding to each of the actions listed in the animator but none of them showed a worker asking the foreman what to do next. In consequence, a change request is issued with the intention to trigger a new elicitation session in which the capture of the missing real world example will have to be done and attached to the goal.

7 Conclusions

Research reported in this paper takes as starting point two scenario-based requirements engineering elicitation and validation techniques and tool environments and investigates how to integrate them in a common tool environment with integrated method support. Elicitation based on real-world examples aims to identify goals underlying the current system from captured real-world phenomena. Validation is performed by cooperatively animating a model (resulting from the operationalisation of the goal model) against instance-level scenarios.

We have highlighted several ways in which these approaches complement each other. Elicitation provides explanation that helps the achievement of animations. Animation, on the other hand, can be useful in checking the correctness of abstractions, for additional analysis of real-world scenes and to carry out new observation sessions.

In order to take the best out of these complementarities, we have proposed fine-grained method support for the integrated environment which comes in addition to the method support which already existed for the two approaches taken in isolation. The proposed integration was threefold. First, the product models of each technique were integrated into an IRDS-based repository. The mutual relationships were formalised. Second, the process integration was performed by defining elementary method chunks. Third, practical tool integration (APIs, service calls, etc.) was done by wrapping the validation tools. Method adaptability of the latter was achieved by hardcoding the defined method chunks.

Moreover, the combination of the elementary method chunks empowered the definition of larger methodological guidance, which supports requirements engineering in the elicitation and validation cycles.

The integration has been validated on a small case study.

In the future, we will focus on the full process integration of the two techniques and the validation of the integrated environment on a larger trial application.

Acknowledgements. This work is founded by the European Community under ESPRIT Reactive Long Term Research 21.903 CREWS (Cooperative Requirements Engineering with Scenarios). The authors are also especially grateful to Luc Claes, Klaus Weidenhaupt, Markus Hoofe, Prof. Eric Dubois, and Prof. Matthias Jarke for participation to the research reported in this paper.

8 References

[1] Annie I. Antón. *Goal-based requirements analysis.* In Proceedings of International Conference on Requirements Engineering (ICRE'96), pages 136- 144. IEEE, 1996.

[2] Hugh Beyer and Karen Holtzblatt. *Contextual Design: Defining Customer-Centered Systems.* Morgan Kaufmann Publishers. 1997.

[3] A. Borgida, J. Mylopoulos and R. Reiter., *On the Frame Problem in Procedure Specification.* IEEE Transactions in Software Engineering, vol SE-21, n°10, October 1995.

[4] Francoise Brun-Cottan and Patricia Wall. *Using video to re-present the user*. Communications of the ACM, 38(5):61-71, 5 1995.

[5] Luc Claes, *Layered Components*. To appear in Component-based Information Systems Engineering Workshop (CBISE'98), Pisa, June 1998.

[6] Alistair Cockburn. *Structuring use cases with goals.* JOOP, September and November, 1997.

[7] A. Dardenne, A. van Lamsweerde and S. Fickas. *Goal-directed requirements acquisition.* Science of Computer Programming, 20:3-50, 1993.

[8] Eric Dubois and Patrick Heymans. *Scenario-Based Techniques for The Elaboration and the Validation of Formal Requirements.* Submitted.

[9] Eric Dubois, Eric Yu, Michael Petit. *From Early to Late Formal Requirements: a Process-Control Case Study.* In Proc. of IWSSD9, Isobe, Japan. April 1998.

[10] Philippe Du Bois. *The Albert II Reference Manual – Language Constructs and Informal Semantics*. Research Report RR-97-002, Computer Science Department, University of Namur. ftp://ftp.info.fundp.ac.be/publications/RR/RR-97-002.ps.Z, July 1997.

[11] Donald C. Gause and Gerald M. Weinberg. *Exploring Requirements: Quality before Design.* Dorset House Publishing, New York, 1989.

[12] Patrick Heymans, *The Albert II Specifications Animator*. CREWS Report Series 97-13. August 1997.

[13] ISO/IEC. *Information Technology - Information Resource Dictionary System (IRDS) framework*. International Standard ISO/IEC 10027. First edition 1990-06-15, 1990.

[14] ITU-T. *ITU-T Recommendation Z.120 — Message Sequence Charts (MSC)*, ITU, 1993.

[15] Michael Jackson. *Software Requirements & Specifications — a lexicon of practice, principles and prejudices.* Addison Wesley Press. 1995

[16] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Oevergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.

[17] Karen McGraw and Karan Harbison. *User-Centered Requirements: The Scenario-Based Engineering Process*. Lawrence Erlbaum Associates, Inc., Publishers, Mahwah, New Jersey, 1997.

[18] Peter Haumer, Klaus Pohl, and Klaus Weidenhaupt. *Abstraction Guides: Interrelating Conceptual Models with Real World Scenes*. In Fourth International Workshop on Requirements Engineering: Foundation for Software Quality (RESFQ), Pisa, Italy, June 8th-9th, 1998.

[19] Stephen M. McMenamin and John F. Palmer. *Essential System Analysis.* Prentice Hall, 1984.

[20] Klaus Pohl. *Process Centered Requirements Engineering.* RSP marketed by J. Wiley & Sons Ltd., England, 1996.

[21] Klaus Pohl, Ralf Dömges, and Matthias Jarke. *Decision Oriented Process Modelling.* In Proc. of the 9th Intl. Software Process Workshop, pp. 124-128, Arlie, Virginia, USA, IEEE Computer Society Press, Oct. 1994.

[22] Klaus Pohl and Klaus Weidenhaupt. *A Contextual Approach for Process-Integrated Tools.* In Proc. of the 6th European Software Engineering Conference (ESEC) and 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering, pages 176-192, Zurich, Switzerland, September 1997. LNCS 1301, Springer Verlang.

[23] Rational Software Corporation. *Unified Modeling Language.* Available on the World Wide Web: http://www.rational.com, Rational, 2800 San Tomas Expressway, Santa Clara, CA 95051-0951, USA, 1 1997.

[24] Björn Regnell, Kristofer Kimbler, and Anders Wesslen. *Improving the use case driven approach to requirements engineering.* IEEE Requirements Engineering '95, pages 40-47, 1995.

[25] Colette Rolland. *A Primer for Method Engineering.* Proceedings of the INFORSID Conference (INFormatique des ORganisations et Systemes d'Information et de Decision), Toulouse, France, June 10-13, 1997.

[26] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N.A.M. Maiden, M.Jarke, P. Haumer, K. Pohl, E. Dubois, and P. Heymans. *A proposal for a scenario classification framework.* accepted for Requirement Engineering Journal, Technical Report CREWS Report Series 96-01, ESPRIT Project CREWS 21.903, available at http://Sunsite.Informatik.RWTH-Aachen.DE/CREWS/, 1996.

[27] Colette Rolland and George Grosz. *A General Framework for Defining the Requirements Engineering Process.* In Proc. of the Intl. Conf. On Systems, Man, and Cybernetics, San Antonio, Texas, USA, IEEE Computer Soc. Press, Oct. 1994.

[28] Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke, and Peter Haumer. *Scenario Usage in System Development: A Report on Current Practice.* IEEE Software, March, 1998.

[29] Roel Wieringa and Eric Dubois. *Integrating Semi-Formal and Formal Software Specification Techniques.* Information Systems Journal, June 1998.