CREWS Report Series 98 - 5

SPECIFYING THE REUSE CONTEXT OF SCENARIO METHOD CHUNKS

C. Rolland[*], V. Plihon[+*], Y. Ralyté[*]

[*]Université Paris1-Sorbonne
CRI
17, rue de la Sorbonne
75231 Paris Cedex 05, France

{rolland, ralyté}@univ-paris1.fr

[+]Université de Toulon et du Var
GECT
BP 132
83957 La Garde Cedex, France

plihon@univ-tln.fr

Appeared in the proceedings of the 10th International Conference CAiSE'98, B. Lecture Notes in Computer Science 1413, Pernici, C. Thanos (Eds), Springer. Pisa, Italy, June 1998

SPECIFYING THE REUSE CONTEXT OF SCENARIO METHOD CHUNKS[1]

C. Rolland[*]**, V. Plihon**[+*]**, Y. Ralyté**[*]

[*]**Université Paris1-Sorbonne**
CRI
17, rue de la Sorbonne
75231 Paris Cedex 05, France
Tel : + 33 (0)1 44 24 93 65
Fax : + 33 (0)1 45 86 76 66
{rolland, ralyté}@univ-paris1.fr

[+]**Université de Toulon et du Var**
GECT
BP 132
83957 La Garde Cedex, France
Tel : + 33 (0)4 94 14 25 41
Fax : + 33 (0)4 94 14 21 65
plihon@univ-tln.fr

[1] This work is partly funded by the Basic Research Action CREWS (ESPRIT N°21.903). CREWS stands for Cooperative Requirements Engineering With Scenarios.

SPECIFYING THE REUSE CONTEXT OF SCENARIO METHOD CHUNKS

Abstract : *There has been considerable recent interest in scenarios for accompanying many of the various activities occurring in the development life cycle of computer based systems. Besides the integration of scenarios in methods such as Objectory and software tools such as Rationale Rose has proven useful and successful. Consequently, there is a demand for adapting existing methods to support specific design activities using scenario based approaches. The view developed in this paper is that scenario based approaches should be looked upon as reusable components. Our concern is therefore twofold : first, to represent scenario based approaches in a modular way which eases their reusability and second, to specify the design context in which these approaches can be reused in order to facilitate their integration in existing methods. The paper concentrates on these two aspects, presents an implementation of our proposal using SGML to store available scenario based approaches in a multimedia hypertext document and illustrates the retrieval of components meeting the requirements of the user by the means of SgmlQL queries.*

1. Introduction

Scenario based approaches have proven useful in a large number of situations occurring in the system development life cycle. In the HCI community, scenarios have been proposed as detailed descriptions of a usage context so design decisions can be reasoned about [Caroll95] or as small examples of an existing product which are used to anchor discussion about different design theories [Young87]. In Software Engineering, use case approaches have been developed to derive the object oriented specification of a system from narrative descriptions of interactions with its users. In the Information Systems community scenarios have evolved to the concept of a rich picture which gives the social setting of a required system so arguments can be developed about the impact of introducing technology, and the matching between user requirements and task support provided by the system [Kyng95]. Finally in Requirements Engineering, scenario scripts based approaches have been proposed to support the checking of dependencies between a requirements specification and the user/system environment in which it will have to function [Potts94].

These examples demonstrate that a scenario based approach aims primarily at supporting some specific design activity. By essence these approaches are not standalone products but instead, they have vocation to be integrated in existing methods to support some specific steps of the design process with the advantage of increasing usability. As a specific scenario based approach provides support to a specific design activity, it might be possible to integrate it in various different methods dealing each with this particular design activity. Our view is that *scenario based approaches should be looked upon as reusable components*. This reuse perspective has been already illustrated, for example by the use case approach originally developed by Jacobson [Jacobson95a], [Jacobson95b], and then, integrated in a number of existing methods including the Fusion method [Coleman94], OMT [Rumbaugh91], [Rumbaugh94] and UML [Booch97]. However reuse has been performed in an 'ad hoc' manner while there is a demand [Jarke97] for a more systematic way of understanding when, why and how, which kind of scenario has to be used. Thus, if we want to support the reuse of the large corpus of available scenario based approaches we shall solve the problem of *characterizing the context* in which they can be reused.

In this paper we are concerned by these two issues : (a) to represent scenario based approaches as reusable components and (b) to specify the context of use of available scenario based approaches in order to facilitate their reuse in different methods to support the design activities they are dedicated to.

Our proposal[2] is based on an analogy with object oriented reuse and comprises two aspects :

(1) to define a scenario based approach as a collection of methods fragments that we call *scenario method chunks* (*scenario chunks for short*) and to make them available in a scenario method base.
(2) to characterize the context of use of scenario chunks in *chunks descriptors* and to store them in the scenario base together with the chunks themselves. This shall ease the retrieval of scenario chunks meeting the requirements of the method base user.

Our view is therefore to organize the scenario method base at two levels, the *method knowledge level* and the *method meta- knowledge level* and to tightly couple scenario chunks with the meta-knowledge describing their context of use. The method level tells us how to apply a specific scenario chunk whereas the meta-level provides knowledge about the conditions under which the scenario chunk is applicable. Our notion of chunk descriptor is close to the view of [De Antonnellis91] and very similar to the one of faceted classification schema [Pietro-Diaz87] developed in the context of software reuse.

We have implemented the proposed approach using SGML (Standard Generalized Markup 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. This provides the required facilities to query the scenario method base and retrieve the scenario chunks which match specific reuse conditions.

In the rest of the paper, we develop in detail the scenario method knowledge and the scenario meta-method knowledge as well as their implementation. Section 2 deals with the former, presents the notion of scenario chunk, illustrates the different levels of granularity of chunks and exemplifies them by describing several existing scenario based approaches. Section 3 deals with the meta-knowledge representation, defines and exemplifies the notion of chunk descriptor. Section 4 covers the implementation of the scenario method base in SGML. In section 5 we illustrate through examples of queries in sgmlQL how scenario chunks can be retrieved from the method base. Finally we draw some conclusions in section 6.

2. Scenario method knowledge level

We adopted a modular approach to represent the scenario method knowledge, in the method base, in the form of *scenario method chunks, scenario chunks* for short. A scenario chunk may represent an entire approach such as the Jacobson's use case approach or part of it, for example the chunk to define abstract use cases. This eases the reusability of chunks and their integration in methods. A chunk tightly couples a product part and a process part. In the product part, the product to be delivered by a scenario chunk is captured whereas in the process part, the guidelines allowing to produce the product are given. As we are interested in *scenario* chunks, at least one of the product parts involved in a chunk must be of the scenario type. The guidelines to define the use case model proposed in the OOSE methodology [Jacobson92], to capture and use scenario scripts [Potts94], to construct interaction diagrams [Rumbaugh96], or abstract usage views [Regnell95] are examples of such scenario chunks.

2.1 The notion of scenario chunk

Our definition of a scenario chunk is based on the process view of the NATURE process modelling formalism [Rolland94], [Plihon95] and consistent with the notion of 'step' in [Thomé93] . According to this view a process can be seen (figure 1) as a black box which transforms an initial *situation* into a result which is the *target* of the *intention* of the process. 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

[2] This work is partly founded by the ESPRIT project CREWS (N°21903).

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 couple *<situation, intention>* which is called *context* (see the example in figure 1).

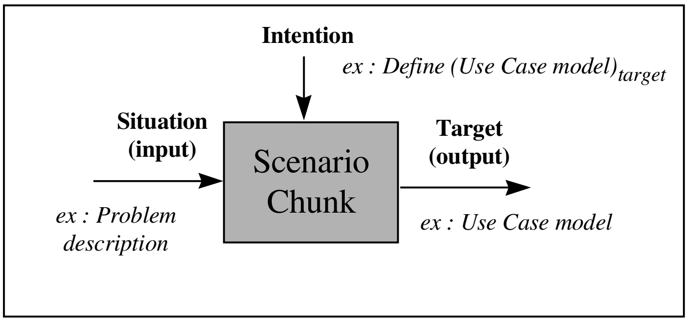

Figure 1 : The behavioral view of a scenario chunk

Following this view, a scenario chunk has two parts (figure 2)[3] : its *interface* which is the couple <situation, 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 fulfill the intention in that particular situation. The body provides *guidelines* to guide the process and relates the process to the *product parts* involved. For example, the interface of the scenario chunk representing the Jacobson's use case approach is the context <(Problem Statement), Define Use Case Model > where the situation is represented by a document called *problem statement* and the intention, *to define use case model*, is based on this problem statement. The target of the intention, *use case model* defines the result to be produced by the application of the chunk. The body provides the guidelines to achieve the intention of defining a use case model out of problem statements (see figure 4 that we will explain later on). Additionally, as shown in figure 2, each scenario chunk in the method base has a *unique name*. It may be *represented graphically* and /or *described informally* in natural language. It is also possible to provide some *examples* of anterior application of this scenario chunk. A scenario chunk is classified either into *for*mal or *informal.* In the former case the guidelines are formally defined whereas they are informal textual recommendations in the latter.

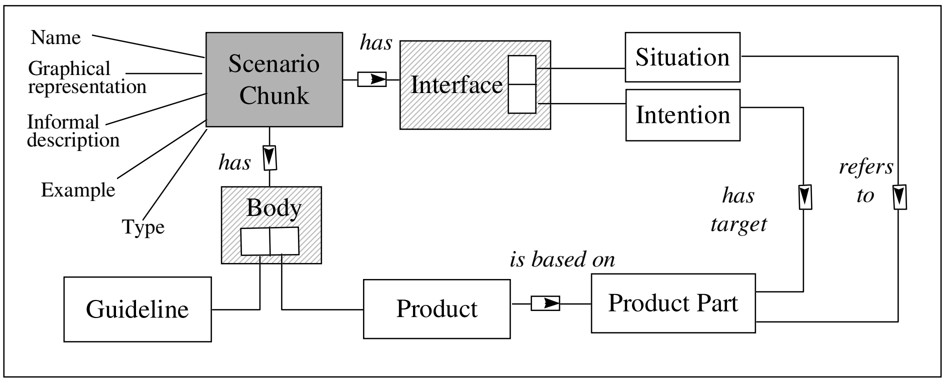

Figure 2 : The scenario chunk structure

As illustrated in the example above, the intention contains the reference to the target. In fact the structure of an intention (figure 3) is more complex than a simple verb. The proposed formalization of the notion of intention permits a fine grain characterization of the chunk which was felt necessary to support efficient retrieval of scenario chunks.

[3] The structure description is based on E/R like notation.

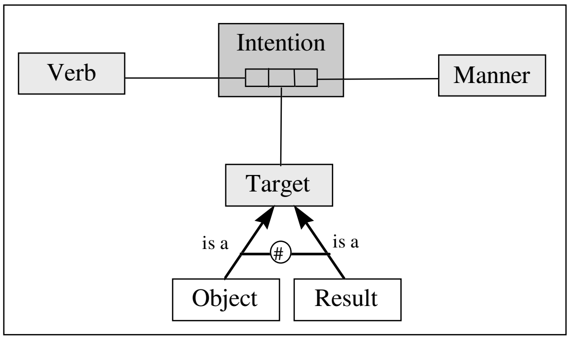

Figure 3: The intention structure

The *intention* is decomposed [Prat97] into a *verb*, a *target* (a product part) the verb is acting on and a *manner*. Depending on the role played by the product part for the verb, we make the distinction between *objects* and *results*. An *Object* exists in the situation of the corresponding context whereas a *Result* is produced by the achievement of the intention. "*Refine use case*", is an example of intention in which the target "*use case*" is an object because it already exists in the situation whereas "*Identify actor*" is an example where the target "*actor*" is a result. It is developed during the execution of this intention. The precise notation of these intentions is as follows : *<(Use Case), Refine (Use Case)$_{Obj}$>, <(Problem Statement), Identify (Actor)$_{Res}$***>.**

In addition to the verb and the target, the intention may have a *manner* which specifies in which way the intention is satisfied. A *manner* is a strategy or an approach used to fulfill the intention. *« One-shot refinement »* or *« stepwise strategy »* are examples of manners.

The proposed definition of a scenario chunk is applicable to any method chunk. The distinction between a scenario chunk from any other method chunk is due to the nature of the product parts. In the latter the product can be of any type whereas in the former either the situation or the target of the intention must refer to a product part of the scenario type. For example, the two chunks with the following interfaces *<(Problem Statement), Define (Use Case Model)$_{Res}$*> and *<(Use Case Model), Interpret <(Use Case Model)$_{Obj}$*> are scenario chunks. Both manipulate scenarios which are called use cases, the former having the use case model as the target of the intention to *Define Use Case Model* whereas the use case model is the object of the *Interpret* intention.

The application of the NATURE contextual approach to the representation of method knowledge has the important effect of *making chunks modular*. A chunk prescribes the way to proceed in a situation to fulfill an intention. A scenario chunk can be qualified as cohesive because it tells us the situation in which it is relevant and the intention that can be fulfilled in this situation. A chunk is loosely coupled to other chunks because it can be used in the appropriate situation (created as the result of another module) to satisfy the intention. Thus, the linear arrangement of method modules is replaced by a more dynamic one. Finally, the hooks to combine a scenario chunk with another chunk (whichever is its type) are parts of its interface : the situation and the target. Two chunks can be assembled if the situation of one of them is compatible with the target of the other.

2.2 The body of a scenario chunk

The interface of a scenario chunk characterizes the conditions of its applicability whereas its body details how to apply it. The interface plays a key role for retrieving a scenario chunk out of the method base while the body is used when applying the method in which the chunk has be integrated. Our approach relies upon the interface structure presented above but does not imply a particular way of describing the chunk body. In the sequel, we illustrate partially the solution we chose for the implemented method base.

We follow the NATURE approach and define the chunk body as a hierarchy of contexts called a *tree*. As illustrated in figure 4 contexts relate one to the other through three types of links : *refinement links* which permit the refinement of a large-grained context into finer ones, *composition links* which allow to decompose a context into component contexts and *action links* which relate the contexts to the actions which directly perform transformations on the product. Each type of link has a corresponding type of context, namely *executable, choice* and *plan* contexts. A detailed description of contexts can be found in [Rolland96]. Let us briefly illustrate them through the example of tree presented in figure 4.

A *choice context* offers choices supported by arguments. The context *<({Use Case}), Refine Use Case>* in figure 4 introduces three alternatives to a Use Case refinement : (1) to generalise a set of use cases into an abstract use case *<({Use Case}), Generalise {Use Case} into Abstract Use Case>*, (2) to specialise an abstract use case into concrete use cases *<(Abstract Use Case), Specialise Abstract Use Case into Concrete Use Case>* or (3) to extend a use case with a use case extension *<(Use Case), Complete Use Case with Use Case Extension>*.

A *plan context* corresponds to an intention which requires further decomposition. The plan context *<(Problem Statement), Define Use Case Model>* in figure 4 is composed of three contexts, namely *<(Problem Statement), Identify Actor>*, *<(Problem Statement, Actor), Define Use Case>* and *<({Use Case}), Refine Use Case>*. This means that, while constructing a use case model, the requirements engineer has first to identify the actors, then to construct use cases, and finally, to refine the use cases. The component contexts of a plan context can be organized within graphs in a sequential, iterative or parallel manner (see the top bubble in figure 4).

An *executable context* corresponds to an intention which is directly applicable through actions which transform the product under development. Performing the action changes the product and may thus generate new situations. In figure 4 the context *<(Problem Statement), Identify a Primary Actor>* is an executable context. The intention to "*Identify a Primary Actor* " is immediately applicable through the performance of an action which creates a new "primary actor" in the use case model under development. Notice that when the chunk is informal, its body is reduced to a textual explanation on how to perform the action (see figure 5).

Contexts can therefore be related through refinement and composition links to form a tree. A tree is considered complete when all its leaves are executable contexts. The scenario chunk in figure 4 is a tree in which the root context *<(Problem Statement), Define Use Case Model>* is decomposed and refined throughout the hierarchy of contexts to the set of executable contexts. Thus, the global intention to *define use case model* is transformed into a set of actions which transform the product to develop the use case model. For the sake of clarity, the tree is only partially represented in figure 4.

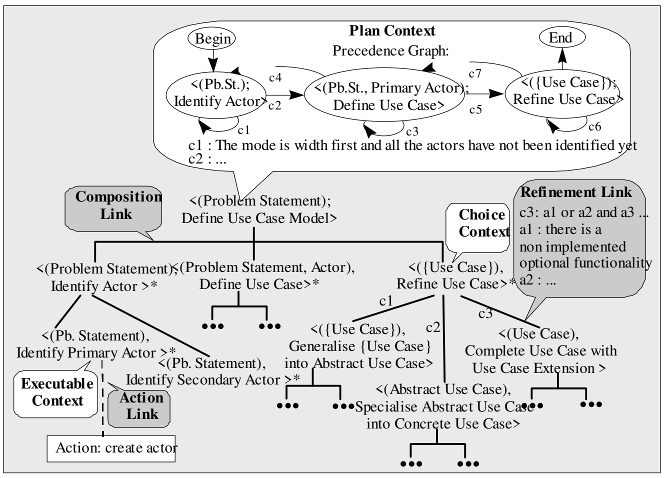

Figure 4 : Excerpt of a scenario chunk [Jacobson92]

Finally, our concrete experience in constructing the SGML scenario knowledge base has raised difficulties due to the lack of formal descriptions of either the process or the product models of the scenario approaches. Moreover, approaches rarely include the process dimension. To cope with this situation, we classify scenario chunks into formal and informal (see figure 2) and propose to associate to informal chunks textual explanations on how to fulfill the intention of this type of scenario. For example, in figure 5 we present a scenario chunk defined according to Kyng's approach [Kyng95]. This chunk proposes to describe work situations in which the users identify some problems and bottlenecks. The author does not detail how to work with these scenarios, he only explains what type of information these scenarios have to contain.

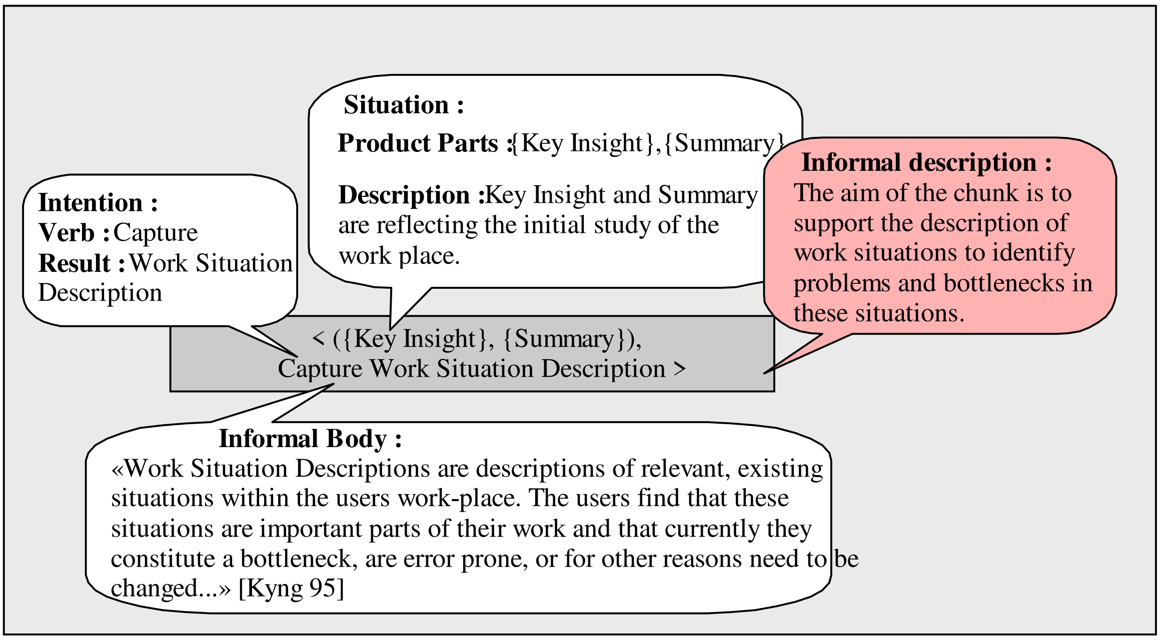

Figure 5 : Scenario chunk from Kyng's approach

2.3 Scenario chunk granularity

Figure 6 sums up the three different levels of granularity of scenario chunks : *contexts*, hierarchies of contexts called *trees* which may be parts of a *scenario approach*.

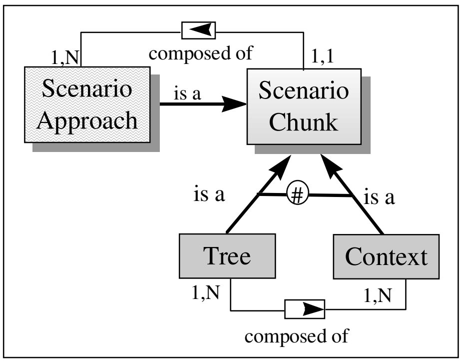

Figure 6 : Structure of the scenario knowledge in the scenario method base

Each of these chunks can be considered either independently or as part of an overall scenario approach. A scenario based approach is viewed itself as a chunk (is-a link in figure 6). Indeed, both the approach itself and its component chunks are reusable. Typically, a scenario approach contains guidelines for the creation, the transformation and the refinement of scenarios into more conceptual products. For example, in the OOSE methodology [Jacobson92], we identified two scenario chunks, one to construct the use case model and a second one to construct the analysis model out of the use case model. The composition of these two chunks corresponds to a scenario approach which is also proposed in the method base as another chunk.

3. Scenario method meta-knowledge level

The scenario method knowledge is about descriptions of available scenario method chunks. The scenario method meta-knowledge we are dealing with in this section aims at specifying the context in which method knowledge can be (re)used.

Assuming that the scenario base has been constructed, the question addressed now is « how to ease the retrieval of scenario chunks meeting the requirements of a method engineer who wants to extend an existing method with scenario features ? ». This raises the need for understanding when, why and how a specific scenario chunk can be reused i.e. to specify the context of its use. Our literature survey [Rolland97] as well as the industrial visits performed within the companies of the CREWS steering committee [Jarke97] have shown that this knowledge is not available. Both have also demonstrated that there is an explicit need for making this knowledge available. Our view is that the knowledge about the context of use of scenario chunks shall be formalized and stored in the scenario method base with the scenario chunks themselves. We call this knowledge method meta-knowledge as it provides information characterizing the use of scenario method knowledge. The scenario method base is therefore organized at two levels, the method meta-knowledge level and the method knowledge level. In the process of reusing scenario chunks, these two levels serve in separate steps. The method meta-knowledge supports the retrieval step whereas the knowledge is the material effectively reused and integrated in the existing method.

In this section we are concerned with the meta-knowledge representation. We shall illustrate the use of this meta-knowledge in section 4 through sgmlQL queries acting on the implemented method base.

We use the notion of descriptor [De Antonnellis91] as a means to describe scenario chunks. A descriptor plays for a scenario chunk the same role as a meta-class does for a class. Our concept of descriptor is similar to the one of faceted classification schema [Pietro-Diaz87] developed in the context of software reuse.

We extend the contextual view used to describe the chunk interface to structure the meta-knowledge in the descriptor. Indeed, we view the retrieval 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 fulfill which intention*. If we remember that scenario based approaches primarily aim at supporting specific design activities in different ways, the descriptor situation shall refer to this design activity whereas the intention expresses a design goal related to this activity. As an example, the descriptor of the Jacobson's chunk described in section 2 shall refer to 'analysis' as a design activity supported by the chunk and 'capture user/system interactions' as the intention within this activity which can be supported by the use case approach provided by the chunk. Then, our descriptor is contextual as it captures situational and intentional knowledge defining the context of reuse a scenario method chunk.

Figure 7 gives an overview of the knowledge captured in the method base for every chunk. The chunk body is actually the fragment of method to deal with a specific type of scenario whereas the chunk interface describes its conditions of applicability, the situation required as input of the chunk, and the intention the chunk helps to fulfill. These two aspects constitute the scenario method knowledge whereas the meta-knowledge is captured in the scenario descriptor. The descriptor expresses the reusability conditions of the chunk by characterizing the design activity in which it can be reused (the situation part) and the design intention that can be supported by the scenario chunk (the intention part). It describes the broad picture in which the scenario approach captured in the chunk can take place. In the sequel, we develop the chunk descriptor in detail.

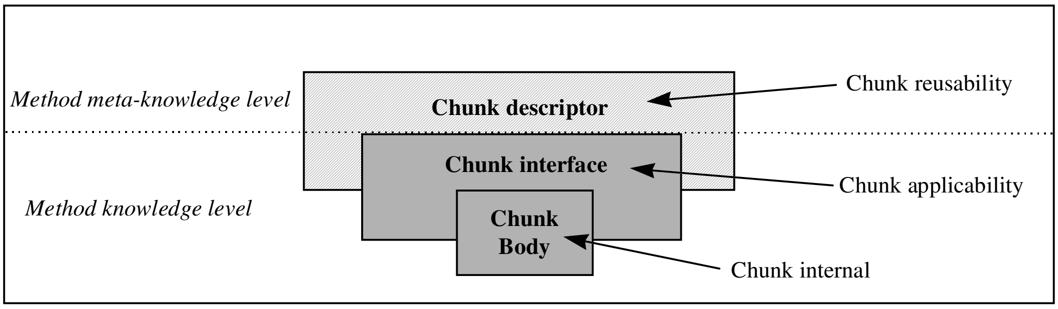

Figure 7 : Chunk overview

3.1 The descriptor structure

Figure 8 depicts the descriptor structure. A chunk descriptor has a *situation* part and an *intention* part that we consider in turn.

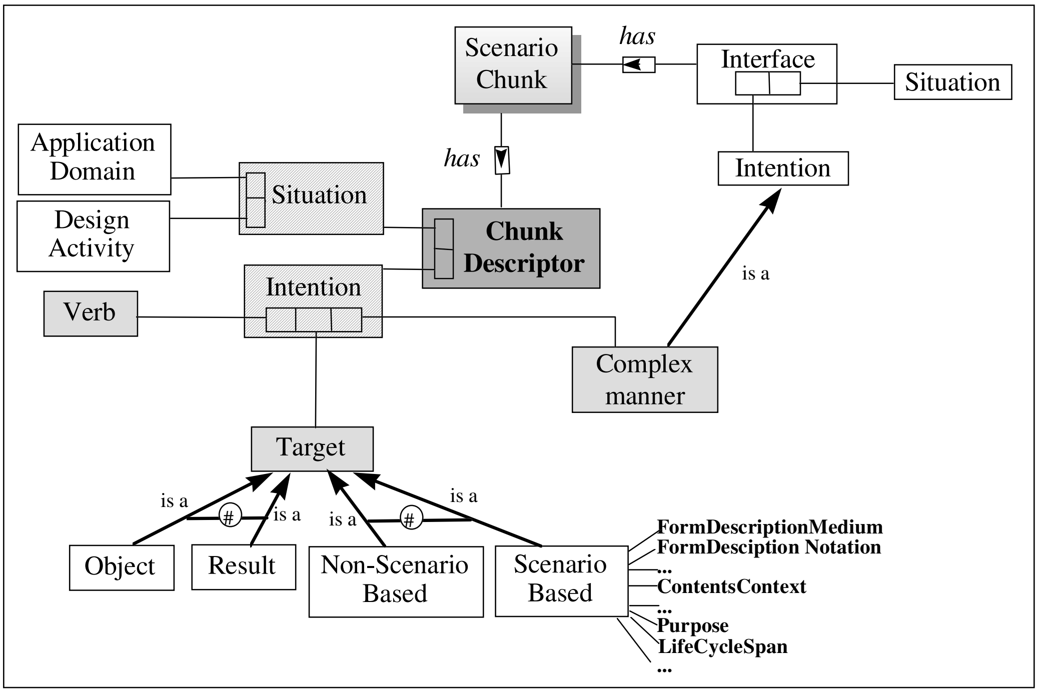

Figure 8 : The chunk descriptor structure

The situation part of a chunk descriptor

The *situation* part of a descriptor comprises two aspects (figure 8) : the *application domain* and the *design activity* in which the scenario chunk is relevant. For instance, considering the Jacobson's chunk (figure 4) which describes how to proceed for defining a use case model, the domain of the descriptor is *Object Oriented Applications* and its design activity is *Analysis*. This means that this chunk can be reused in Object Oriented Application for facilitating the Analysis step.

While populating the scenario method base, we have identified a list of application domains in which scenarios are used. Table 1 presents the current list of domains in the implemented scenario base. HCI (see [Caroll95] for a survey) and OO applications [Cockburn95], [Glinz95], [Jacobson92], [Lalioti95], [Regnell95], [Robertson95], [Rubin92], [Rumbaugh91], [Wirfs-Brook95], [Leite97] are the two domains where scenarios are nowadays extensively used.

Usability Engineering
OO applications
Requirements Engineering
HCI (Human Computer Interfaces)
Workflow applications
Critical systems
Information systems
Socio-technical applications

Table 1 : Application domains for scenario use

Similarly we have identified a list of design activities, (similar to the one proposed in [Caroll95], Table 2) each of which is supported by at least one scenario chunk.

Design Activity	**Scenario Based Approach**
Analysis	[Caroll 95],[Cockburn 95], [Glinz 95], [Jacobson 92], [Kyng 95], [Regnell 95], [Robertson 95], [Rubin 92], [Rumbaugh 91], [Wirfs-Brook 95]
Envisionment	[Jacobson 92], [Nielsen 95],[Kyng 95], [Karat 95]
Requirement Elicitation	[Holbrook 90], [Jacobson 92], [Johnson 95], [Kyng 95], [Potts 94],
Design Rationale	[Nielsen 95], [Kyng 95]
Validation	[Holbrook 90], [Glinz 95],[Lalioti 95], [Nielsen 95]
Software Design	[Holbrook 90],[Hsia 94]
Software Testing	[Kyng 95]
Team Work Building	[Filippidou 97]
User-Designer Communication	[Holbrook 90], [Jacobson 92], [Potts 94], [Erickson 95]
Documentation / Training	[Kyng95], [Potts94]

Table 2 : Design activities covered by different scenario chunks

The intention part of a chunk descriptor

The chunk descriptor intention expresses how the scenario approach encapsulated in the chunk participates to the achievement of the design activity. For example, the intention of the descriptor of the Jacobson's chunk presented in figure 4 is 'capture user/system interactions' as the chunk provides a scenario based approach supporting the capture of the interactions between the future system and its users. The descriptor intention is an expression of the role that a scenario approach can play in a particular design activity. We found in our survey of both literature and practice a large panel of roles, all being informally expressed and therefore difficult to classify and organize to support the retrieval process in the method base. Table 3 gives some examples of our findings.

Supporting the analysis of the users workplace and work situations
Expressing how a system being designed should look like and behave
Facilitating the discovery of user needs
Helping evaluating possibilities for usability and functionality
Supporting the identification of central problem domain objects
Helping to develop cohesion in the team
Facilitating the communication on the systems problems between users and designers
Helping to test whether the system satisfies or not all the user's requirements
Supporting user training
Bridging the gap between the system presented as an artifact and the tasks the users want to accomplish using it

Table 3 : Examples of scenario roles

Instead of using role names as described in the literature, we use a more formal description of intentions based on [Prat97] leading to the intention structure presented in figure 8. This structure is compatible with the one used for the chunk interface. It is extended in order to link the intention of the chunk and the intention of its descriptor. The intention in the chunk descriptor is specified by the intention *verb*, the *target* of by this intention and the *manner* to satisfy this intention (figure 8). Let us detail these various elements of the intention structure in turn.

Similarly to the target in the scenario chunk interface (see figure 3), the *target* of the descriptor is specified into *object* or *result* depending on the role played by the target for the verb. These roles have been explained in section 2. Moreover, in the chunk descriptor intention we make the distinction between *non-scenario based target* and *scenario based target* (see *is a links* in figure 8*)*

- *Non-scenario based targets* represent product parts other than scenarios. Functional system requirements, non functional system requirements, object model, alternative design option, etc. are examples of non-scenario based targets.
- *Scenario based targets* represent product parts of the scenario type. Use case, scenario script, episode, work situation description or use scenario are examples of scenario based targets. In order to ease the retrieval process, there is a need for characterizing scenario targets with enough details to differentiate one from the other. Our characterization is based on the framework defined in the CREWS project [Rolland97]. Properties such as the scenario formality, the level of abstraction, the nature of interactions, the covered requirements or the scenario life cycle are proved necessary to select more precisely the adequate scenario chunks. There are eleven properties which are surveyed in appendix 1.

The chunk descriptor intention is completed by a *manner* which is a *complex manner* by opposition to a *simple manner* as used in the scenario chunk interface. A *complex manner* is recursively described as an intention. The intention to *Capture user/system interactions by defining a use case model with Jacobson's refinement strategy* is an example of descriptor intention using a complex manner. The manner (*by defining a use case model with the Jacobson's refinement strategy)* is recursively defined as an intention (*defining*) with a result (*use case* model) and a manner (*with the Jacobson's refinement strategy*). The intention is denoted as follows : *Capture (User/System Interactions)*$_{Res}$ *(by Defining (Use Case Model)*$_{Res}$ *(with Jacobson's Refinement Strategy)*$_{Man}$ *)*$_{Man}$.

The descriptor intention always refers to a complex manner. This allows us to link the scenario chunk intention to the descriptor intention. This is modeled in figure 8 by an *is a* link from the manner to the chunk interface intention. In the example above, the intention to *define use case model with Jacobson's refinement strategy* is the intention of the Jacobson's chunk presented in figure 4. It is embedded in the descriptor intention as the manner of the intention to *Capture user/system interactions.* The scenario approach captured in a given scenario chunk is formally defined as the manner to achieve a design intention. Both the design intention and the manner to achieve it are expressed in the descriptor intention.

3.2 Examples

In figure 9, we present the example of the descriptor corresponding to the *Define Use Case Model* scenario chunk depicted in figure 4. This descriptor tells us that the corresponding scenario chunk is useful in the domain of object oriented applications for supporting the analysis activity. The intention in this situation is to capture the interactions between the user and the system. Moreover, the intention is clarified by the manner to fulfill it. This manner defines precisely the way the scenario chunk will help fulfilling the intention of user/system interaction capture : it is by defining a use case model in the Jacobson's style. Because the target of this intention (*define use case model*) is a scenario based one, the descriptor specifies its discriminant properties, namely the FormDescriptionMedium (textual), the FormDescriptionNotation (informal), the ContentsCoverage (Functional), the ContentsContext (user/system interactions), the LifeCycleSpan (persistent) and the Purpose (descriptive).

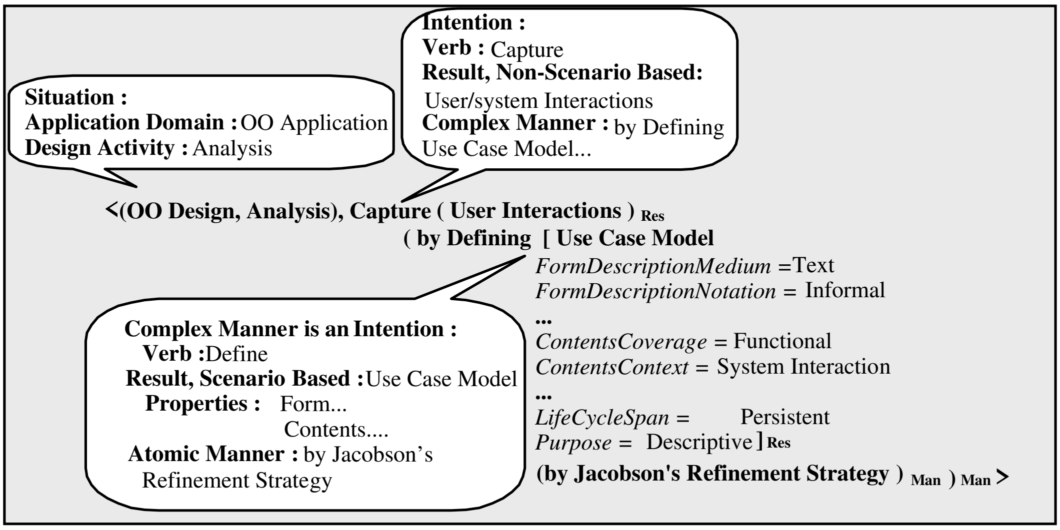

Figure 9: Example of the Jacobson's chunk descriptor

As another example of chunk descriptor, figure 10 presents the descriptor associated to the informal Kyng's scenario based approach whose chunk was sketched in figure 5.

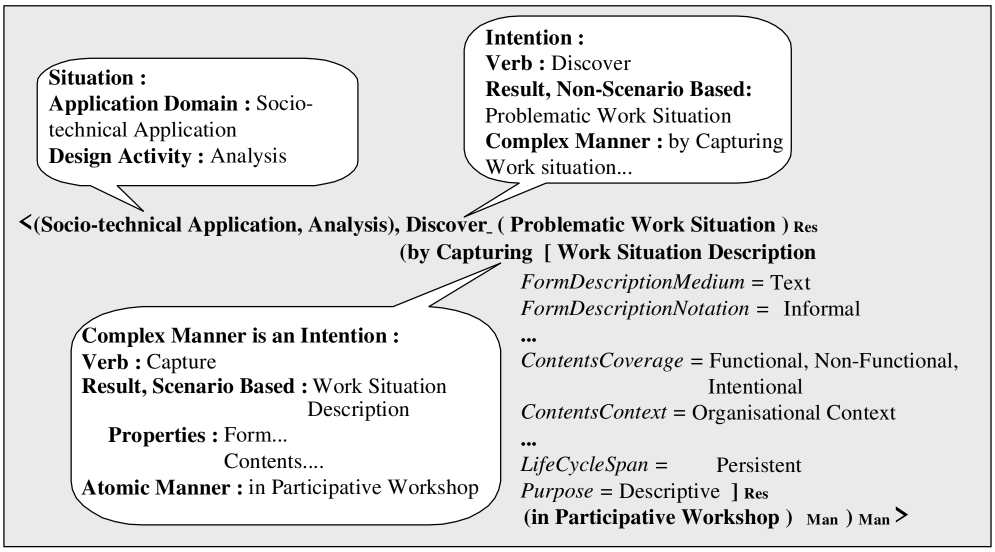

Figure 10 : Descriptor of the Kyng's chunk

4 Using SGML to implement and query the scenario method base

SGML (Standard Generalized Markup Language) [Goldfarb90] is an international standard language to describe a document using a set of mark ups defined in a grammar. SGML documents are structured as trees. We found the language adequate for representing our scenario method base. Besides SgmlQL [Lemaitre95] is available to query an SGML base of documents. We sketch in this section the SGML structure of our implemented scenario method base and will sketch the query process in the next section.

4.1 Overview of the SGML structure of the scenario method base

The structure of the scenario method base is described in a DTD (Data Type Definition). The whole DTD is a *document type*. It is composed of *elements* which are characterised by a *mark up identifier*, *constraints* on the existence of the opening and closing tags and the definition of their structure, i.e. the component elements. An element can recursively be composed of other elements. Based on this principle, the scenario method base is represented as a document type named METHODBASE (see Table 4) which is composed of the element named DESCRIPTIONS. This element consists in a set of descriptions which are themselves called DESCRIPTION. Thus, the structure of the element DESCRIPTIONS is represented by DESCRIPTION* where the « * » means many.

```
< ! DOCTYPE METHODBASE [
        < ! ELEMENT DESCRIPTIONS           - -    (DESCRIPTION*)>
        < ! ELEMENT DESCRIPTION            - -    (META_KNOWLEDGE_LEVEL,
                                                            KNOWKEDGE_LEVEL)>
        < ! ELEMENT META_KNOWLEDGE_LEVEL   - -    (DESCRIPTOR)>
        < ! ELEMENT KNOWLEDGE_LEVEL        - -    (CHUNK|APPROACH)>
...                                                                    ]>
```

Table 4 : overview of the SGML structure of the scenario base

This way of modelling is recursively applied to integrate all the elements composing the document type. Thus, the DESCRIPTION element is characterised by a meta-knowledge level (META_KNOWLEDGE_LEVEL) and a knowledge level (KNOWLEDGE_LEVEL) which are, as presented in the previous sections, respectively composed of a *descriptor* (DESCRIPTOR), and of either a *chunk* or an *approach* denoted by (CHUNK | APPROACH). The resulting structure of the METHODBASE document type is the tree presented in figure 11.

It is possible to attach *attributes* to elements to characterise them. For example, the attribute TYPE attached to the element CHUNK characterises its type (FORMAL, INFORMAL). An attribute has a name, a type (enumerated or not) and may be optional (#REQUIRED, #IMPLIED). Underneath is the Sgml description of the mandatory attribute TYPE of CHUNK.

```
< ! ATTLIST   CHUNK        TYPE          (FORMAL | INFORMAL)      #REQUIRED>
```

The overall description of the document type defined for the scenario base is provided in the appendix 2. In the following section, we illustrate the Sgml document contents with the Jacobson's chunk and its associated descriptor.

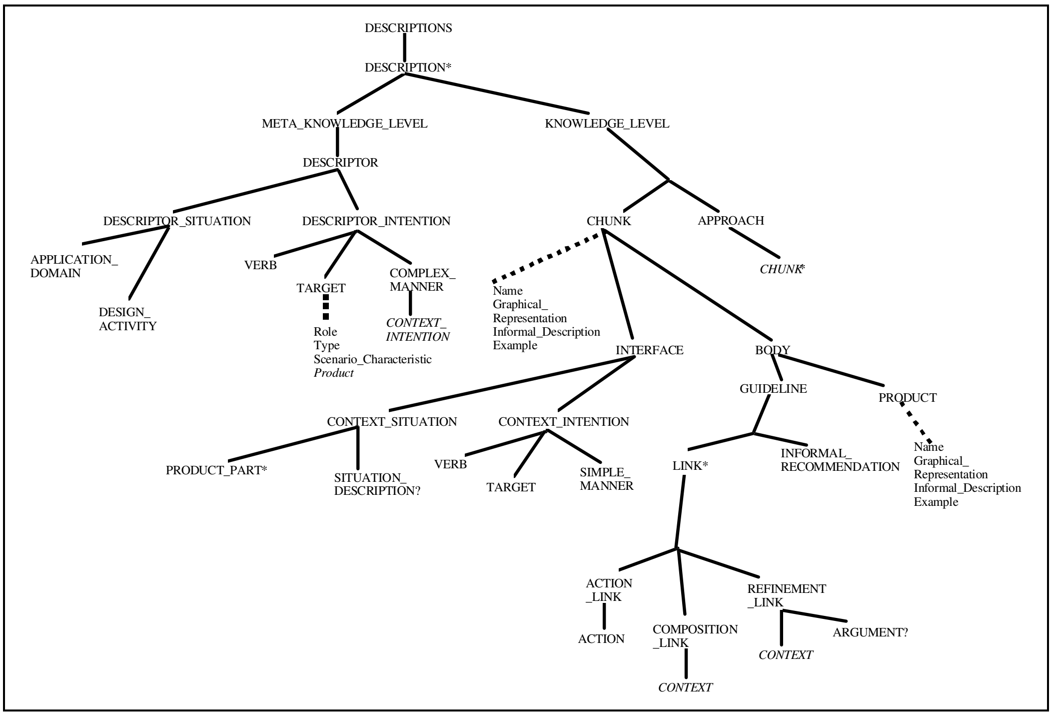

Figure 11 : Overview of the SGML structure of the scenario method base

4.2 Examples of Sgml chunks and chunk descriptors

Let us start with chunk descriptors. As explained in section 3, a chunk descriptor has a situation part and an intention part. According to our approach, the *situation part* (see DESCRIPTOR_SITUATION in Table 5) is characterised by an *application domain* (APPLICATION_DOMAIN) and by a *design activity* (DESIGN_ACTIVITY) which explain respectively, the area and the design activity in which the chunk can be reused. In our example, the area is *Object Oriented Applications* and the activity is *Analysis*.

<DESCRIPTOR_SITUATION >
<APPLICATION_DOMAIN>Object oriented applications</APPLICATION_DOMAIN>
<DESIGN_ACTIVITY>analysis</DESIGN_ACTIVITY>
</ DESCRIPTOR_SITUATION >

Table 5 : Example of descriptor situation

The *intention part* of the descriptor (DESCRIPTOR_INTENTION) is illustrated in Table 6. It is composed of :

- a *verb* (VERB), to *Capture* in our example,

- a *target* (TARGET) which can either play the role of a result or of an object. This information is denoted in the attribute *role* of the target by the values « *result* » and « *object* ». In the intention *Capture user/system Interactions*, the target *User Interactions* is considered as a result.

Moreover, the target is either a scenario-based product (like *Use Case Model* in Table 6) or not (like *User/system Interaction* in Table 6). In the former case the target has a number of characterising properties represented as attributes (*FormDescriptionMedium, FormDescriptionNotation, ...*) attached to the TARGET element.

- a *manner* (COMPLEX_MANNER)

As presented in section 3, the manner of the intention in the chunk descriptor is a *complex manner* (COMPLEX_MANNER) whereas the one of the intention in the chunk interface is a *simple manner* (SIMPLE_MANNER). A simple manner is represented by a string (#PCDATA) whereas the complex one has the structure of an intention. In Table 6, « *Jacobson's Refinement Strategy* » is a simple manner whereas « *by Defining Use Case Model by Jacobson's Refinement Strategy* » is a complex one.

```
<DESCRIPTOR_INTENTION>
<VERB>Capture</VERB>
<TARGET role = « object »
      type = « non scenario-based »>user interactions</TARGET>
<COMPLEX_MANNER>
    <VERB>Defining</VERB>
<TARGET role = « result »
            type = « scenario_based »
            FormDescriptionMedium = « text »
            FormDescriptionNotation = « informal »
            FormPresentationAnimation = « false »
            FormPresentationInteractivity = « none »
            ContentsCoverage = « functional »
            ContentsContext  = « system interaction »
            ContentsAbstraction = « type »
            ContentsArgumentation « false »
            LifeCycleSpan = « persistent »
            LifeCycleOperation = « refinement »
            Purpose = « descriptive »>Use Case Model</TARGET>
<SIMPLE_MANNER> by Jacobson's Refinement Strategy</SIMPLE_MANNER>
</COMPLEX_MANNER>
</DESCRIPTOR_INTENTION>
```

Table 6 : Example of Sgml descriptor intention

Now that we are aware of the Sgml description of the meta-knowledge level of the scenario method base, let's concentrate on the knowledge level. The knowledge level (KNOWLEDGE_LEVEL) (see figure 11) is represented in the Sgml structure either by a *chunk* element (CHUNK) or by an *approach* (APPROACH) which is composed of chunks (CHUNKS*).

As illustrated in Table 7, the *chunk* (CHUNK) element contains two parts : an *interface* (INTERFACE) and a *body* (BODY). It has general characteristics, namely a *name*, a *type* (formal, informal), an *informal description* and a reference to a *graphical representation*.

```
<CHUNK name = « Define Use Case Model by Jacobson's Refinement Strategy »
        type = « formal »
        informal description = « Defining a use case model by Jacobson's refinement strategy consists in identifying actors, then in constructing use cases out of this actors and finally in refining the use cases constructed before »>
<GRAPHICAL_REPRESENTATION><A HREF= « JacobProc1.gif »></A>
</ GRAPHICAL_REPRESENTATION>
 <INTERFACE>
<CHUNK_SITUATION>
        <PRODUCT_PART>Problem Statement</PRODUCT_PART>
        <SITUATION_DESCRIPTION>The problem statement is an initial textual and informal description of the expectations about the future system. It contains some requirements and constraints resulting from interviews with the end users </SITUATION_DESCRIPTION>
</CHUNK_SITUATION>
<CHUNK_INTENTION>
   <VERB>Define</VERB>
   <TARGET role= « result »
          type = « scenario-based »>Use Case Model</TARGET>
   <SIMPLE_MANNER>by Jacobson's Refinement Strategy</SIMPLE_MANNER>
</CHUNK_INTENTION>
<BODY><PRODUCT name = « Use case model product »
informal description = « A use case model is composed of one to n actor(s) and one to n use case(s). An actor is associated to one and only one use case model. Jacobson distinguishes two kinds of actors : the primary actors who execute the use cases and the secondary actors who play an indirect role in the execution of use cases.
A use case belongs to one and only one use case model, it is characterised by a topic and a description. There are five different types of use cases: the abstract, the concrete, the basic, the alternative, and the extension use cases. Concrete use cases use abstract use cases ; basic use cases have alternative use cases, and extension use cases extend other use cases. »>
   <PRODUCT_GRAPHICAL_REPRESENTATION> <A HREF= « JacobProd.gif »></A>
</PRODUCT_GRAPHICAL_REPRESENTATION></PRODUCT>
<GUIDELINE>
<LINK><COMPOSITION_LINK><A HREF= IdentifyActor.gif></A>
</COMPOSITION_LINK></LINK>
<LINK><COMPOSITION_LINK><A HREF= DefineUseCase.gif></A>
</COMPOSITION_LINK></LINK>
<LINK><COMPOSITION_LINK><A HREF= RefineUseCase.gif></A>
</COMPOSITION_LINK></LINK>
</GUIDELINE> </BODY></CHUNK>
```

Table 7 : Example of Sgml description of a chunk

The *interface* is composed of two parts : a *situation* (CHUNK_SITUATION) and an *intention* (CHUNK_INTENTION).

The *situation* of the chunk interface (CHUNK_SITUATION) is composed of two elements :
- one or several *product parts* referenced by PRODUCT_PART* in the Sgml tree, and
- a *description* (SITUATION_DESCRIPTION) which is optional.

All these elements are strings (#PCDATA).

The *intention* of the chunk (CHUNK_INTENTION) is composed of a *verb* (VERB), a *target* (TARGET) and a *simple manner* (SIMPLE_MANNER). This is exemplified in Table 7.

Following our definitions in section 2, the *body* is composed of two parts, the *product* and the *guideline*.

- 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. This graphical representation is referenced in Table 7 by JacobProd.gif and is presented in figure 12.

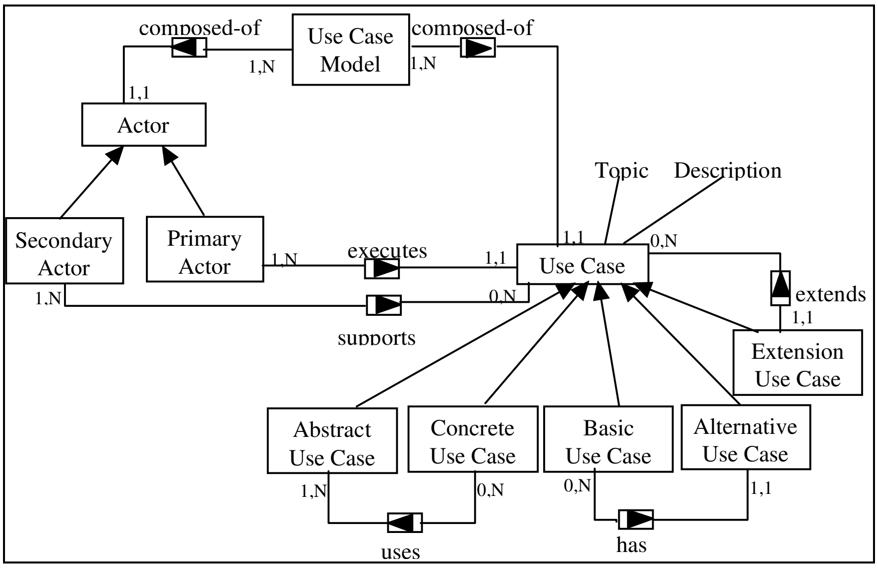

Figure 12 : JacobProd.gif

- The guideline (GUIDELINE) can be either represented by an informal description (INFORMAL_RECOMMENDATION) or by a set of links (LINK*) depending on whether the chunk is informal or not. In the case of a formal chunk, the guideline has the form of either a *context* or a *tree of contexts*. It is represented in the Sgml structure by the set of *links*, connecting the contexts one with the others in the tree. Depending on the type of its source context, a *link* can be either a *composition*, a *refinement* or an *action link* (Table 7). The tree structure can be visualised through the *graphical representation* (GRAPHICAL_REPRESENTATION) element of the structure. This graphical representation is referenced in Table 7 by JacobProc1.gif and is presented in figure 13.

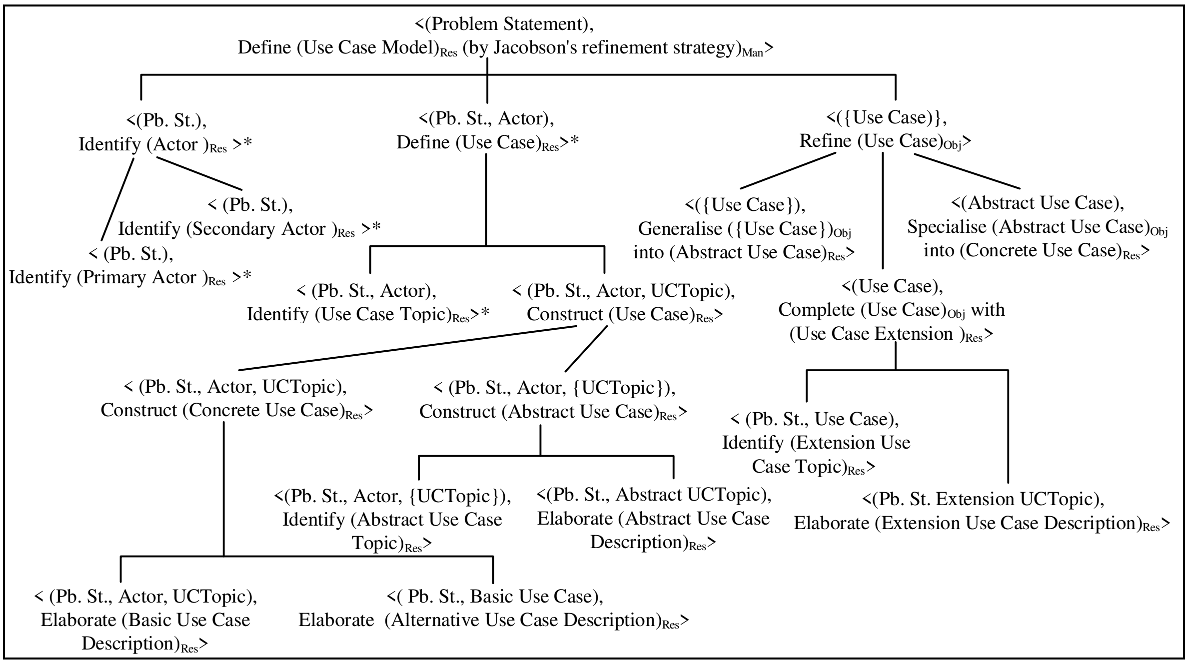

Figure 13 : JacobProc1.gif

5. Examples of SgmlQL queries

This section illustrates the way SgmlQL queries can support the process of reusing scenario chunks. This illustration is based on a reuse scenario that can be described in the following way. Let's assume that Mr Bean,. a requirements engineer is in charge of a project for developing an object oriented application. In the early phase of the project the project used the Enterprise Modelling approach [Bubenko94] to model the rich picture of the organisational context in which the system will operate. The result of this step is a set of high level goals. Mr Bean is now faced to the operationalisation of these goals i.e, the specification of the system functions which fulfill these goals. The project constraint is that the system functional specification should be done in an object oriented manner. Mr Bean's belief is that the project should benefit from using a scenario based approach. The argument in favour of such an approach is twofold : (1) the system is too large and complex to be tackled in a global way and, (2) the domain experts are not familiar with semi-formal or formal notations and prefer to communicate with requirements engineers in an informal manner.

Thus, we propose to help Mr Bean by querying the Sgml scenario method base to retrieve chunks that match his requirements i.e. chunks which help bridging the gap between goal models and object oriented specifications. We first propose to extract from the method base all the chunks whose situation is based on goals. The query is formulated as follows :

Q1 : **Select the chunks having goals as product parts of their situation.**
select text($a->NAME)
from $d in every DESCRIPTION within $myfile[4], $a in every APPROACH within $d,
$pp in first PRODUCTPART within $a
where text($pp) match « goals » ;

The answer to this query proposes two chunks, the :

- Holbrook's, and
- Cockburn's ones.

[4] A preliminary query (*global $myfile = file « MethodBase.sgml »)* should be typed in order to indicates which sgml document (here *MethodBase.sgml)* is queried.

Mr Bean is not familiar with the Cockburn's approach, but he heard of the Holbrook's one and wish to explore further the possibilities offered by this approach. As the constraint is to produce an object oriented specification, the question is to identify which is the output of the Holbrook's chunk. Q2 gives the answer to this question.

Q2 : **Select the target generated by the « Holbrook » chunk.**

```
select text(first TARGET within $cm)
from $d in every DESCRIPTION within $myfile, $descro in every DESCRIPTOR within $d,
$cm in every COMPLEXMANNER within $descro, $a in every APPROACH within $d
where text($a->name) match « Holbrook » ;
```

The target is:
Use Scenario

An access to the PRODUCT part of the chunk in the Sgml document (to the HolbProd.gif in particular) convinces Mr Bean that unfortunately, the output of the Hoolbrook's chunk is not an object oriented specification. Thus, we suggest to search for another scenario based chunk that supports the transformation of input scenarios into object-oriented models. This can be done with query Q3 , presented below.

Q3 : **Select chunks which are using scenario or scenario-based product as input and generate an analysis model as output.**

```
select text($c->NAME)
from $d in every DESCRIPTION within $myfile, $descro in every DESCRIPTOR within $d,
$c in every CHUNK within $d, $pp in first PRODUCTPART within $c
where ((text($pp) match « scenario» ) or (text($pp) match « use-case» )) and
(text(first TARGET within $descro) match « analysis model ») ;
```

This query results in the selection of the chunk named « *Define analysis model* ». Mr Bean is happy and decides to explore further on the solution consisting of combining the *Holbrook's* chunk based on *use scenarios* with the *Define analysis model* chunk supporting construction of an *analysis model* out of scenarios.

Even short and schematic, this example illustrates the use of SgmlQL queries for retrieving scenario base chunks meeting the requirements of a user. It suggests at least two comments : (1) the need for a thesaurus to support an efficient query process and (2) the possibility to capitalise from experience, for instance, by inserting in the method base the new approach when fully generated. This insertion can be done using specific commands provided by SgmlQl.

6. Conclusion

In this paper we propose an approach for supporting the reuse of scenario based chunks made available in a scenario method base. The motivation for developing such an approach was twofold : first, there exists a large corpus of available scenario-based approaches that has not been formalised yet, and secondly there is an explicit need for incorporating scenario-based approaches in existing methods. FUSION, OMT and UML are examples of such enhancements that the method engineers would like to reproduce.

The proposed approach advocates a modular representation of scenario chunks and an intentional description of their reuse context. The former results is cohesive chunks which are applicable in specific situations for specific purposes whereas the latter provides contextual information identifying in which specific design situations for which specific design intentions the chunks are reusable. The paper also reports on the implementation of a scenario method base in SGML and illustrates the reuse process through an example.

Future work shall concentrate on developing guidelines to integrate scenario chunks in existing methods and the implementation of these guidelines in the Sgml context. Besides, in order to support the process for retrieving chunks matching specific requirements we are developing a set of SgmlQL macro-queries. Finally we shall work on an HTML presentation of the query responses.

7. Références

[Booch97] : G. Booch, J. Rumbaugh, I. Jacobson, *«Unified Modeling Language »,* version 1.0, Rational Software Corporation, Jan. 1997.

[Bubenko94] : J. Bubenko, C. Rolland, P. Loucopoulos, V. De Antonellis, « Facilitating 'Fuzzy to Formal' Requirements Modelling », Proc. of the First International Conference on Requirements Engineering, April 1994, Colorado Springs, Colorado.

[Carroll95] J.M. Caroll, *« The Scenario Perspective on System Development »,* in J.M. Carroll (ed.*), Scenario-Based Design: Envisioning Work and Technology in System Development* (1995).

[Cockburn95] 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).

[Coleman97] : D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes, *«Object-Oriented Development : The FUSION Method »*, Prentice Hall, 1994.

[De Antonellis91] : V. De-Antonellis., B. Pernici, P. Samarati, (1991*) «F-ORM METHOD : A Methodology for Reusing Specification»,* In Object Oriented Approach in Iforamtion Systems, Van Assche F., Moulin b., Rolland C. (eds), North Holland, 1991

[Erickson95] : T. Erickson, *« Notes on Design Practices : Stories and Prototypes as Catalysts for Communication »*, in J.M. Carroll (ed.*), Scenario- Based Design: Envisioning Work and Technology in System Development* (1995).

[Filippidou97] : D. Filippidou, P. Loucopoulos, *«Using Scenarios to Validate Requirements in a Plausibility-Centred Approach »,* Proc. Of the 9Th Conference on Advanced Information Systems Engineering, Barcelona, Catalonia, Spain, June 1997.

[Glinz95] M. Glinz, *«An Integrated Formal Model of Scenario based on Statecharts»,* Lecture Notes in Computer Science'95,pages 254-271, 1995.

[Goldfarb90] : C. F. Goldfarb, *« The SGML Handbook »,* Oxford Clarendon Press, 1990.

[Holbrook90] C. H. Holbrook_III, *« A Scenario-Based Methodology for Conducting Requirement Elicitation»,* ACM SIGSOFT Software Engineering Notes, 15(1), pp.95-104, 1990.

[Hsia94] Hsia P, Samuel J, Gao J, D., Toyoshima, Y. and Chen ,C . (1994*) «Formal Approach to Scenario Analysis»,* IEEE Software, **11**, 33-41

[Jacobson92] I. Jacobson, M. Christerson, P. Jonsson and G. Oevergaard*, « Object Oriented Software Engineering: a Use Case Driven Approach »,* (Addison-Wesley, 1992).

[Jacobson95a] I. Jacobson, *« The Use Case Construct in Object-Oriented Software Engineering »,* in John M. Carroll (ed.), *Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 309-336.

[Jacobson95b] I. Jacobson, M. Ericsson and A. Jacobson*, « The Object Advantage, Business Process Reengineering with Object Technology»* (Addison-Wesley Publishing Company, 1995).

[Jarke97] : M. Jarke, K. Pohl, P. Haumer, K. Weidenhaupt, E. Dubois, P. Heymans, C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyte, A. Sutcliffe, N. A. Maiden and S. Minocha, *« Scenario use in european software organisations - Results from site visits and Questionnaires »,* Esprit Reactive Long Term Research Project, 21.903 CREWS, Deliverable W1 : Industrial Problem Capture Working Group, 1997.

[Johnson95] P. Johnson, H. Johnson and S. Wilson*, « Rapid Prototyping of User Interfaces driven by Task Models »*, in John M. Carroll (ed*.), Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 209-246.

[Karat95] : J. Karat, *« Scenario Use in the Design of a Speech Recognition System »,* in J.M. Carroll (ed*.), Scenario- Based Design: Envisioning Work and Technology in System Development* (1995).

[Kyng95] : M. Kyng, Creating Contexts for Design, in John M. Carroll (ed*.), « Scenario-Based Design: Envisioning Work and Technology in System Development »* (John Wiley and Sons, 1995) 85-107.

[Lalioti95] : V. Lalioti and B.Theodoulidis, «*Use of Scenarios for Validation of Conceptual Specification*», Proceedings of the Sixth Workshop on the Next Generation of CASE Tools, Jyvaskyla, Finland, June 1995.

[Leite97] : J.C.S. do Prado Leite, G. Rossi, F. Balaguer, A. Maiorana, G. Kaplan, G. Hadad and A. Oliveros, «Enhancing a Requirements Baseline with Scenarios», In *Third IEEE International Symposium On Requirements Engineering (RE'97)*, Antapolis, Maryland (IEEE Computer Society Press, 1997) 44-53.

[Lemaitre 95] : 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.

[Nielsen95] : J. Nielsen, Scenarios in Discount Usability Engineering, in John M. Carroll (ed*.), « Scenario-Based Design: Envisioning Work and Technology in System Development»* (John Wiley and Sons, 1995) 59-85.

[Plihon95] : V. Plihon, C. Rolland, « Modelling Ways-of-Working » Proc. Of the 7th Int. Conf. On « Advanced Information Systems Engineering », (CAISE), Springer Verlag (Pub.), 1995.

[Potts94] : C. Potts, K. Takahashi and A.I. Anton, *« Inquiry-based Requirements Analysis », in IEEE Software* 11(2) (1994) 21-32.

[Prat97] : N. Prat, *«Goal Formalisation and Classification for Requirements Engineering* », Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'9 , Barcelona, june 1997.

[Prieto-Diaz87] : R. Prieto-Diaz, P. Freeman, *« Classifying Software for Reusability»,* IEEE Software, Vol 4 No 1, 1987.

[Regnell95] : B. Regnell, K. Kimbler and A. Wesslen*, «Improving the Use Case Driven Approach to Requirements Engineering»,* in *the Second IEEE International Symposium On Requirements Engineering*, York, England (I.C.S. Press, March 1995) 40-47.

[Roberston95] : S.P. Robertson, « Generating *Object-Oriented Design Representations via Scenarios Queries»,* in John M. Carroll (ed.*), Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 279-308.

[Rolland94] : 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.

[Rolland96] : C. Rolland, N. Prakash, *«A proposal for Context-Specific Method Engineering»,* IFIP TC8 Working Conference on Method Engineering, Atlanta, Gerorgie, USA, 1996

[Rolland97] : C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyte, A. Sutcliffe, N.A.M. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois and P. Heymans*, «A Proposal for a Scenario Classification Framework»,* ESPRIT Reactive Long Term Research Project 21.903 CREWS, Deliverable I1: Initial Integration Workpackage (1997).

[Rumbaugh91] : J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen*, «Object-Oriented Modeling and Design»*, (Prentice Hall, 1991).

[Rumbaugh94] : J. Rumbaugh, *« Getting started, using use cases to capture requirements »,* Journal of Object Oriented Programming, Sept. 1994.

[Rumbaugh96] : J. Rumbaugh and G. Booch, *« Unified Method »,* Notation Summary Version 0.8 (Rational Software Corporation, 1996).

[Rubin95] : K.S. Rubin and A. Goldberg, *«Object Behaviour Analysis», Communications of the ACM* 35(9) (1992) 48-62.

[Thomé93] : B. Thomé*, « Systems Engineering : Principles and Practice of Computer-based Systems Engineering »,* in B. Thomé (ed), John Wiley & Sons (1993).

[Wirfs-Brock95] : R. Wirfs-Brock*, « Designing Objects and their Interactions: A Brief Look at Responsability-driven Design »,* in John M. Carroll (ed.*), « Scenario-Based Design: Envisioning Work and Technology in System Development»* (John Wiley and Sons, 1995) 337-360.

[Young87] : M. R. Young, P. B. Barnard, *« The Use of Scenario in Human-Computer Interaction Research: Turbocharging the tortoise of Cumulative Science »*, CHI + GI 87 Human Factors in Computing Systems and Graphics Interface, Toronto, 1987.

Appendices

Appendix 1 : Characterising scenarios

In [Rolland97] a framework for scenario classification is proposed to characterise scenarios according to a certain number of facets which are grouped into views. Four views have been identified :

1. the form view,
2. the contents view,
3. the purpose view and
4. the life cycle view.

The *form view* answers the question '*in which form is a scenario expressed* ?'. The response is provided through two facets namely the description facet and the presentation facet.

- The *description facet* characterises the *level of formality* and the *medium* used for the scenario description. Texts, graphics, images, videos and software prototyping are examples of media. Note that several media can be used at the same time for describing a scenario.

- The *presentation facet* tells whether a scenario is *static or animated*, and its *interactivity* i.e. the capabilities offered to the user to control the way the scenario progresses through time.

Consequently, in the descriptor the form view is represented by four properties of scenarios namely, *FormDescriptionMedium, FormDescriptionNotation, FormPresentationAnimation* and *FormPresentation Interactivity.*

The *contents view* answers the question *'what is the knowledge expressed in a scenario ?'*. The response is provided through four facets namely the abstraction facet, the context facet, the argumentation facet and the coverage facet.

- The *abstraction facet* indicates whether the scenario is concrete, abstract or mixed.
- The *context facet* explains in which kind of context the scenarios are used. System internal, system interaction, organisational context and organisational environment are examples of contexts where the scenarios can be used.
- The *argumentation facet* indicates whether argumentation concepts are used within the scenarios or not.
- The *coverage facet* indicates the kind of information captured in the scenarios, i.e. whether it is functional, non functional or intentional. The information concerning the structure, the function and the behaviour are qualified as functional, the ones which are tackling performance, time constraints, cost constraint, user support, documentation examples, back up/recovery, maintainability, flexibility, portability, security/safety, design constraints, error handling are non functional and the information concerning goal, problem, responsibility, opportunity cause and goal dependency are said intentional.

This leads to the following properties of scenarios in the chunk descriptor : *ContentsAbstraction, ContentsContext, ContentsArgumentation* and *ContentsCoverage.*

The *purpose view* answers the question '*why using a scenario ?*'. The response is provided through three criteria. A scenario can be used

- in a *descriptive* purpose, i.e. for describing something which happens in the real world,
- in a *exploratory* purpose i.e. for constructing requirements elicitations, or
- in a *explanatory* purpose, i.e. when explanations about the rationale of these issues are required.

The purpose perspective is associated in the chunk descriptor to the property called *Purpose*.

The *life cycle view* is characterised by two facets : the life span facet and the operation facet. It explains '*how to manipulate scenarios'*.

- The *life span facet* indicates whether the scenario is transient or persistent in the RE process.
- The *operation facet* defines if and how scenarios are *captured*, *refined*, *integrated*, *expanded*, and *deleted*.

This is represented in the chunk descriptor by two properties, namely *LifeCycleSpan* and *LifeCycleOperation.*

More details on the scenario classification framework and its application on several scenario based approaches can be found in [Rolland 97].

Appendix 2 : The Scenario Method Base Structure

```
< ! DOCTYPE METHODBASE [
	< ! ELEMENT DESCRIPTIONS	- -	(DESCRIPTION*)>
	< ! ELEMENT DESCRIPTION	- -	(META_KNOWLEDGE_LEVEL ,
				KNOWLEDGE_LEVEL )>
	< ! ELEMENT META_KNOWLEDGE_LEVEL	- -	(DESCRIPTOR)>
	< ! ELEMENT DESCRIPTOR	- -	(DESCRIPTOR_SITUATION,
				DESCRIPTOR_INTENTION)>
	< ! ELEMENT DESCRIPTOR_SITUATION	-  (APPLICATION_DOMAIN, DESIGN_ACTIVITY)>
	< ! ELEMENT APPLICATION_DOMAIN	- -	(#PCDATA)>
	< ! ELEMENT DESIGN_ACTIVITY	- -	(#PCDATA)>
	< ! ELEMENT DESCRIPTOR_INTENTION	- -	(VERB, TARGET, COMPLEX_MANNER)>
	< ! ELEMENT (VERB | TARGET)	- -	(#PCDATA)>
	< ! ELEMENT COMPLEX_MANNER	- -	(VERB, TARGET, SIMPLE_MANNER)>
	< ! ELEMENT SIMPLE_MANNER	- -	(#PCDATA)>
	< ! ELEMENT KNOWLEDGE_LEVEL	- -	(CHUNK | APPROACH)>
	< ! ELEMENT CHUNK	- -	(INTERFACE, BODY,
				GRAPHICAL_REPRESENTATION)>
	< ! ELEMENT INTERFACE	- -	(CONTEXT_SITUATION,
				CONTEXT_INTENTION)>
	< ! ELEMENT CONTEXT_SITUATION	- -	(PRODUCT_PART*,
				SITUATION_DESCRIPTION?)>
	< ! ELEMENT PRODUCT_PART	- -	(#PCDATA)>
	< ! ELEMENT SITUATION_DESCRIPTION	- -	(#PCDATA)>
	< ! ELEMENT CONTEXT_INTENTION	- -	(VERB, TARGET, SIMPLE_MANNER)>
	< ! ELEMENT BODY	- -	(PRODUCT, GUIDELINE)>
	< ! ELEMENT PRODUCT	- -	(GRAPHICAL_REPRESENTATION)>
	< ! ELEMENT GRAPHICAL_REPRESENTATION	- -	(#PCDATA)>
	< ! ELEMENT GUIDELINE	- -	(INFORMAL_RECOMMENDATION | LINK*)>
	< ! ELEMENT INFORMAL_RECOMMENDATION	- -	(#PCDATA)>
	< ! ELEMENT LINK	- -	(COMPOSITION_LINK | REFINEMENT_
				LINK, ACTION_LINK)>
	< ! ELEMENT COMPOSITION_LINK	- -	(#PCDATA)>
	< ! ELEMENT REFINEMENT_LINK	- -	(#PCDATA, ARGUMENT?)>
	< ! ELEMENT ARGUMENT	- -	(#PCDATA)>
	< ! ELEMENT ACTION_LINK	- -	(ACTION)>
	< ! ELEMENT ACTION	- -	(#PCDATA)>
	< ! ELEMENT APPROACH	- -	(CHUNK*)>
	< ! ATTLIST	PRODUCT	NAME	(#PCDATA)	#REQUIRED>
	< ! ATTLIST	PRODUCT	INFORMAL_DESCRIPTION	(#PCDATA)	#REQUIRED>
	< ! ATTLIST	PRODUCT	EXAMPLE	(#PCDATA)	#IMPLIED>
	< ! ATTLIST	TARGET	ROLE	(OBJECT | RESULT)	#IMPLIED>
	< ! ATTLIST	TARGET	TYPE	(SCENARIO-BASED | NON SCENARIO_BASED)
					#IMPLIED>
	< ! ATTLIST	TARGET	FORMDESCRIPTIONMEDIUM	(#PCDATA)	#IMPLIED>
	< ! ATTLIST	TARGET	FORMDESCRIPTIONNOTATION	(#PCDATA)	#IMPLIED>
	< ! ATTLIST	TARGET	FORMPRESENTATIONANIMATION (#PCDATA)	#IMPLIED>
	< ! ATTLIST	TARGET	FORMPRESENTATIONINTERACTIVITY (#PCDATA)
					#IMPLIED>
	< ! ATTLIST	TARGET	CONTENTSABSTRACTION	(#PCDATA)	#IMPLIED
	< ! ATTLIST	TARGET	CONTENTSCONTEXT	(#PCDATA)	#IMPLIED>
	< ! ATTLIST	TARGET	CONTENTSARGUMENTATION	(#PCDATA)	#IMPLIED>
	< ! ATTLIST	TARGET	CONTENTSCOVERAGE	(#PCDATA)	#IMPLIED>
	< ! ATTLIST	TARGET	LIFECYCLESPAN	(#PCDATA)	#IMPLIED>
	< ! ATTLIST	TARGET	LIFECYCLEOPERATION	(#PCDATA)	#IMPLIED>
	< ! ATTLIST	TARGET	PURPOSE	(#PCDATA)	#IMPLIED>
	< ! ATTLIST	IMG	SRC	(#PCDATA)	#REQUIRED>
	< ! ATTLIST	CHUNK	NAME	(#PCDATA)	#REQUIRED>
	< ! ATTLIST	CHUNK	TYPE	(FORMAL | INFORMAL)	#REQUIRED>
	< ! ATTLIST	CHUNK	INFORMAL_DESCRIPTION	(#PCDATA)	#REQUIRED>
	< ! ATTLIST	CHUNK	EXAMPLE	(#PCDATA)	#IMPLIED>
] >
```