Product Modeling for Requirements Engineering Process Modeling £

Jean-Roch SCHMITT

- Centre de Recherche en Informatique ; Université de Paris 1 Panthéon-Sorbonne
  17, rue de Tolbiac ; 75013 PARIS ; FRANCE
- GSI-Tecsi ; 176 Av. Charles de Gaulle ; 92200 NEUILLY SUR SEINE ; FRANCE
  e-mail : JRSchmitt@masi.ibp.fr

Abstract *: Current Information Systems (IS) development methodologies only sketch the definition of their development process, resulting in a low level of support to the developers.*

This paper first stresses the need to define new process models to underlie better the specifications development process. Second, it provides an overview of the solution proposed in the ESPRIT project NATURE , in the context of the Requirements Engineering phase.

Then, the emphasis is put on the need to precisely model the IS development product in order to support process modeling. A product model is thus detailed, argued and illustrated on examples.

Keywords Codes : D.2.1; D.2.6; D.2.10
Keywords : Software Engineering, Requirements/Specification; Programming Environments; Design

1. INTRODUCTION

Today's Information Systems (IS) developments suffer from the unbalanced description of (1) their modeling formalisms and (2) the way developers must follow to obtain the expected computerised system as it is prescribed by methodologies used in these developments [Rolland & Cauvet 1992].

The modeling formalisms provided by the methodologies define sets of concepts that allow the description of the system. They let designers reason about the system and validate its representation according to users' requirements.

The way-of-working of a methodology prescribes what kind of activities can be performed in the development and when they can be performed.

Up to now, a large amount of research effort has been dedicated to the modeling of Information Systems. This research has resulted in the development of a great number of sophisticated modeling formalisms adapted to the particularities of the system to be built and to the development environment.

Furthermore, Information Systems meta-models of have been developed that abstract from specific models and define generic concepts suitable to describe models. Such meta-models allow the comparison of several models and the adaptation of models according to the particular needs of an IS development.

£ This work was supported by ESPRIT project Basic Research Action 6353 (NATURE) which is concerned with Novel Approaches to Theories Underlying Requirements Engineering.

Contrarily, current methodologies hardly define the way-of-working designers can or must follow to proceed in the development : most of the process semantics is missing. This weakness can be explained by the little amount of research work that has been devoted to the modeling of the development process, relatively to the IS modeling effort.

This imbalance between models and way-of-working definition has also notable consequences on the CASE tools that more and more often support IS developments. The current CASE tools are efficient in recording, retrieving and manipulating the schemes representing the system undergoing development. But they are almost unable to actually provide assistance to the developers in deciding and performing the activities transforming these schemes.

We consider that this weakness in the prescription of methodologies ways-of-working stems from the inadequacy of the currently available process modeling approaches that support their description. We think that, in order to be suitable, new approaches should allow the ways-of-working description at various granularity levels and with a deeper semantic representation capability.

This need to model the IS development process motivates the "Process Theory" stream of the "NATURE" ESPRIT project. NATURE aims at investigating three complementary aspects of the Requirements Engineering (RE) phase of IS development : (1) knowledge representation, (2) domain knowledge definition and usage and (3) RE process modeling.
The objective of this last "Process Theory" stream is to propose an approach for the modeling of the development process adapted to the RE phase. This will allow detailed descriptions of methodologies ways-of-working and their integration in CASE tools.

The contribution of this paper is twofold. It first stresses that the particular process modeling approach proposed in NATURE requires the RE product to be precisely modeled. Secondly, it presents and exemplifies a product model that meets the requirements.

After a short presentation of the main arguments supporting the development of a new way of modeling the RE process, **section 2** gives an overview of the approach adopted in NATURE. In **section 3**, we stress why this process modeling approach requires a parallel modeling of the product and detail the model we propose. Finally, in the **section 4**, we shall show how this product modeling is used in the modeling of the process through the notion of "Situation" whose modeling will be sketched and illustrated through examples.

2. THE NATURE PROCESS MODELING APPROACH

2.1. Motivations for a new approach in development process modeling

As mentioned in the introduction, current IS development methodologies offer a great number of IS modeling formalisms but hardly detail the way-of-working developers must follow in order to deliver the computerised system starting from the initial users' requirements.
Indeed, most of the existing methodologies give only few insights concerning the way developers can make the system description evolve. Developers are most of the time provided with the definition of big phases and with constraints stating that a given phase cannot be undertaken before the previous one has been completed. These phases most often correspond to the various schemes to be developed. The developers are thus required to produce these

schemes at predetermined milestones, but they are not guided in the actual production of these schemes.

Moreover, although some methodologies recognise the need of iteration in design by allowing it among phases, they rarely prescribe the iteration conditions, resulting in an overall poor level of prescription.

This limited ability to specify the way-of-working developers must follow is still more hindering in the context of RE which is particularly characterised by :

- its highly creative, non-deterministic, interactive and cooperative nature,
- the alternative techniques that can be applied to proceed, at any stage : domain analysis, reverse engineering, reuse, etc.…
- its major importance over all the development life-cycle : the solutions adopted during the RE have consequences on all the succeeding phases,
- and its mixing of modeling formalisms, with various levels of formality.

To be more efficiently usable, a RE methodology should specify what kind of activities can be performed, what they modify, when and by whom they can be performed ; and this should be specified for any stage of the process. In other words, the developers would much more precisely know how to proceed during RE.

Furthermore, such a detailed knowledge could be introduced in CASE tools that would then be able to guide developers in their decision making process and in the resulting IS schemes transformations.

The prescription capability of a way-of-working specification highly depends on its underlying process modeling approach. According to [Dowson 1987], the existing process models, on which methodologies ways-of-working specifications are based, can be classified in three categories:

- activity-oriented models,
- product-oriented models,
- decision-oriented models.

Activity-oriented models come from an analogy with problem solving, i.e. finding and executing a plan of actions leading to the solution. They are sequential in nature and provide a frame for manual management of projects developed in a linear fashion. Such a linear view of the design process is inadequate for methodologies which support backtracking, reuse of previous designs and have to support parallel engineering activities. The first widely used model, the Waterfall model [Royce 1970], falls into this category, along with the Spiral model [Boehm 1988] and the Fountain model [Henderson-Sellers & Edwards 1990] which try to eliminate the well recognised lack of flexibility of the Waterfall model. Limitations and drawbacks of activity-centred development approaches come from their representation of development processes like programs which do not integrate at all the interactive aspects of IS development.

The product-oriented process models represent the development process through the evolution of the product. They promote a view of development processes which is still centred around the notion of development activity but present the advantage to link development activities to their output : the product. The ViewPoints model [Finkelstein & al. 1990] belongs to this category as well as the development process model proposed in the European Software Factory (ESF) project [Peugeot and Franckson 1991].

A more recent class of process models follows a *decision-oriented paradigm*. The successive transformations of the product are looked upon as consequences of decisions. The process model of the DAIDA project [Jarke & al. 1990; Rose & al. 1991] and of [Potts 1989] fall into this category. Such models are semantically more powerful than the previous ones because they explain not only how the process proceeds but also why transformations happen.

They provide a more complete knowledge about the process which is particularly useful for reuse of developed products, or of design decisions, and for backtracking purposes when a modification in requirements has to be considered either during the design process itself or during the maintenance of the IS.

2.2. Overview of the NATURE process modeling approach

Considering the characteristics of the RE mentioned above, it appears that only the decision-oriented approach, with its transformation justification capability, is partially appropriate to the NATURE process modeling goals.

In this context, a *decision* is a choice the requirements engineer makes and it induces a transformation of the product.

However, the decision-oriented approach offers only limited knowledge about when and how to decide on what.

To solve this limitation, we enhance the decision-oriented approach by introducing the notion of *context*. This enhancement stems from the recognition that analysts react *contextually* according to the domain knowledge they acquire and react by analogy with previous situations they have encountered.

A *context* is defined as the association of a decision with some the parts of the IS being developed, called a *situation*, on which the decision is made.

Based on this intimate grouping of a decision with the situation on which it is made, called "context", the NATURE process modeling approach allows a better representation of RE processes. These processes are thus considered not only as collection of transformation activities but explicitly as ordered sets of decisions made on situations, recognised in the IS specification.

This approach is an extension of the one presented in [Grosz & Rolland 1990] and implemented in the "OICSI" prototype CASE tool [Rolland & Cauvet 1991].

The NATURE process meta-model, whose main concepts are the ones of situation, decision, context, action and argument is very simply overviewed in the figure 1 :

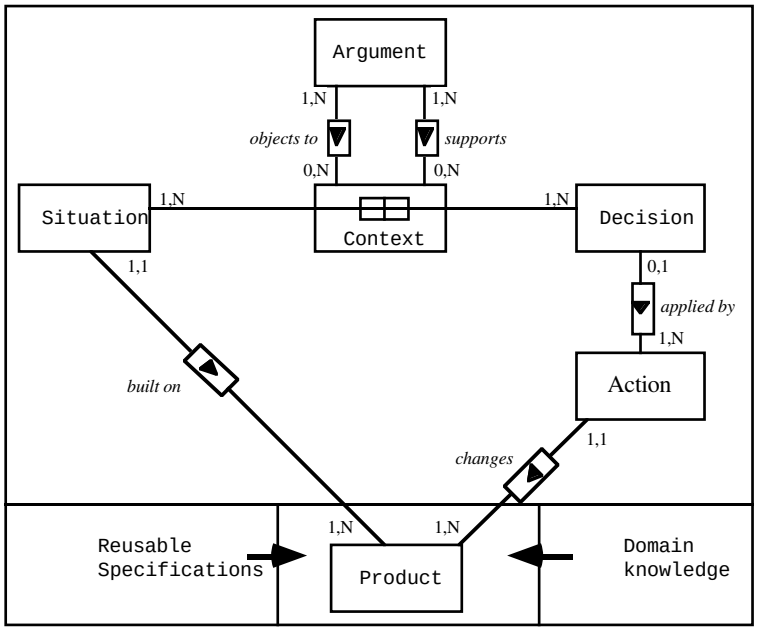

Figure 1 : Simplified overview of the NATURE Process Model

In this figure, an ER-like notation is adopted to highlight the main concepts (rectangles) and their inter-relationships (boxed arrows). "context" is both the situation-decision association and a concept. The bottom of the figure illustrates the integration, in the "product" concept, of previously developed reusable specifications and of domain knowledge.

Note that this figure also highlights the basic loop of the process modeling approach : situations motivate decisions that are applied by actions that modify the product in which situations are recognised.

A *situation* is most often a part of the product it makes sense to make a decision on. It represents the context of a decision. It can be a statement about the application domain or a requirement. It can also be an historical knowledge about previous developments or about generic domains.

A *decision* reflects a choice that a requirements engineer makes at a given point in time of the RE process. A decision can be looked upon as the intention of product transformations.

A *context* is the association of a situation and a decision made on it.

An *action* performs a transformation on the IS description. It is a materialisation of the decision. Performing an action changes the description and may generate new situations, subjects to new decisions.

Arguments are statements supporting or objecting to decisions made in a given situation, i.e. support or object to contexts.

[Rolland 1993] details the NATURE process meta-model and the reader is invited to refer to it for a complete presentation of these concepts.

With such a process model, the description of a way-of-working is made by specifying what kinds of decisions can be made in what kinds of situations and what kinds of actions have to be performed to apply the decisions.

The figure 1 also illustrates the link existing between the RE product development process and the product itself : the product is the source of the situations recognition and is modified by the performance of actions.

The main consequence of this intimate relationship is that a detailed process model cannot be obtained without a parallel *product model.*

In the remaining of this paper, we present the product model and illustrate its use in the building of situations; the relationship between the process model and the product model, through the action, will not be presented here.

Let first make clear the abstraction level we focus on.

One of the requirements of the NATURE process modeling approach is that it must be defined independently from any specific methodology. The result of the work must then be a model (an abstraction), of process models : a process meta-model.

Like any meta-model, this process meta-model is a set of generic concepts suitable to represent the process model of any given RE methodology.

In NATURE, the Telos [Mylopoulos et al. 1990] knowledge representation language allows a formal expression of process models and of the process meta-model.

3. PRODUCT MODELING

The purpose of this section is to present the RE "product" modeling we propose in NATURE to fit the needs discussed above. In this section, we successively give an intuitive definition of the concept of "product" and detail its modeling, illustrated on examples.

3.1. Intuitive definition of the concept of product

Let's first define the RE "product" as the IS undergoing development during the Requirements Engineering phase.

Traditional IS development methodologies define their product like they define their process : they decompose it into sub-products as they decompose broad phases into sub-phases. Each sub-phase of the development process being defined relatively to its output : a sub-product.

These methodologies often decompose the product into conceptual, logical and physical sub-products. The conceptual sub-product is for example decomposed into structural and functional schemes.

The very nature of the IS development product is however much more complex, particularly in the context of Requirements Engineering. The RE product has some important characteristics that must be taken into account for its precise modeling.

Firstly, let's now define the "Product" as all the knowledge the different stakeholders of the RE phase have about the system to be built. At the beginning of the RE phase, the product may be a collection of users' requirements, whatever their expression media : natural language sentences, forms, drawings, etc.… This initial understanding of the system is usually controversial, opaque and informal [NATURE 1992]. At the end of the RE phase, the product must contain the conceptual specification of the system.

In between these two stages, the product is actually a mix of initial requirements, intermediate schemes and partial conceptual schemes. Moreover, each product component may be characterised by its own specification, agreement and formality levels, according to the framework proposed in [NATURE 1992].

Secondly, the schemes that compose the product are sets of instances of modeling formalisms concepts, such as the E/R or data-flow concepts. But the use of a given formalism is not unique. For example, let's consider two methodologies using the data-flow formalism to describe the functional perspective of a system. The first one may define "flat" data-flow schemes as being composed of processes, flows, stores, etc.… . The second methodology may define data-flow schemes as being composed of data-flow diagrams, themselves composed of processes, flows, etc. , i.e. it introduces an hierarchical building of data-flow-schemes. This example shows that the product is built from views of modeling formalisms that is relative to the methodology.

Thirdly, the product may also carry knowledge concerning the state of the process. For example, a data-flow schema may be in a "verified" state. The RE process then making these sub-products evolve from states to states. Such a piece of information is of course not defined by the data-flow formalism and is thus relative to the methodology.

The last two points motivate the clear distinction we make between the modeling formalisms used by a methodology and the product model defined by the methodology.

Section 2 has introduced the intimate relationship existing between the modelings of the *process* and the *product* as proposed in NATURE. The previous discussion has shown that the modeling of the product is based on the *modeling formalisms* provided by development methodologies. Figure 2 illustrates the relationships between these three kinds of models.

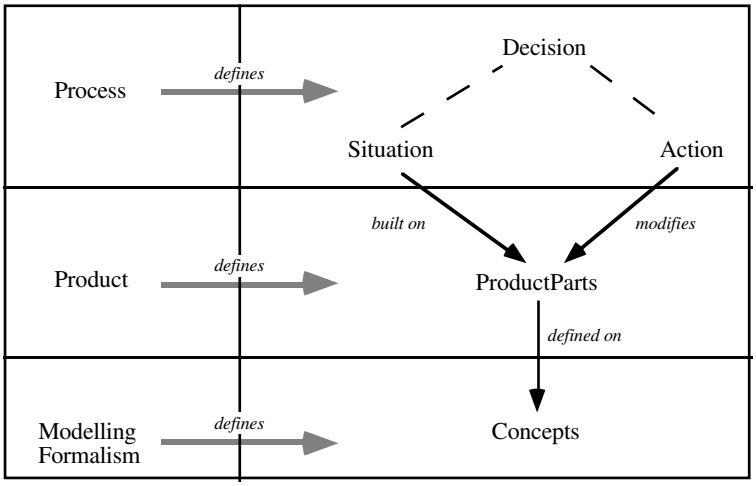

Figure 2 : Modeling formalisms, product and process models relationships.

Finally, IS developments more and more often tend to reuse knowledge acquired during previous developments. This existing knowledge can be made, for examples, of conceptual specifications whose elements can be integrated in the new specification or of DB-programs code which can be reverse-engineered. It can also be made of generic domain knowledge pieces that can be specialised and adapted to the system undergoing development.

As the activities of reusing this existing knowledge are actual activities of the RE process, the corresponding decisions are made on situations built both on parts of the product undergoing development and on parts of existing products. This brings the need to extend the notion of product with existing knowledge. The complete RE product is then made of two parts : the existing products and the product under development.

To summarise this discussion, we recognise the following characteristics of the definition of the product of IS development methodologies :

- its building :
  * from which models, of which formality level
  * by which decomposition hierarchy
- the allowed advancement states of its parts
- the kind of existing products that can be reused to build it

3.2. Product model

According to section 2, the product model we present here is in fact a product meta-model, i.e. the definition of generic concepts allowing the description of the types of products defined by RE methodologies. However, for the sake of clarity, we will describe this meta-model using terms belonging to the process abstraction level instead of the process model level. For example, instead of saying that instances of the "ProductElement" concept of the meta-model are ProductElement-types whose instances are elements of the schemes composing the product, we will say that instances of the î"ProductElement" concept are elements of the schemes composing the product.

Figure 3 presents, using a E/R-like notation, the product model centred around the notion of "Product Component". In the sequel, we present each of the concepts of this model in turn.

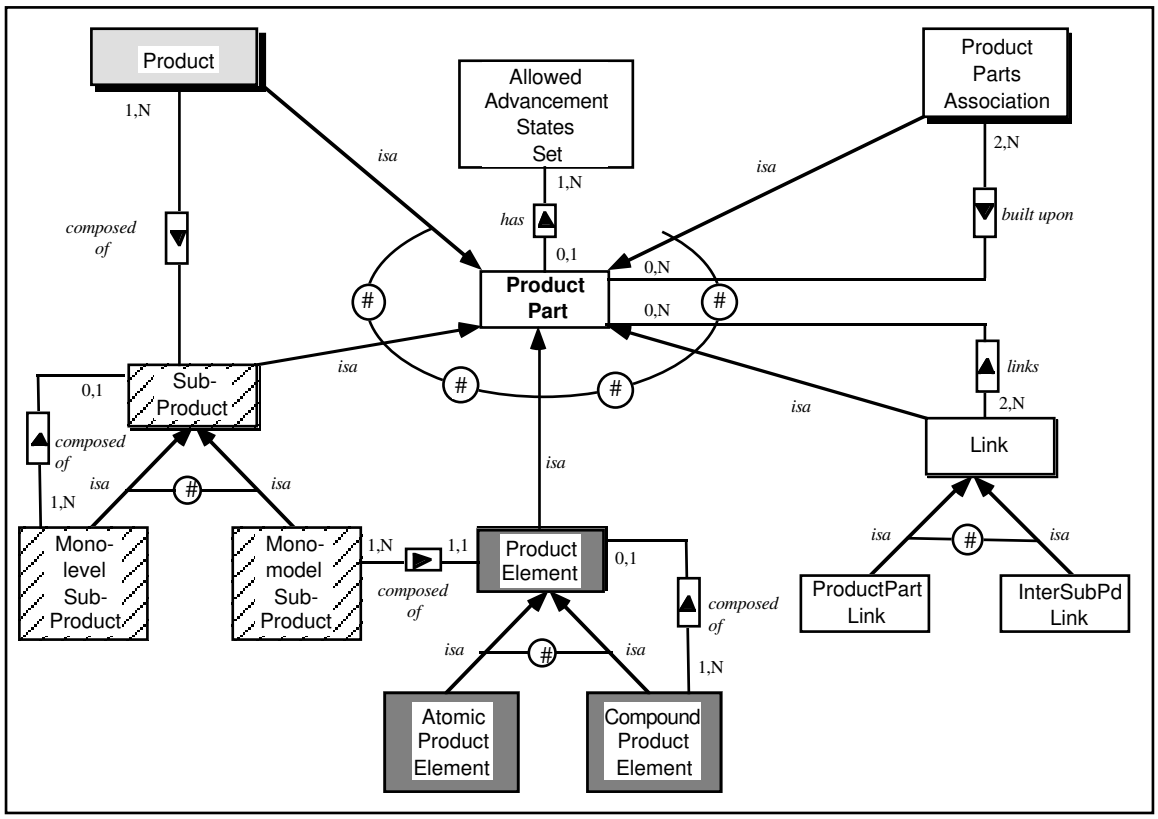

Figure 3 : Product model

The following RE-product example, depicted in figure 4, is a partial instanciation of the product model (figure 3) and will be used throughout this section to illustrate its concepts :

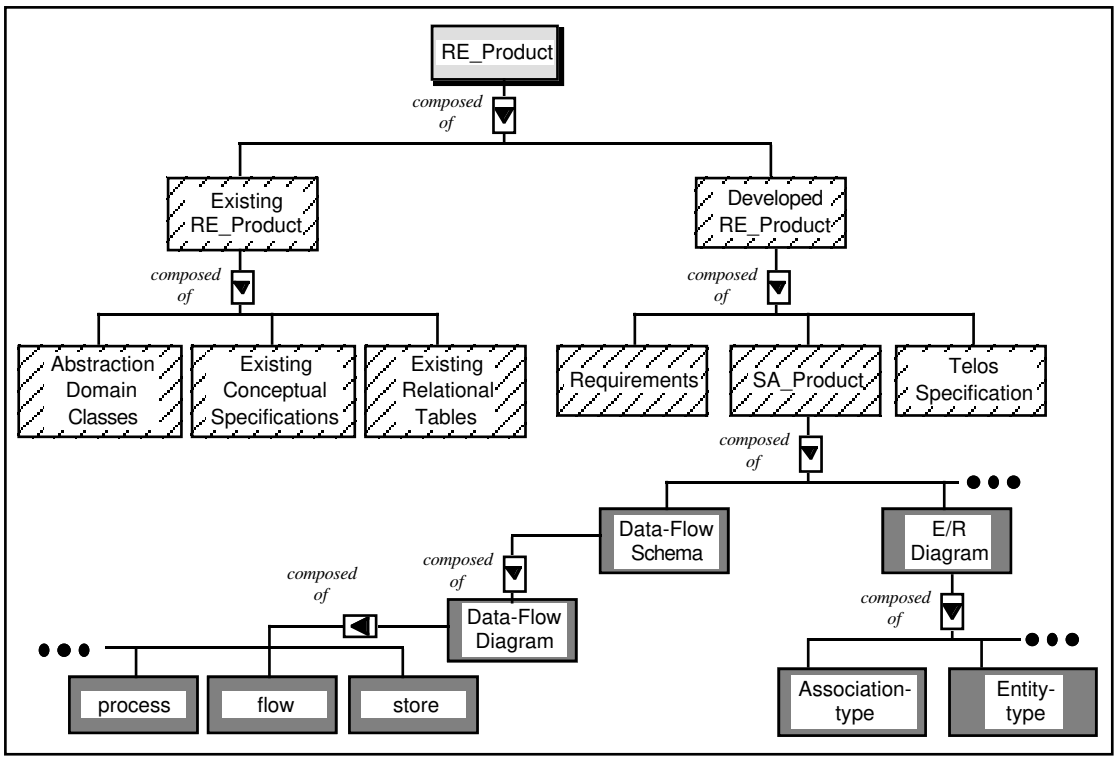

Figure 4 : Instanciation example of the product model

The central concept of the product model is the "ProductPart". It represents any set of elements carrying knowledge about the product, be it the developed product or the existing product. These sets of product elements can be either the product itself, sub-products, instances of modeling formalisms concepts, links or product parts associations. The "ProductPart" concept is thus specialised in the concepts "Product", "SubProduct", "ProductElement", "Link" and "ProductPartsAssociation".

In the sequel, we detail each of those product model parts in turn.

IS development methodologies - ways-of-working - are usually divided into phases, such as the "conceptual modeling phase" or the "formalisation phase". Such phases yield in parts of the product, referred to as "SubProducts" in figure 3. These phases may themselves be sub-divided into sub-phases, resulting in an hierarchical composition of sub-products. As each phase of a methodology results in a partial specification of the system at a given level of abstraction, each sub-product belongs to only one abstraction level.

Furthermore, a sub-product may either use one or several of the modeling formalisms provided by the methodology. In figure 3, this distinction is represented by the "MonoLevelSubProduct" and "MonoModelSubProduct" subtypes of "SubProduct". "Mono-ModelSubProduct" represents the sub-products that are made of concepts of a same modeling formalism (i.e. what is traditionally called "schema"). "MonoLevelSubProduct" represents sub-products that are composed of sub-products and belong to the same abstraction level.

In the example (figure 4), the RE-product is made of, on one side, an existing product composed of conceptual specifications, relational tables and domain abstraction classes [NATURE 1992]. Elements of the conceptual specifications may be reused if satisfactory similarities are found with elements of the system to be developed. Relational tables may be reverse-engineered into reusable conceptual specifications. Domain abstraction classes may be matched against elements of the developed product and then be adapted.

The developed product, on the other side, is composed of a set of requirements, a Structured-Analysis-oriented modeling and a formal Telos specification. These sub-products are at the "conceptual" level and are respectively "informal", "semi-formal" and "formal".

The "SA_Product" is composed of the sub-products "ER-diagram" and "Data-Flow-schema", to cite only two of them. Each of these component sub-products uses one formalism, they are thus instances of the "MonoModelSubProduct" concept. Since its different components do not use the same formalism, the "SA_Product" is contrarily an instance of the "MonoLevelSubProduct" concept.

The dark grey boxes shown on figure 3 represent the building of schemes from the concepts provided by the modeling formalisms. We are now going to detail this part of the product model.

According to [Rolland & Souveyet 1990], any modeling formalism provided by IS development methods defines a set of concepts and constraints suitable to represent information systems. For example, extended E/R formalisms define the concepts "entity-type" and "association-type" and the disjunction constraint between two sub-types of an entity-type.

These concepts and constraints serve as a basis for the definition of the product elements that compose the schemes representing an IS.

We saw that a methodology defines its product as a decomposition into sub-products. Complementarily, it also specifies the decomposition hierarchy of these sub-products into product elements based on the concepts and constraints of the formalisms.

The concept "ProductElement" (figure 3) represents the components of a "MonoModelSubProduct". For examples, an E/R diagram is composed of entity-types and association-types, a data-flow schema may be composed of data-flow diagrams, themselves composed of processes, flows, stores, etc. . These two examples illustrate that there are two

kinds of product elements : atomic and compound. Instances of the "AtomicProductElement" concept cannot be decomposed into other product elements and are directly defined on the concepts and constraints of their formalism. Instances of the "CompoundProductElement" concept, in contrary, are composed of other product elements.

Up to now, we have seen how the product model allows the specification of the hierarchical decomposition of the product into sub-products which are themselves decomposed until undecomposable product elements are reached. These "leaves" product elements are then defined on the concepts and constraints defined by the modeling formalisms provided by the method.

It is important to remember that given a common set of modeling formalisms, two different methodologies may specify two different decomposition hierarchies, thus defining two different products.

Let's now consider the "Link" sub-type of the "ProductPart" concept in figure 3.

This concept represents the knowledge the developers have about the relationships existing between parts of the product. A link relates two parts of the product, be them sub-products, product elements, product parts association or even links.

For example, it may be necessary to record in the product that an entity-type of an E/R diagram corresponds to a relational table of the existing product. It may also be necessary to record the similarity value [NATURE 1992] existing between two data-flows diagrams, or to record that a given store in a data-flow diagram corresponds to a given entity-type.

We recognise three sub-types of the "Link" concept, according to the semantics of the link. The first type of links is the *"MappingLink"* type which represents the links that relates an association of product parts to the one it has been mapped into. Note that the two product parts associations must then be at different abstraction levels. The previous example concerning the relational table / entity-type link falls in this category. The example of the store / entity-type link is of the second type of link : the *"CrossRefLink"* which represents correspondence links within a given level of abstraction. These two types of links are represented by the "InterSubPdLink" concept that generally handles the modeling of the links defined between modeling formalisms. The last type of links, the *"SpecificLink"* concept, is the one of the links which do not fall in the two previous categories and that depend on methodologies. Similarity links for example fall in this category. This category is represented in figure 3 by the "ProductPartLink" concept.

Finally, the last kind of product part that can carry knowledge about the system is a grouping of product parts, represented by the concept "ProductPartsAssociation". An instance of this concept is a set of product parts, whatever their type is, except the product itself : sub-products, product elements, links or even product parts associations.

Actually, this concept is necessary to represent any group of product elements carrying knowledge about the system that is neither a node of the product decomposition hierarchy nor a link.

For example, the following figure shows two product parts associations between which a similarity link has been established by a developer. :

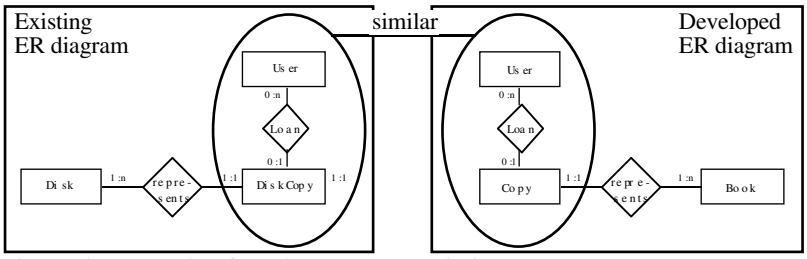

Figure 5 : Example of product parts associations

According to section 3.1, the definition of a methodology's product must contain the definition of the possible states of advancement of each part of the product and must specify the allowed transitions between states.

IS development methodologies often define states for the product parts, states that correspond to the achievement of sub-phases. The OPML model [Peugeot and Franckson 1991], for example, allows detailed representations of object-states graphs. Here, such graphs are referred to as "evolution graphs".

Furthermore, [NATURE 1992] introduces a three dimensional framework in which the RE product can be located, at any stage of the RE. The three axis represent the specification (degree of understanding) , representation (formality level) and agreement (stakeholders' view) dimensions. The specification and agreement dimensions are good extensions of the classical definition of the state of product parts while the formality level is defined as a characteristic of the modeling formalism and does not need to be modeled in the product model. The "AllowedAdvancementStatesSet" concept (figure 3) is introduced in the model to handle the representation of this kind of information.

4. SITUATION BUILDING UPON THE PRODUCT

The purpose of this section is to show how the concepts of the product model presented in the previous section can serve as basic building blocks of the situations developers may make decisions on.

As it is beyond the scope of this paper to detail the whole process model of NATURE, we only sketch the modeling of situations and we shall give examples of situations building and decisions that may be made on them.

4.1. Definition of the notion of situation

According to the presentation of the process model proposed in NATURE as described in section 2, the concept of "Situation" is a key notion in the contextual approach. Requirements engineers react to situations recognised in the product by making decisions on them. Those decisions which are in turn applied by actions performed on the product.

As a first definition of the notion of situation, let's define it as the minimal set of product parts a given decision can be associated to, e.g. on which it makes sense to make this decision on. This definition says firstly that the existence of a situation is motivated by the existence of a decision. Secondly, the definition states that two situations associated to a same decision should not have an inclusion relationship between them (minimality).

It is important to note that although situations are built from products parts similarly to the building of compound product parts from smaller ones, the two building semantics are different.

On the one hand, product parts are defined because they carry knowledge about the system undergoing development.

On the other hand, situations are defined and built from product parts because they represent products parts sets it makes sense to make decisions on.

4.2. Situation modeling

Before presenting the modeling of situations, let's discuss its requirements. First, we need to give a means for representing all the combinations of product parts on which the decisions of a methodology can be made. Secondly, conditions may constraint product parts sets in order to be acceptable situations.

For example, the way-of-working of a given methodology may specify that the decision to verify an E/R diagram cannot be made if the diagram is not in the "frozen" state. Similarly, it

may be specified that the decision to create a disjunction constraint between two subtypes of an entity-type should not be made if the constraint already exists.

Thus, the modeling of situations must allow the representation of the building of situations upon the product parts and the expression of conditions constraining this building.
This modeling is illustrated by the figure 6.

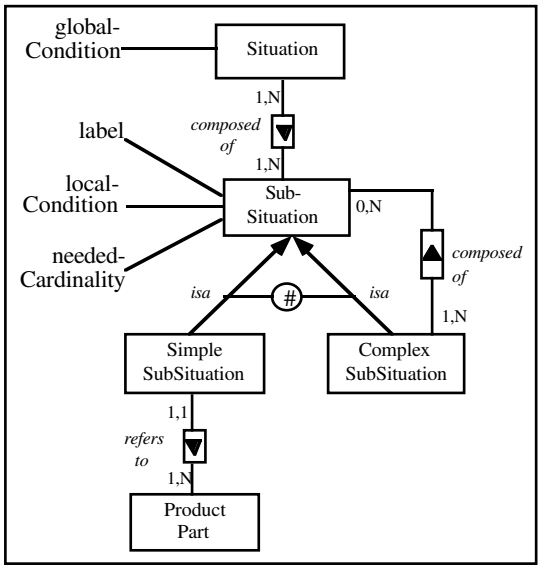

Figure 6 : Situation modeling

This modeling of situations is centred around the concept "SubSituation" which represents sets of product parts of the same type and satisfying the same condition. A situation is composed of a set of sub-situations.
The attributes of "SubSituation" are its needed cardinality, a local condition to be satisfied by product parts to belong to this sub-situation and a label allowing to reference its elements in the expression of the conditions. The needed cardinality may be exactly known or not ; if it is exactly known, it may be any integer - including "0" - , if it is not exactly known, it may be either "some" or "all". A value of "0" means that no corresponding product part satisfying the local condition should be in the situation. An integer value other than "0" means that the situation contains exactly this number of product parts of this type and satisfying the local condition. The values "some" and "all" respectively mean that some or all of the product parts, of the given type, can be in the situation, if they satisfy the local condition.
Sub-typing sub-situations into simple and complex one allows a hierarchical building of situations. A complex sub-situation is composed of sub-situations. A simple sub-situation refers to the type of product parts is defined on.
A global condition is also defined to represent the conditions constraining the situation and that are not related to its elements, or to represent the conditions existing between elements of the situation. For example, the User-Loan-Copy situation in figure 5 may be constrained by a condition stating that the ER diagram should be in a given state for the decision to be made : this condition does not bear on elements of this situation.

4.3. Situation examples

In order to illustrate the use of this situation modeling, we are now presenting some examples of situations, together with the decision that justify their definition. Remember that the association situation-decision partly represents the conditions under which a decision can be made.

Assume that the methodology at hand only specifies its way-of-working by defining phases and by stating conditions constraining the succession of these phases. This can be represented by defining a decision for each phase, by defining the product parts corresponding to the phases and by defining states for these products parts. These states are then used in the definition of the situations in which the decisions can be made.

For example, the decision "MakeDesign" is associated to the situation made of the "AnalysisSubProduct" which is constrained by the "state is 'analysis completed' " condition (mark 1 in figure 7). The definition of the corresponding situation is the following :

```
Decision : MakeDesign
Situation :
        globalCondition = none
        SubSituation1 :
                label = none
                ReferredProductPart = "AnalysisSubProduct"          (1)
                localCondition = "State is 'AnalysisCompleted' "
                NeededCardinality = 1
```

Figure 7 : Coarse grained situation example

If the methodology would provide a much finer level of granularity, it may specify that whenever two entity-types are sub-types of a same entity-type a decision may be made that creates a disjunction constraint between the sub-types, if this constraint does not already exists (mark 1 in figure 8). The corresponding situation is defined as follows :

```
Decision : CreateDisjuctionConstraint
Situation :
globalCondition =    "superType = IsA.x.target"
                              "constraint.left = IsA.1" & "constraint.right = IsA.2"
        SubSituation1 :
                label = "superType"
                ReferredProductPart = "EntityType"
                localCondition = none
                NeededCardinality = 1
        SubSituation2 :
                label = "subTypesGroup"
                NeededCardinality = 2
                localCondition "IsA_link.source = 'subType.x' "
                SubSituation2.1
                        label = "subType.x"
                        NeededCardinality =1
                        localCondition = none
                        ReferredProductPart = "EntityType"
                SubSituation2.2
                        label = "IsA.x"
                        NeededCardinality =1
                        localCondition = none
                        ReferredProductPart = "IsA_link"
        SubSituation3 :
                label = "constraint"
                NeededCardinality = 0                       (1)
                localCondition = none
                ReferredProductPart = "DisjunctionConstraint"
```

Figure 8 : Fine grained situation example

These examples of situation, although quite simple, illustrate how the situation-decision association of the notions of allows the representation of the prescriptions of a methodology, at any level of granularity : a very broad one as in the first example and a very fine one in the second.

5. CONCLUSION

In this paper, we proposed a model of the Requirements Engineering (RE) product.

We first argued that the low level of detail provided in the specification of their way-of-working by the classical IS development methodologies is a real handicap, particularly for the RE phase and in the context of CASE tools supported development environments.

We then showed how the RE process modeling approach proposed in the NATURE project can solve this limitation.

Thereafter, we explained that the particular NATURE process model requires the RE product to be modelled in details,

The main part of the paper was then a presentation of our product model.

Although it was not discussed in the paper, we want to pinpoint here that our RE product model can be easily extended to allow the representation of IS development products over all the life-cycle, from RE to maintenance.

In order to operationally prove the validity of the product model by developing the NATURE RE CASE tool prototype, we are currently undertaking its formal description in the Telos language.

Acknowledgement

The author thanks the NATURE project partners for their feedback and is particularly grateful to Georges Grosz and Carine Souveyet for their fruitful advice.

REFERENCES

[Boehm 1988] Boehm B.W.: A spiral model of software development and enhancement; IEEE Computer 21.

[Dowson 1987] Dowson M.: Iteration in the software process. Proc. 9th Intl Conf Software Engineering, Monterey, Ca.

[Finkelstein et al. 1990] : Finkelstein A., Kramer J., Goedicke M.; "ViewPoint Oriented Software Development"; Proc. Conf "Le Génie Logiciel et ses Applications", Toulouse, p 337-351, 1990.

[Grosz & Rolland 1990]: G. Grosz, C. Rolland; Using artificial intelligence techniques to formalize the information system design process; Proc. Intl. Conf. Databases and Expert Systems Applications, 374-380.

[Henderson-Sellers and Edwards 1990] : Henderson-Sellers B., Edwards J.M.; "The Object-oriented Systems Life-Cycle"; Comm. ACM, 09/1990.

[Jarke et al. 1992] : Jarke M., Mylopoulos J., Schmidt J.W., Vassiliou Y.; “DAIDA - An Environment for Evolving Information Systems”; ACM Transcactions on Information Systems, Vol. 10, No. 1, 1992.

[Mylopoulos et al. 1990] Mylopoulos J., Borgida A., Jarke M. and Koubarakis M.: Telos:Representing Knowledge About Information Systems. ACM Trans. Information Systems 8, 4, 325-362.

[NATURE 1992] : Jarke M., Sutcliffe A., Vassiliou Y., Rolland C., Bubenko J.; NATURE Esprit III project n° 6353, Deliverable NATURE.D-I, 12/1992.

[Peugeot and Franckson 1991] : Peugeot C., Franckson M.; "Specification of the Object and Process Modeling Language", ESF Report n° D122-OPML-1.0, 1991.

[Potts 1989] : Potts C.; “A Generic Model for Representing Design Methods”; Proceed. 11th International Conference on Software Engineering, 1989.

[Rolland & Cauvet 1991] : Rolland, C., Cauvet, C. : An Expert System for Requirements Engineering, in "Advanced Information Systems Engineering"; R. Andersen, J.A. Bubenko, A. Solvberg (Eds), Springer Verlag 1991.

[Rolland & Souveyet 1990] : C. Rolland, C. Souveyet;" An Object Oriented Framework for Information Systems"; Proc of the Intern. Conf. on Management of Data (COMAD'90), New Delhi, India, 12-14 déc. 1990, pp 280-303.

[Rolland & Cauvet 1992] : Rolland C., Cauvet C.; "Trends and Perspectives in Conceptual Modeling"; in "Conceptual Modeling, Databases, and CASE, An integrated View of Information Systems Development", Loucopoulos P. & Zicari R. Eds, WILEY, 1992, pp 27-48.

[Rolland 1993] : Rolland C. ; "Modeling the Requirements Engineering Process" ; 3rd European-Japanese Seminar on Information Modeling and Knowledge Bases, Budapest, Hungary, 1993, to be published.

[Rose et al. 1991] Rose T., Jarke M., Gocek M., Maltzahn C., Nissen H.: A decision-based configuration process environment. Software Engineering Journal 6, 5.

[Royce1970] : Royce W.W.; "Managing the development of large Software Systems"; Proc. IEEE WESCON 08/1970.