CREWS Report Series 98 - 22

A REUSE-ORIENTED APPROACH FOR THE CONSTRUCTION OF SCENARIO BASED METHODS

V. Plihon[*], J. Ralyté[**], A. Benjamen[**],
N.A.M. Maiden[+], A. Sutcliffe[+], E. Dubois[#], P. Heymans[#]

[*]GECT University of Toulon Var BP 132 83957 La Garde Cedex France plihon@univ-tln.fr
[**]CRI University of Paris 1 90 rue de Tolbiac 75013 Paris France {ralyte, benjamen}@univ-paris1.fr
[+]City University Northampton Square London, EC1V 0HB United Kingdom
{N.A.M.Maiden, A.G.Sutcliffe}@cs.city.ac.uk
[#]FUNDP, University of Namur, Rue de Bruxelles, 61, B-5000 Namur, Belgium
{dubois, heymans}@info.fundp.ac.be

Appeared in the proceedings of the International Software Process Association's
5th International Conference on Software Process (ICSP'98),
Chicago, Illinois, USA, 14-17 June 1998.

A Reuse-Oriented Approach for the Construction of Scenario Based Methods

V. Plihon*, J. Ralyté**, A. Benjamen**, N.A.M. Maiden+, A. Sutcliffe+, E. Dubois#, P. Heymans#

*GECT University of Toulon Var BP 132 83957 La Garde Cedex France plihon@univ-tln.fr

**CRI University of Paris 1 90 rue de Tolbiac 75013 Paris France {ralyte, benjamen}@univ-paris1.fr

+City University Northampton Square London, EC1V 0HB United Kingdom

{N.A.M.Maiden, A.G.Sutcliffe}@cs.city.ac.uk

#FUNDP, University of Namur, Rue de Bruxelles, 61, B-5000 Namur, Belgium

{dubois, heymans}@info.fundp.ac.be

Abstract

Despite the recent interest in scenarios, the development of new methods and tools for Requirements Engineering integrating scenario based approaches has been limited. This paper reports on four different processes developed from research undertaken as part of the CREWS project which the authors believe will improve scenario use and make it more systematic. Furthermore CREWS aims to integrate these approaches into a method for scenario-based requirements engineering. To achieve this objective and be able to include existing approaches such as use case analysis we develop a component based approach which reflects a shift towards a reuse-centric approach to method engineering. The paper presents CREWS method and meta-method knowledge through the implementation of an SGML database to store, retrieve and dynamically compose chunks of CREWS processes.

Keywords

Method Engineering, Process, Reuse, Requirements Engineering, Scenario, Method Construction.

1. Introduction

There has been considerable recent interest in the use of scenarios to acquire and validate system requirements. However, few processes, methods or software tools are available to achieve systematic scenario use, or even to generate useful scenarios in the first place. A recent analysis of scenario use by practitioners reveals that current commercial methods and software tools provide insufficient and unsystematic guidance [30]. This paper presents four different processes developed from research undertaken as part of the ESPRIT IV 21903 'CREWS' (Cooperative Requirements Engineering With Scenarios) reactive long-term research project which the authors believe will improve scenario use and make it more systematic. The paper also describes how these four processes have been synthesised together and integrated with processes developed outside CREWS into a more complete and integrated process for scenario-based requirements engineering. The paper ends with lessons learned for modelling large-scale software engineering processes, and in particular advantages from a more component-oriented approach to process modelling.

Despite the recent interest in scenarios, development of new methods and tools has been limited to a small number of academic institutions (e.g. [17]), research-oriented organisations (e.g. [8]), consultancies (e.g. [8]) and method vendors (e.g. [10]). Current software tools provide little prescriptive guidance for scenario use. To remedy this, CREWS aims to make scenario-based requirements acquisition and validation more systematic and hence more useful, widespread and cost-effective. It proposes four approaches which are being developed and evaluated in parallel: requirements acquisition from real-world scenes; acquiring requirements from natural language scenario descriptions [21]; systematic scenario generation and use to validate requirements [14], and scenario animation to validate requirements [9].

Furthermore, CREWS aims to integrate these approaches into a method for scenario-based requirements engineering. It draws on the ESPRIT 6353 'NATURE' basic research action's integration of different process modelling approaches into a comprehensive modelling framework [22]. However, NATURE lacked guidelines for integrating process chunks, deciding the granularity of these chunks, guiding and tracing the process, and defining standards for process chunks. CREWS, through the development of its method base, will have to develop new guidelines to overcome omissions in the NATURE framework.

This paper presents extensions to the NATURE framework to meet the needs of CREWS. It presents CREWS's method and method meta-knowledge through an implementation of an SGML database to store, retrieve and dynamically compose chunks of CREWS processes. Section 2 presents the CREWS approaches for guiding the different uses of scenarios during the acquisition and validation of systems requirements. Section 3 introduces the notion of a process chunk and chunk descriptor, presents the SGML structure of a chunk, and gives several examples of process chunks. Section 4 presents the CREWS process glossary to provide a basis for consistent description of process chunks. Section 5 presents the hierarchical organisation of the process chunks to facilitate chunk retrieval and composition. Section 6 presents how the four processes have been synthesised together and integrated with processes developed outside CREWS into a more complete and integrated process for scenario-based requirements engineering. Section 7 concludes the paper and discusses future problems to be overcome to produce an effective method for scenario-based requirements engineering.

2. The CREWS Approaches

The four CREWS approaches guide the different uses of scenarios during the acquisition and validation of system requirements. Two approaches guide requirements acquisition. Existing approaches do not give method guidance to represent rich descriptions, or scenes, of current system use and transform these descriptions into conceptual models. Therefore, the first approach guides co-operative elicitation of system requirements from scenes recorded in multimedia representations such as video footage and audio recordings, to elicit different types of model and system requirement from these representations. A method and prototype multi-media software tool supports this process. The second approach guides the semi-automatic extraction of system requirements from natural language descriptions of scenarios and use cases. The process encourages an author to use CREWS style and content guidelines when writing a use case/scenario to ensure both its completeness and correctness, as well as making the use case/scenario amenable to computational analysis. A software tool then applies a set of case frames to the use case/scenario description to extract candidate agents, objects, actions and system requirements.

The other two approaches guide requirements validation. Scenarios provide useful 'test scripts' for a requirements specification, however there is a lack of process guidance for systematic scenario generation and use. The third approach provides a method and software tool to generate useful scenarios then walk users through these scenarios, as well as to semi-automate the detection of missing or incorrect system requirements through computational analysis of these scenarios. The fourth process validates system requirements through the animation of scenarios derived from a formal specification of the system. This language is compatible with the ALBERT agent-based requirements modelling language, thus enabling easy inter-linking of requirements specifications, declarative non-deterministic scenario scripts and deterministic scenario execution traces as a basis for requirements animation and hence validation. This approach is also supported by a software tool.

One of the strengths of the four CREWS approaches is that they have been designed to complement each other. As a result, CREWS can offer a coherent scenario-based requirements engineering method with different techniques and software tools to achieve each process. Furthermore, this method also includes existing approaches such as use case analysis [10] and notations such as the UML [27]. Indeed, it is imperative to link it to existing methods to fill the gaps between the CREWS processes, or even to complement them such as through the application of CREWS use case authoring guidelines in the OOSE approach [10]. This shift towards a more 'component-based' process model reflects a wider shift in software engineering towards a more reuse-centric approach to systems engineering. Reusable process knowledge is bite-size rather than large-scale, reflecting the nature of human expertise. The development of such a method will, we hope, enable CREWS to evaluate its approaches on large-scale case studies.

However, several issues need to be resolved if we are to integrate approaches into an effective method. Three specific questions which the research reported in this paper addressed are: (i) what is the most effective size of reusable process chunks, (ii) how should we represent these chunks to facilitate reuse, and (iii) how should we organise these chunks to facilitate reuse ? In answering these questions, the CREWS method comprises: (i) definitions of a scenario-based approach as a collection of process chunks made available in the method base, (ii) glossaries to harmonise the terms used to describe these process chunks and to facilitate the retrieval queries formulation, and (iii) hierarchical organisation of these chunks to facilitate their retrieval.

The CREWS method base stores both *method knowledge* (i.e. the contents of the method chunk) and *method meta-knowledge* (i.e. the context in which to use the chunk) in tightly-coupled descriptions of each process chunk. The description of each chunk is faceted [19] and close to that of [2]. Each process chunk is described using the SGML (Standard Generalised Mark-up Language). The two knowledge levels of the method base are parts of the same SGML document in order to facilitate their joint manipulation. Our motivation for using SGML has been on the one hand, its ability to represent hyper-text documents and on the other hand, the availability of SgmlQL which is an SQL like language tailored to query SGML documents. The next section describes the SGML descriptions of the process chunks in more detail.

3. The structure of the CREWS method base

3.1 Overview of the structure of the CREWS method base

Figure 1 gives an overview of the knowledge stored about each process chunk in the CREWS method base. The *chunk body* is the description of the scenario-based process itself. The *chunk interface* describes preconditions which control the application of the chunk (e.g. actors who must be involved in the process), the current situation which is the input to the process chunk (e.g. the current state of the 'scenario' product) and the intention or goal that the chunk of process achieves (e.g. to validate system requirements). The chunk body and interface constitute the reusable knowledge about the process itself which is delivered to the person who enacts the process chunk. In contrast, meta-knowledge about reuse of the process chunk is in the *chunk descriptor*. It describes the design activities which the process chunk achieves (the situation part) and the design intention that can be supported by the scenario chunk (the intention part). This partial duplication of knowledge about the chunk's situation and intention is purposeful, because it enables to distinguish between the conditions of applicability of the chunk and the conditions under which it is reusable in design activities.

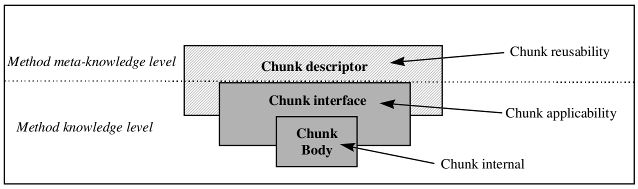

Figure 1: Chunk overview

3.2 The SGML structure of the CREWS method base

The SGML (Standard Generalised Mark-up Language) [7] is an international standard language to describe a document using a set of mark ups. SGML documents are structured as trees. SGML's query language, SgmlQL [12] enables a user to query the SGML method base. SGML uses the notion of composition to relate its elements. The root of the CREWS method base is the element DESCRIPTIONS which represents a collection of DESCRIPTIONs. The element DESCRIPTION is itself characterised by a METHOD_KNOWLEDGE_LEVEL and a KNOWLEDGE_ LEVEL which are composed of a DESCRIPTOR and of a CHUNK respectively. A chunk is considered as *atomic* when it reaches an intention which cannot be decomposed into more detailed intentions, on the contrary it is called *composed*. The SGML structure of the CREWS method base is the tree presented in Figure 2 .

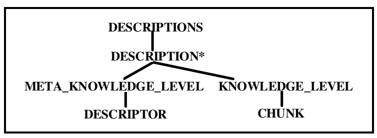

Figure 2: The top levels of the CREWS method base

3.2.1 **The structure of a chunk.** Our definition of a scenario chunk is based on the process view of the NATURE process modelling formalism [22], [15] and consistent with the notion of 'step' in [29]. According to this view a process transforms an initial *situation* into a result which is the *target* of the *intention* of the chunk. The *situation* represents the part of the product undergoing the process and the *intention* reflects the goal to be achieved in this situation. The *target* of the intention is the result produced by the process execution. As the target is embedded in the intention, this leads to the characterisation of a process by a *<situation, intention>* couple which is called *context.*

Several advantages of the NATURE formalism are discussed in [16]. As an example we can precise that the NATURE formalism allows to describe in a modular way chunks belonging to several levels of granularity , using a situational/intentional paradigm. The notion of choice used in NATURE allows to provide flexible guidelines to achieve a certain intention, and finally, the chunks can be formally described using a context algebra based on a rational language.

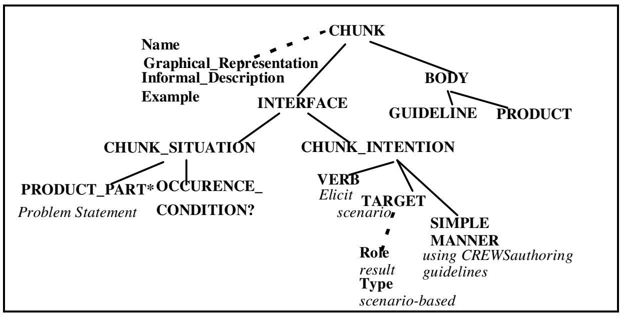

Figure 3: The Sgml scenario chunk structure

Following the NATURE view, a chunk has two parts (Figure 3): its *INTERFACE* which is the couple <CHUNK_SITUATION, CHUNK_INTENTION> and a *BODY.* We chose these designations by analogy with object descriptions in object oriented approaches. The *interface* is the visible part of the chunk. It tells us in which situation and for which intention the chunk is applicable. The *body* explains how to proceed to fulfil the intention in that particular situation. The body provides *guidelines* (GUIDELINE) to guide the process and relates the process to the *product parts* (PRODUCT) involved. For example, the interface of the scenario chunk which describes how to elicit a scenario using CREWS's author guidelines is the *couple <(Problem Statement), Elicit scenario using CREWS authoring guidelines>*, where the situation is defined by a product which is a *problem statement*, and the intention, *to elicit scenario using authoring guidelines*, uses information in this problem statement as input to the process. The target of the chunk of process, *scenario,* defines the result produced by the application of the chunk. The chunk's body describes how to define a use case out from the problem statement. Other attributes of a process chunk are *unique name*, *graphical representation of the process* and *informal description.* It is also possible to define *examples* of the process application.

The *situation* defined in the chunk interface (CHUNK_SITUATION) is composed of one or several *product parts* referenced by PRODUCT_PART* in the SGML tree, and an *occurrence condition* (OCCURRENCE_CONDITION) which is optional. All these elements are strings (#PCDATA).

The *intention* is described [18] in terms of a *verb*, a *target* (the product resulting from an application of the process chunk) and a *manner* (which describes how the intention is achieved). Examples of manners include 'one-shot refinement', 'stepwise strategy', 'using authoring guidelines'. Each target product part can either be an *object* or a *result.* An *object* already exists in the situation whereas a *result* indicates that the product is produced by the process. '*Refine scenario'*, is an example of intention in which the target '*scenario'* is an object because it is defined in the situation whereas '*Elicit scenario'* is an example where the target *'scenario'* is a result. It is developed during the execution of this intention. The precise notation of these intentions is as follows :

Refine (scenario)$_{Obj}$, and

Elicit (scenario)$_{Res}$ *(using CREWS authoring guidelines)*$_{Man}$.

This process chunk definition can be applied to model different types of software engineering method. However, the focus of CREWS is scenario-based requirements engineering, so often the situation or the target of the intention will make reference to products which are use cases or scenarios.

The *body* of a process chunk is decomposed into the product part and the guideline part. The product (PRODUCT) is characterised by a name, an informal description, an example of instantiation of the product and a reference to a graphical representation which is a picture stored in the SGML document. For sake of clarity, this attributes have not been represented on the above figure. The guideline (GUIDELINE) can be either represented by an informal description (INFORMAL_DESCRIPTION) or by a set of links (LINK*) depending on whether the chunk is informal or not. The body of a formal chunk is described following the NATURE approach, detailed examples of such a description can be found in [23], [25].

The Figure 4 illustrates the interface and the body part of the scenario chunk referred to with the intention *'Elicit scenario using CREWS authoring guidelines'*. In this example the scenario chunk named 'P13' helps to fulfil the intention *'Elicit scenario'*, for a given *'problem statement'*. The aid provided by the scenario chunk is stated in the guidelines attached to its body. These guidelines propose to first write a scenario from the problem statement either using style guidelines, or using contents guidelines, or under the control of a word processor. Then, the scenario may be reviewed in a clarification step. As stated in the result part of the intention, the output of the scenario chunk is a textual scenario.

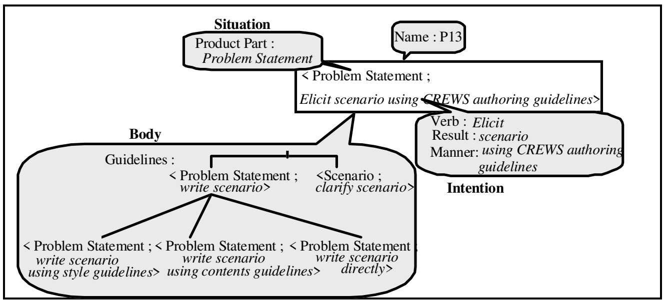

Figure 4: Example of scenario chunk from the CREWS base

3.2.2 **The structure of the descriptor.** The process chunk descriptor describes the meta-level knowledge to facilitate effective reuse of the chunk. It fulfils for a process chunk the same role as a meta-class does for a class. We extend the chunk's interface to structure the meta-knowledge in the descriptor to enable the retrieval of the right process chunk to achieve the right intention in the right situation. As you might expect, a chunk descriptor has a *situation* part and an *intention* part that we consider in turn, see Figure 5.

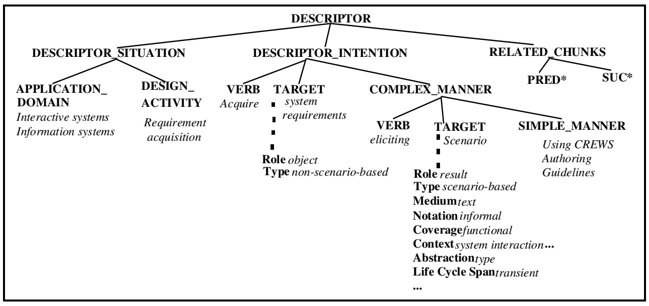

Figure 5: The SGML structure of the chunk descriptor

Indeed, we view the reuse process as being contextual : a user of the method base is faced to reuse situations at which he / she looks with some intention in mind. Therefore , the descriptor seeks to capture *in which situation a scenario chunk can be reused to fulfil which intention.*

The *situation* part of a descriptor (DESCRIPTOR_SITUATION) has two attributes. The *application domain* attribute defines the domain to which the chunk is applicable. In the process chunk shown in figure 4, the chunk to elicit a scenario using CREWS authoring guidelines applies to *Interactive systems or information systems*. The *design activity* attribute defines in which process the chunk applies, i.e. *requirements acquisition*. A design activity could be defined as a very general Requirements Engineering intention you try to fulfil in a certain application domain.

The *intention* part of a descriptor (DESCRIPTOR_INTENTION) expresses how the chunk may participate to the achievement of the design activity. For example the descriptor intention of the chunk P13 is *'Acquire system requirements by eliciting scenario using CREWS authoring guidelines'* as the process of eliciting scenarios is a means to achieve the design intention *to 'Acquire system requirements'*. The descriptor intention is an expression of the role that a scenario approach can play in a particular design activity.

The intention part is similar to the intention attribute in the interface part of the chunk. The intention of the descriptor is specified by the intention *verb*, the *target* of this intention, but a manner which is a *complex manner* (i.e. recursively defined as an intention). For example in the intention *Acquire system requirements by eliciting scenarios using CREWS authoring guidelines*, the manner (*by eliciting scenario using CREWS authoring guidelines)* is recursively defined as an intention (*eliciting*) with a result (*scenario*), a manner (*using CREWS authoring guidelines*).

CREWS has defined a glossary of process intentions which informs the definition of each chunk intention. However, for the CREWS method, there is one important addition to an intention. Because the method is predominately scenario-based, we have refined the *target* in the intention using a set of *facets* which defines properties of the scenario-based product, for example its formality, level of abstraction, and the nature of the interactions described in the scenario [26].

Finally, the descriptor relates the chunk ones which precede (PRED*) and the ones which follow it (SUC*).

4. The CREWS glossary

One problem with component reuse is semantic ambiguities in component descriptors (e.g. [20]). A partial solution for the CREWS method base is a glossary of terms which describe CREWS's different scenario-based approaches. The glossary enables a user to search for scenario chunks using synonyms from the glossary. The glossary itself has two parts. The first describes *process products*, for example "goal", "specification", "scenario" and "message trace diagram" (a synonym for which is a "sequence diagram"). The second describes *process intentions*, for example "elicit" and "validate". Each term in the glossary has a definition and synonyms. The current version has 14 basic terms to describe product parts and 30 terms to describe process intentions.

At the level of the product, one can find terms like *goal*, *specification*, *scenario*, etc (see the complete list in the appendix). All terms which are heavily used in the R.E. community but for which no standard terminology exists. As an example of definition, we can consider the following one :

> *Elicit* : the process of systematically obtaining from people new facts (scenarios, requirements) about the domain / business processes / the system under consideration ; (Syn) to acquire, to discover, to capture.

Because of the complexity of the part dedicated to intentions (see the Appendix), we have decided to structure them according to a hierarchy of intentions. This hierarchy has been defined by distinguishing among the different types of intentions one can have when undertaking an RE activity. We have considered five types of intentions (namely Elicit, Conceptualise, Document, Verify, Validate) and created the corresponding entries in the glossary. Elicit is presented above and the four other intentions are listed below with their associated definitions :

> *Conceptualise* : the process of systematically abstracting (existing or envisaged) real-world phenomena into models which highlight the essential aspects and hide the unimportant details (relative to the viewpoint taken) ; (Syn) to model, to abstract.
>
> *Document* : as opposed to *'to conceptualise'*, write down the product of activities such as analyse, compare, change, etc. (Syn) to describe, to specify, to record, to write.
>
> *Verify* : to get the product right ; (Syn) to attest, to check.
>
> *Validate* : to process of checking against stakeholders that the right product is being built ; (Syn) test, quality assure.

The rest of the intention part of the CREWS glossary is made of terms denoting intentions corresponding to basic activities. These terms, themselves, cannot be attached to a single high-level term type that they specialise. For example, the lower-level term 'to compose', whose the definition is the following,

> *Compose* : (to be composed of) to be formed of a group of parts ; (Syn) to assemble, to aggregate, to integrate, to combine.

could be used to denote intentions which specialise e.g. 'conceptualise' (like composing objects to form aggregations) or 'document' (like composing text and images to make an illustrated document). But, as we have seen in the previous section, writing an intention (either of the chunk itself or its descriptor) requires more than just a term : it requires the term, plus a result, plus a manner in which the result should be obtained. This manner can in turn be complex and again decomposed in the same way as an intention. This helps make the expression of the intention more precise. For example, one of the partners of the CREWS project has a chunk whose intention is *Conceptualise (System Requirements)*$_{Res}$ *(by Creating (Albert Specification)*$_{Res}$ *(using Generic Method)*$_{Man}$*)*$_{Man}$. This intention specialises the 'Conceptualise' intention type.

An important benefit of writing intentions that specialise only one of the five types of intentions is that this gives us a criteria to determine the size of chunks : chunks that denote activities pertaining to more that one of the five activity types should be split into lower level chunks. In the next section, we present how the chunks are organised according to these five types.

In the previous section, the overall structure of chunks was presented and indicated how to write and use them. At that level, the impact of the CREWS glossary is the following :

- for the chunk producer, the interface and the descriptor of the chunk has to be written by using the entries defined in the glossary ;
- for the chunk users, they can select the desired chunks on the basis of the basic entries or of their synonyms which enrich the questioning of the database.

5. Hierarchical indexing of the CREWS chunks

In order to facilitate the retrieval of chunks from the repository it is useful to index the collection of chunks by a hierarchy of intentions. This conforms to practice in Object Orientation [11]. Figure 6 shows the top level of the CREWS hierarchy whereas figures 7, 8, 9 and 10 relate each of the five key requirements engineering intentions to the available CREWS chunks.

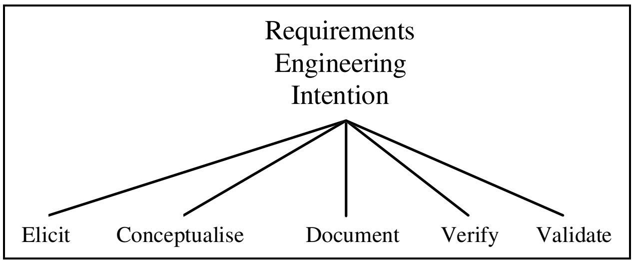

Figure 6: Top level of the CREWS hierarchy of RE intentions

For example Figure 7 shows that the *'Elicit'* intention may have four different targets, namely , scenario, use case, goal, and requirements. This leads to the four sub-intentions, '*Elicit scenario'*, '*Elicit use case'*, '*Elicit goal'* and '*Elicit requirements'*. For each of these *'Elicit'* sub-intentions, CREWS proposes a number of chunks corresponding to the different CREWS approaches to fulfilling the intention. Each of them is introduced in the hierarchy by an associated manner. For example there are five different manners to '*Elicit scenario'*. The corresponding processes are captured in chunks 1 (C1), 2 (P13), 3 (P1), 4 (C2), and 5 (A1). Chunks 2 and 3 support scenario authoring through style and contents guidelines displayed by L'ECRITOIRE software tool environment whereas chunk 1 provides simple word processing facilities. In chunk 4, scenarios are generated and filtered automatically from use case descriptions by the CREWS - SAVRE environment.

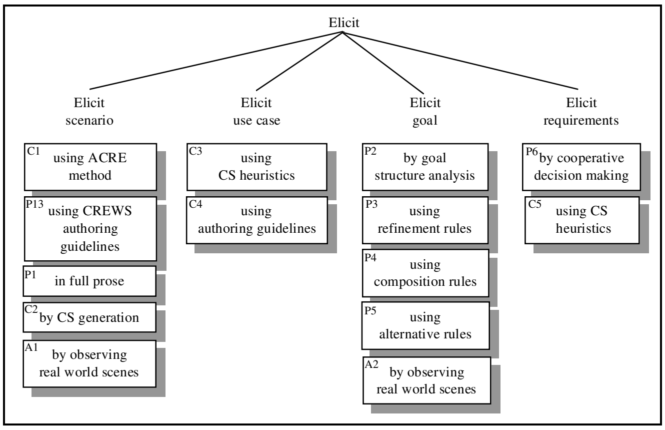

Figure 7: Hierarchy of the CREWS chunks for the *'Elicit'* intention

Similarly, the hierarchies in Figure 8, 9 and 10 show which chunks are made available in the CREWS base to contribute respectively to the intentions '*Conceptualise*', '*Document*', '*Verify*' and '*Validate*'.

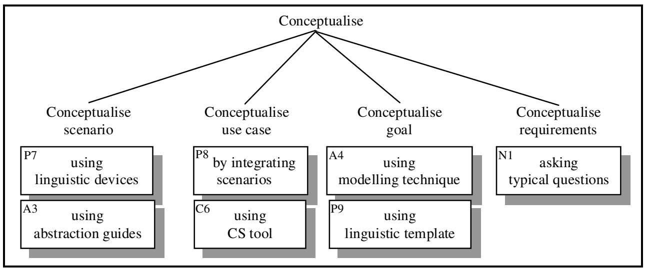

Figure 8: Hierarchy of the CREWS chunks for the *'Conceptualise'* intention

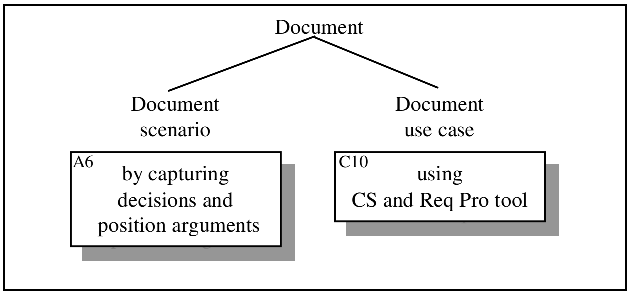

Figure 9: Hierarchy of the CREWS chunks for the *'Document'* intention

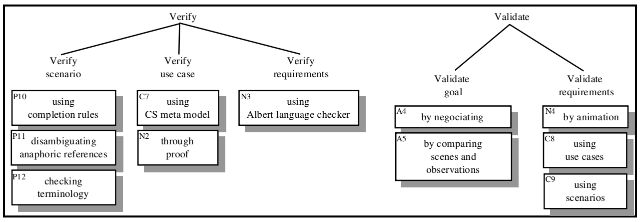

Figure 10: hierarchy of the CREWS chunks for the *'Verify'* and *'Validate'* intentions

The hierarchy of intentions provides a mean to browse over the contents of the method base. It therefore facilitates the retrieval of the chunks matching the requirements of the method base user, i.e. the method engineer.

It is complementary to the query facilities provided by the SgmlQL language which allows the user to retrieve chunks on the basis of the information provided by the descriptors. This will be illustrated in the next section.

6. Developing RE processes through the composition of chunks

Requirements Engineering processes can be developed by *selecting* and *composing* the different chunks made available in the chunk method base. This activity is supported by the information contained in the glossary and in the hierarchy presented in the previous section.

In this section, we will first illustrate the result of this activity by presenting a specific global RE process defined for some application development. Then, for a fragment of this global process, we will show how it has been constructed by using tools for retrieving and composing chunks.

6.1 An example of RE process

The proposed RE process is inspired from the experience that some of the authors got in the context of the Esprit 2RARE project, a project where was studied the use of novel requirements techniques in the context of two trial industrial applications. One of the two applications was related to the development of a Video-on-Demand system where the basic issues were concerning (i) the clarification of unstructured and poorly expressed requirements and (ii) the use of techniques supporting the detection of missing requirements. More details about the application context can be found in [31] together with some details on the semi-formal (like, ERA) and formal (the Albert language) techniques used.

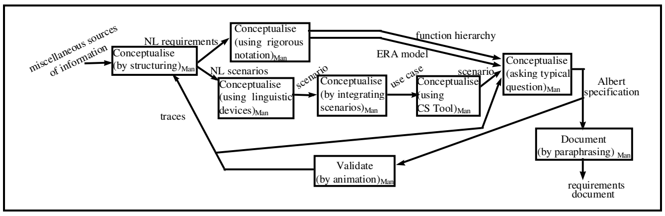

Figure 11: application of the 2RARE project

Figure 11 presents the whole enriched RE process that could be applied for the development of this application. This process is partly based on different available chunks defined by the CREWS participants (see P7, P8, C6, N1, N4 in figures 8 and 9).

- The inputs of the '*Document by structuring'* chunk are all the various unstructured sources of information collected from the different stakeholders. The objective is to somehow structure these pieces of information by distinguishing among informal texts related to requirements and those related to scenarios.
- From the informal texts related to requirements, the chunk '*Conceptualise using rigorous notation'* aims at producing semi-formal models, namely an ERA diagram related to the data structure and a functional hierarchy diagram related to the static part of the functions. This chunk is clearly a composed chunk since it is composed by finer chunks associated with the two semi-formal notations used.
- The '*Conceptualise using linguistic devices'* chunk, denoted as P7 in figure 8, has the objective to clarify and achieve a better quality for the informal scenarios proposed. The use of linguistic techniques together with the application of appropriate heuristics leads to the formalisation of scenarios and their integration into use-cases [21]. This chunk is a composed one made of finer chunks introduced in the previous section.
- The '*Conceptualise by integrating scenarios'* chunk, denoted as P8 in figure 8, has the objective to integrate several scenarios in order to conceptualise a use case.
- The '*Conceptualise using CS tool'* chunk, denoted as C6 in figure 8, has the use-cases produced by the previous chunk as an input. This elementary chunk (see previous section) has for objective the generation of a set of complete and correct scenarios.
- The '*Conceptualise asking typical questions'* chunk, denoted as N1 in figure 8, is a basic chunk taking different semi-formal models as input and

transforming them (using heuristics and additional stakeholders pieces of information) into Albert formal descriptions. Albert is a formal requirements specification language designed for the purpose of capturing requirements inherent to real-time distributed systems [5].

- The '*Validate by animation'* basic chunk, denoted as N4 in figure 9, starts from the Albert formal requirements specification and proposes to the stakeholders to interactively and co-operatively use a tool (the so-called animator) in order to explore different possible behaviours (or traces) of the future system allowed by the formal requirements [4].
- Finally, the '*Document by paraphrasing'* basic chunk aims at the paraphrasing of the formal Albert specification into its natural language counterpart which constitutes the requirements document. This document should necessarily be written in natural language because of the variety of stakeholders who have to read it.

6.2 Elaborating an RE process

Figure 11 shows the result of the method engineering activity. In this subsection, we give some flavours about how this final result has been obtained. Considering a subpart of the process, we exemplify the retrieval of scenario chunks from the chunk method base and their assembly.

One of the basic constraint in the trial application was related to the use of a formal requirements specification language, namely the Albert language. On the other hand, from the beginning, this was clear that the inputs were badly structured texts associated with the descriptions of requirements scenarios. To summarise, the basic situation is depicted as in the Figure 12 below.

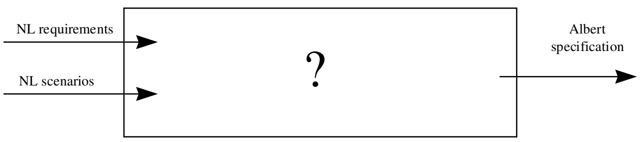

Figure 12: The basic situation of the RE process

Being driven by this need for formal requirements, one can apply a SgmlQL query on the chunk method base in order to retrieve the possible chunks leading to an Albert specification :

Select text($i) . « \n »
from $c in every CHUNK within $Crewsfile,
$i in every CHUNK_INTENTION within $c,
$t in every TARGET within $i
where text($t) match « Albert specification » ;

The unique result of the query (Conceptualise requirements asking typical questions) is incorporated in the process under construction (see Figure 13 below).

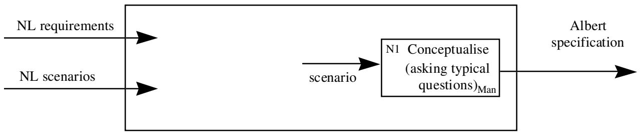

Figure 13 : The RE process resulting of the first query

Now, focusing our attention on the scenario inputs (i.e. on the product parts of the situation), we can query the chunk base in order to identify potential chunks satisfying the condition.

Select text($i). « \n »
from $d in every DESCRIPTION within $Crewsfile,
$descro in every DESCRIPTOR within $d,
$dom in every DOMAIN within $descro,
$da in every DESIGN_ACTIVITY within $descro,
$cm in every COMPLEX_MANNER within $descro,
$t in every TARGET within $cm
$c in every CHUNK within $d,
$i in every CHUNK_INTENTION within $c,
$v in every VERB within $i
where ((text($dom) match « interactive system »)
and (text($da) match « requirements acquisition »)
and (text($t) match « scenario »)
and ((text($v) match « conceptualise »)
or (text($v) match « elicit »))
and (text($t->Role) eq «object»)
and (text($t->Notation) eq « Semi-formal »)
and (text($t-> Coverage) eq « Functional »)
and (text($t->Context) eq « System interaction »)) ;

Obviously, there are too many chunks satisfying the condition *(Elicit scenario using ACRE method (C1), Elicit scenario using CREWS authoring guidelines (P13), Elicit scenario by CS generation (C2), Elicit scenario by observing real world scenes (A1), Conceptualise scenario using linguistic devices (P7), Conceptualise scenario using abstraction guides (A3)),* and this is difficult to decide for one chunk rather than for another. As an alternative strategy, one may prefer to start from the NL scenarios descriptions considered as an input of a chunk.

Select text($i) . « \n »
from $d in every DESCRIPTION within $Crewsfile,
$descro in every DESCRIPTOR within $d,
$dom in every DOMAIN within $descro,
$da in every DESIGN_ACTIVITY within $descro,
$cm in every COMPLEX_MANNER within $descro,
$t in every TARGET within $cm,
$c in every CHUNK within $d,
$i in every CHUNK_INTENTION within $c,

```
        $v in every VERB within $i,
        $pp in every PRODUCT_PART within $c
where ((text($dom) match « interactive system »)
            and (text($da) match « requirements acquisition »)
            and (text($t) match « scenario »)
            and ((text($v) match « conceptualise »)
                        or (text($v) match « elicit »))
            and (text($t->Notation) eq « Informal »)
            and (text($t-> Medium) eq « Text »)
            and (text($pp) eq « NL scenarios »)) ;
```

The query results in the '*Conceptualise scenario using linguistic devices* ' chunk (P7) and the composed process is now as shown below (

Figure 14).

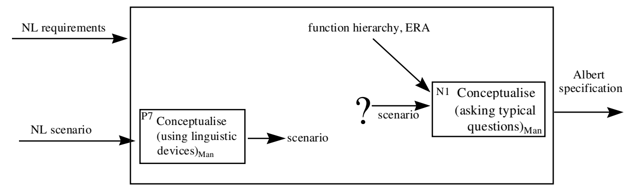

Figure 14 : The RE process resulting of the third query

As the output of the chunk P7 is a scenario, one possibility could be to simply combine P7 and N1. However a look to the relationships that P7 has with other chunks indicates four possibilities P3, P4, P5 and P8 (see the corresponding query below). The three first help moving from scenario to goals and are not of interest whereas P8 supports the integration of scenarios in a single use case, in a semi-automated way. P8 is selected.

```
Select text($suc). « \n »
from $d in every DESCRIPTION within $Crewsfile,
        $descro in every DESCRIPTOR within $d,
        $suc in every SUCC within $d,
        $c in every CHUNK within $d,
       $i in every CHUNK_INTENTION within $c
where text($i) match «conceptualise scenario using linguistic devices » ;
```

The status of the RE process under construction is now as depicted in figure 15.

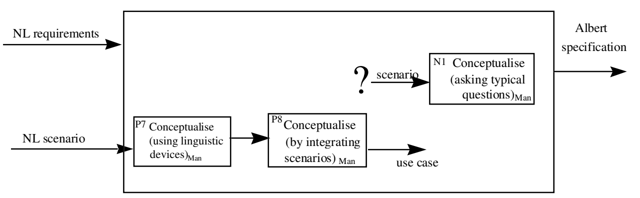

Figure 15 : RE process resulting of the fourth query

The next query will try to identify a chunk having a use case as input and scenario as output.

```
Select text($i) . « \n »
from $c in every CHUNK within $Crewsfile,
        $i in every CHUNK_INTENTION within $c,
        $t in every TARGET within $i,
        $v in every VERB within $i,
        $pp in every PRODUCT_PART within $c
where ((text($t) match « scenario »)
            and (text($v) match « conceptualise »)
            and (text($pp) eq « use case »)) ;
```

The result of this query is the *'Conceptualise use case using CS tool'* chunk (C6) which can be integrated with the others as depicted in the previous sub-section (see Figure 11).

To overcome the problem of readability of the SgmlQL queries, for the method engineer which are not familiar with the SgmlQL formalism, we plant to build a tool prototype able to automatically perform the authoring of the most useful queries for retrieving chunks during the requirements engineering processes development.

7. Discussion

This paper reports the results of preliminary research within CREWS into a scenario-based requirements engineering method. The development of this method has led to advances in two directions. First, the four CREWS approaches to scenario-based requirements acquisition and validation have been integrated to provide a partial but coherent and novel method based of requirements elicitation from observations from real-world practice, computational analysis of natural language descriptions of scenarios and goals to acquire system requirements, systematic generation and walkthroughs of scenarios to validate system requirements, and animation of formal requirement specifications to ensure their correctness and completeness. Furthermore, the current incomplete coverage of these four approaches is overcome through their integration with other existing approaches within the CREWS method. The second, more important development is the definition, design and implementation of a data base of reusable process components which prescribe how to undertake the CREWS approaches. This development has implications for defining and delivering new processes and techniques which complement rather than replace existing software engineering methods. The remainder of this section will discuss this second development.

Our modular approach to process modelling draws extensively on ideas from software reuse (e.g. [19]). It is a "white-box" approach because the "reuser", that is the method engineer, uses the external description of a process chunk as well as descriptions of its internal contents to retrieve, understand and apply the process. We believe that such a white-box approach is effective for method engineering for several reasons. First, the contents of a chunk will have to be adapted to fit each new process. Acquiring, modelling and validating system requirements are complex processes which can be affected by factors as diverse as the previous systems development process, the experience of the requirements engineering team and the amount of time available to undertake each process (e.g. [13]). Although the influence of context on the process is reflected in the situation description of a chunk, it is impossible to predict and hence describe all influencing factors. Therefore, processes will often need minor modification to fit each new situation.

Second, the composition of a coherent process can be improved by making all three descriptors of each chunk (the situation, intention and description) available to the method engineer. Our assumption that minor modifications to processes are inevitable means that perfect "cohesion" between any two chunks is unlikely, that is there will be few perfect "fits" between chunks. However, the likelihood of a sufficient fit can be increased by a larger number of chunk facets to provide links between chunks. For example, two chunks which have a good fit using the intention and situation facets but not the description facets can still be composed together in a process, but with the caveat that changes in the scenario product (the focus of the description) might be needed.

So far, the application of our method engineering approach has been limited to the four CREWS approaches and two of the most common commercial use case-driven approaches. Further evaluation is still needed to demonstrate its potential effectiveness. However, the experiences of the authors within CREWS indicate that the approach is viable. One of the most critical problems to overcome is composition of process chunks into a method. The example reported in section 6 demonstrates that the CREWS chunks can be linked together in a meaningful way. Indeed, within CREWS, the acts of defining our approaches using a common form of description and comparing these descriptions enabled the authors to identify previously unforeseen overlaps between the approaches which has led to greater method and software tool integration between them.

Further work is needed to evaluate the structure and glossaries for defining each process chunk. To achieve this, we will model reported "good practice" with scenarios for acquiring, modelling and validating systems requirements (e.g. [17]). We anticipate that this will led to extension and refinement of the CREWS process chunks as well as the way to describe them. One specific tactic for evaluating the current contents of the CREWS method base is to make them available for public access and critiquing using the internet. The SGML data base is configurable for Internet access. Not only will this enable the authors to elicit constructive criticism of their approaches from a large population, but it will also provide the starting point for a living repository of expertise and experience reports about scenario reuse that the academic and practitioner communities can reference and contribute to over time. We look forward to reporting on the base and feedback on its contents in the near future.

The description of process chunks in SGML provides a starting point to enact processes (e.g. [3], [6]) through formalisation of the description of each process in a chunk. We believe that the modularisation of processes through chunks makes the formalisation and hence enactment of processes more feasible due to the reduction size and removal of contextual factors from each chunk process. Future research will be in two parts. The first will be to model each chunk of process using a process modelling formalism ([22], [28]) to enable its enactment. The second will be to define composition formalism to enable enactment of more than one chunk in a single process. The way we envision this is based on a meta-process to guide the construction of the integrated process expressed with the same process modelling formalism as the one used for process chunks. Therefore the global process shall be enacted « on the fly » using single enactment mechanism. Our long-term goal is complete process enactment through "real-time" composition of a tailored requirements engineering method from reusable process chunks. The achievement of this goal, the authors believe, will be a significant step towards adaptive and scaleable approaches to guiding the systems development process.

Acknowledgement : This paper was prepared as part of work package W2 of the CREWS project. The CREWS team includes in addition to the authors of this paper : C. Rolland as leader of the work package, C. Ben Achour, M. Jarke, K.Pohl, P. Haumer, and S. Minocha.

References :

[1] A. Cockburn, *Structuring use cases with goals*. Technical report. Human and Technology, 7691 Dell Rd, Salt Lake City, UT 84121, HaT.TR.95.1, http://members.aol.com/acocburn/papers/usecases.htm (1995).

[2] : V. De-Antonellis., B. Pernici, P. Samarati, *F-ORM METHOD : A Methodology for Reusing Specification,* In Object

Oriented Approach in Information Systems, Van Assche F., Moulin b., Rolland C. (eds), North Holland, 1991

[3] M. Dowson, *Software Process Themes and Issues*, Proc. of the 2nd Int. IEEE Conf. on Software Process, Berlin, Germany , pp 28-40, 1993.

[4] : E. Dubois and P. Heymans, *Scenario-Based Techniques for Supporting the Elaboration and the Validation of Formal Requirements.* CREWS Report, University of Namur, 1998.

[5]. : P. Du Bois, E. Dubois and J.M. Zeippen, *On the use of a formal RE language: the generalized railroad crossing problem.* Proc. of the IEEE International Symposium on Requirements Engineering (RE'97), Annapolis MD, January 1997, pp. 128-137, IEEE Computer Society Press

[6] A. Finkelstein, J. Kramer, B. Nuseibeh (eds), *Software Process Modelling and Technology*, John Wiley, 1994

[7] : C. F. Goldfarb, *The SGML Handbook ,* Oxford Clarendon Press, 1990.

[8] : P. A. Gough, F. T. Fodemski, S. A. Higgins, S. J. Ray, *Scenario - an industrial Case Study and Hypermedia Enhancements»*, Second IEEE International Symposium On Requirements Engineering, 1995.

[9] : P. Heymans, *Some toughts about the animation of formal specifications written in the ALBERT language* , Proceedings of the Doctoral Consortium of the 3rd IEEE International Symposium on Requirements Engineering (RE'97), Annapolis, MD, USA, January 6-10, 1997.

[10] I. Jacobson, M. Christerson, P. Jonsson and G. Oevergaard, *Object Oriented Software Engineering: a Use Case Driven Approach ,* (Addison-Wesley, 1992).

[11] : Kang K., Cohen S., Hess J., Novak W., Peterson S., *Feature-oriented domain analysis (FODA) feasibility study* CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1990

[12] : J. Lemaitre, E. Murisasco, M. Rolbert, *SgmlQL, « Un langage d'interrogation de documents SGML »,* Proceedings of the 11th conference on Advanced DataBases, August 1995, Nancy, France.

[13] : N.AN. M. Maiden, G. Rugg, *ACRE : A Framework for Requirements Acquisition Methods*. Software Engineering Journal 11(1) , Jan 96

[14] : N.A.M. Maiden, S. Minocha, K. Manning, M. Ryan, *CREWS-SAVRE : systematic Scenario Generation and Use*. Third International Conference on Requirements Engineering (ICRE'98), Colorado-Springs, USA, 1998.

[15] V. Plihon, C. Rolland, *Modelling Ways of Working*. Proc. Of the 7th International Conference on Advanced Information Systems Engineering, CAiSE'95, Springer Verlag, 1995.

[16] V. Plihon, *Un environnement pour l'ingénierie des méthodes*, PhD thesis, Jan 1996.

[17] C. Potts, K. Takahashi, A.I. Anton, *Inquiry-based requirements analysis*. In IEEE Software 11(2), pp. 21-32, 1994.

[18] N. Prat, *Goal formalisation and classification for requirements engineering*. Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'97, Barcelona, pp. 145-156, June 1997.

[19] : R. Prieto-Diaz, P. Freeman, *« Classifying Software for Reusability»,* IEEE Software, Vol 4 No 1, 1987.

[20] Prieto-Diaz R., "Implementing Faceted Classification for Software Reuse", Communications of the ACM, Vol.34, No.5, May 1991.

[21] C. Rolland, C. Ben Achour, *Guiding the construction of textual use case specifications*. Data and Knowledge Engineering Journal, 1997.

[22] C. Rolland, G. Grosz, *A General Framework for Describing the Requirements Engineering Process* , IEEE Conference on Systems, Man and Cybernetics, CSMC94 , San Antonio, Texas, 1994.

[23] C. Rolland, V. Plihon, *Using generic chunks to generate process model fragments*. Proceedings of the 2nd International Conference on Requirements Engineering, ICRE'96, Colorado Springs, 1996.

[24] C. Rolland, C. Souveyet, M. Moreno, *An Approach for defining Ways of Working*. Information Systems, Vol 20, No 4, pp337-359, 1995.

[25] : C. Rolland, V. Plihon, J. Ralyte, *Specifying the reuse context of Scenario Method Chunks* , Paper accepted to the Conference on Advanced System Engineering (CAISE'98), 1998.

[26] : C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N.A.M. Maiden, M. Jarke, P. Haumer, K. Pohl, Dubois, P. Heymans, *A proposal for a scenario classification framework*. Requirements Engineering Journal 3 :1, 1998.

[27] : J. Rumbaugh, G. Booch, *«Unified Method »,* Notation Summary Version 0.8 (Rational Software Corporation, 1996).

[28] S. Si-Said, C. Rolland, G. Grosz, *MENTOR :A Computer Aided Requirements Engineering Environment*, in Proceedings of CAiSE' 96, Crete, GREECE, May 1996.

[29] : B. Thomé, *Systems Engineering : Principles and Practice of Computer-based Systems Engineering*, in B. Thomé (ed), John Wiley & Sons (1993).

[30] M. Jarke, K.Pohl, P. Haumer, K. Weidenhaupt, E. Dubois, P. Heymans, C. Rolland, C. Bena Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N.A.M. Maiden, S. Minocha, *Scenario Use in European Software Organizations --- Results from Site Visits and Questionnaires*. CREWS report series 97-10.

[31] R. Wieringa and E. Dubois, *Integrating semi-formal and formal requirements*, Information Systems Journal, (to appear), 1998.

Appendix: The CREWS Glossary

At the product level

Animation

Definition : an animation is the creation of a finite set of finite sequences describing normative or non normative behaviours of the composite system. While scenarios focus on the interactions taking place between the system and its environment, the result of the animation considers the possible behaviours of the whole composite system and helps in exploring them.

Behaviour

Definition : a possible behaviour consists in an alternate sequence (possibly infinite) of states and state transitions, where :

The state is structured in terms of components and the values of components stay unchanged between two state transitions,

The state transitions correspond to the beginning and/or the ending of actions, called events.

Some behaviours can be considered as more normative than some other. Thereby, a behaviour can be classified as normative or non-normative according to the fact it is considered to include a few or a lot of exceptions.

Episode

Definition : According to Regnell and Potts, on episode is a « part of a use case representing a demarcated and coherent flow of events ». They help structure a use case in manageable units.

Goal

Definition : Goal is a future system state or behaviour to avoid, maintain, attain, cease, etc.

Synonym: Intention.

Message trace diagram

Definition : a Message Trace Diagram (MTD) is a graphical way of representing the communication part associated with scenarios. It exists some extensions which allow to express also internal actions which widens their scope from just expressing communication.

Synonyms : sequence diagram.

Open Issue

Definition : result of the RE process.

Problem statement

Definition : something that you say or write about a situation that causes difficulties.

Product

Definition : a product is the result which remains after the execution of a process.

Requirement

Definition : Requirement is a change or quality criterion for some future system (version). We distinguish functional and non-functional requirements.

Scenario

Definition : At a functional level, a scenario is a description denoting similar parts of possible behaviours limited to a subset of purposeful state components, actions and communications taking place among two or several agents.

More external (richer) scenarios include information about roles, responsibilities, organisation policies, ...

Synonyms : contextual scenario.

Scene

Definition : all the things that are happening in a place, and the effect or situation that they cause.

Specification

Definition : set of behaviours of the system and of its environment.

Use case

Definition : a use case is defined as a possibly structured set of scenarios grouped together to achieve a specific stakeholder goal.

Use case model

Definition : see UML definition [27].

At the process level

Analyse :

Definition : a cognitive activity involving the decomposition , the structuring and scoping of a knowledge as well as deducing properties of the thing, under analysis, e.g. incompleteness, incorrectness, etc. These RE-specific properties should be quite easy to list.

Synonyms : to understand, to reason about.

Animate

Definition : the interactive process of visualising the dynamic properties associated with fragments of normative or non-normative behaviours of the composite system.

Synonyms : to activate, to simulate.

Change

Definition : to make something or someone different.

Synonyms : to modify.

Compare

Definition : to consider two or more process, product, requirements, etc. in order to show how they are similar to or different from each other.

Compose

Definition : (to be composed of) to be formed of a group of parts.

Synonyms : to assemble, to aggregate, to integrate, to combine.

Conceptualise

Definition : the process of systematically abstracting (existing or envisaged) real-world phenomena into models which highlight the essential aspects and hide the unimportant details (relative to the viewpoint taken).

Synonyms : to model, to abstract.

Create

Definition : to make something exist that did not exist before. The process of (semi-) automatically building a product (scenario, requirements specification, etc.) in some targeted formalism, starting from a semantic definition of its content.

Synonyms : to compose, to design, to generate.

Decompose

Definition : the process of partitioning a product/ process/ problem into more manageable units.

Synonyms : to atomise, to partition.

Document

Definition : as opposed to 'to conceptualise', write down the product of activities such as analyse, compare, change, etc.

Synonyms : to describe, to specify, to record, to write.

Elicit

Definition : the process of systematically obtaining from people new facts (scenarios, requirements) about the domain / business processes / the system under consideration.

Synonyms : to acquire, to discover, to capture.

Envision

Definition : to project what a product will be.

Synonyms : to project, to imagine.

Explain

Definition : to make something clear or easy to understand, to give a reason for something to someone.

Synonyms : to clarify, to illustrate.

Explore

Definition : the process of envisaging (evaluating) alternatives, or scope or pathways.

Synonyms : to navigate.

Find

Definition : to achieve or get something that you need.

Synonyms : to achieve, to retrieve.

Gather facts

Definition : from documents.

Synonyms : to collect facts.

Identify

Definition : to recognise and correctly name an element of a product or of a process; to perceived some coherent entity of the (existing or envisaged) real-world (or Universe of discourse) and, optionally, express it as an element of a product, process, ...

Synonyms : to name.

Negotiate

Definition : the process of integrating different viewpoints of different stakeholders on a certain topic, in order to try to reach an agreement of all involved stakeholders. Involves mediation and reconciliation.

Synonyms : to mediate, to reconcile.

Refine

Definition : the process (more detailed or more precise) of changing a product (or process) in a systematic way so that the changed product/process is better (more detailed or more precise) according to some characteristic than the former one.

Synonyms : to elaborate, to improve.

Relate

Definition : the process of explicitly defining links between products between which a semantic or structural relationship exists.

Synonyms : to associate, to link, to structure, to map.

Remove

Definition : to take something away from the place where it is.

Synonyms : to delete.

Review

Definition : to examine, consider and judge a product or a process carefully with respect to completeness and correctness.

Synonyms : to assess, to evaluate.

Scope

Definition : to draw the boundaries of the system.

Synonyms : to delimit (« the part of the Universe of Discourse some product, process, requirement, ...refers to »).

Search

Definition : to try to find a solution to a problem, an explanation for something, etc.

Synonyms : to explore, to investigate.

Select

Definition : to choose among candidates products or processes subset by carefully thinking about which is the best, most suitable, etc. for satisfying a higher-level intention

Synonyms : to choose.

Sort

Definition : to put things in a particular order or to arrange them in groups according to size, rank, type, etc.

Synonyms : to prioritise, to rank, to order, to categorise, to classify.

Suggest

Definition : to provide someone with useful information with respect to the intention (s)he as to satisfy.

Synonyms : to advise, to recommend.

Trace

Definition : to record and subsequently retrieve information about the product and process evolution in a sequential or time-ordered manner.

Synonyms : to record.

Validate

Definition : the process of checking against stakeholders that the right product is being built.

Synonyms : to test, quality assure.

Verify

Definition : to get the product right.

Synonyms : to attest, to check.

Walk through

Definition : to validate a model in a co-operative setting using a sequential process that tests components in an order. The validation is made manually, not in an automatic manner, it follows a stepwise process for checking.

Synonyms : to check, to validate.