Modelling Ways-of-Working

Véronique Plihon, Colette Rolland
University of Paris I Panthéon-Sorbonne
C.R.I., 17 rue de Tolbiac
75013 Paris FRANCE
email : {vplihon , rolland}@masi.ibp.fr

Abstract. We propose in this paper, an approach for defining in a systematic manner, ways-of-working providing guidelines for the development of information systems. It is a modelling approach in which a given way-of-working is constructed by instantiation of a way-of-working model allowing to deal with a large variety of situations in a flexible, decision-oriented manner. The paper presents the way-of-working model and exemplifies the construction of a specific way-of-working based on the OMT methodology.

1 Introduction

Despite thirty years of constant improvements of Information Systems (IS) methodologies, the IS community is not satisfied with existing ways-of-working. They are too large-grained descriptions of the system life-cycle, unclearly defined, do not guide application engineers efficiently and consequently are not used as intended. They lack flexibility and cannot be easily adapted to the requirements of a specific project. One approach to solve the problem is to deal with methods as we deal with information systems, i.e. to engineer a method as a product subject to design, customisation, adaptation and evolution. This is the aim of *Method engineering*. Method engineering is defined in [8] as the engineering discipline to design, construct and adapt Information Systems Development Methods and CASE tools to specific projects.

Within the area of Method Engineering (ME) the formal description of a method takes an important place. Many definitions of a method have been proposed [2], [12], [17], [10] and most of them converge to the idea that a method is based on models (systems of concepts) and consists of a number of steps which must/should be executed in a given order. For example [15] proposes a framework for information systems development *methods* which comprises the *way of thinking* (the philosophy), the *way of modelling* (the models to be constructed), the *way of organising* (subdivided in the *way of working*, i.e. how to perform the development and the *way of control* i.e. how to manage the development) and the *way of supporting* (the description techniques and the corresponding tools).

In this paper we are concerned with the formal description of ways-of-working in the sense of the above framework. Our proposal consists of a *way-of-working model,* i.e. a set of generic concepts allowing to construct various ways-of-working for various methods in a structured and detailed manner. The way-of-working model plays the role of a theory and functions like a shell, a model from which specific ways-of-working can be generated by instantiation and then executed to guide the development of various projects.

Ways-of-working we want to generate are *guidance centred process models*. There is a need for such process models, in particular, in the early phase of Requirements Engineering (RE) and conceptual modelling. While performing highly intellectual and creative activities required to construct the system specification, application engineers need to be guided and advised, locally to handle the particular situations they are faced to, and globally, to monitor the flow of decisions they have to make. The required support is mainly based on heuristic knowledge. For this reason, our approach differs from the Software Engineering (SE) process modelling view which is focusing on process centred software environments (see a survey in [1]) for enforcing process performance that conforms to some preceptive software process definition [4].

The proposed way-of-working model corresponds to a certain way-of-thinking, a philosophy for process guidance which is decision and situation based. In other words, the underlying assumption of our way-of-working model is that the Information Systems Development (ISD) process is mainly a decision making process during which application engineers have to make decisions adapted to the situations they are faced to. Consequently the way-of-working intends to capture the types of situations the developer could encounter during ISD and to provide guidance, in each situation-type, on the decisions to make.

The way-of-working model focuses on the Requirements Engineering phase of systems development. It has been developed within the Process Theory of the NATURE[1] project and is being implemented as part of the CAPE (Computer Aided Process Engineering) environment to support process model definition, process guidance, trace and improvement.

For the sake of clarity, informal examples are used to illustrate the way-of-working structure; nevertheless a formal description of this process model, based on a decisions algebra can be found in [14].

The paper is organised as follows. The core of the paper deals with the presentation of the way-of-working model and its illustration to describe the OMT [13] method. This is made in section 2. Section 3 is a discussion of the approach. Conclusions are drawn in section 4.

2 The Way-of-Working Model

2.1 An Overview

The model we propose to support the construction of ways-of-working is centred around the concept of *context* which constitutes the basic building block of our approach. Contexts can be linked repeatedly in a hierarchical manner to define trees. A way-of-working is therefore a forest of trees as it is shown in the overview of the way-of-working model presented in the figure 1.

[1] This work is partly supported by the Basic Research Action NATURE (ESPRIT N°5363) which stands for Novel Approaches to Theories Underlying Requirements Engineering.

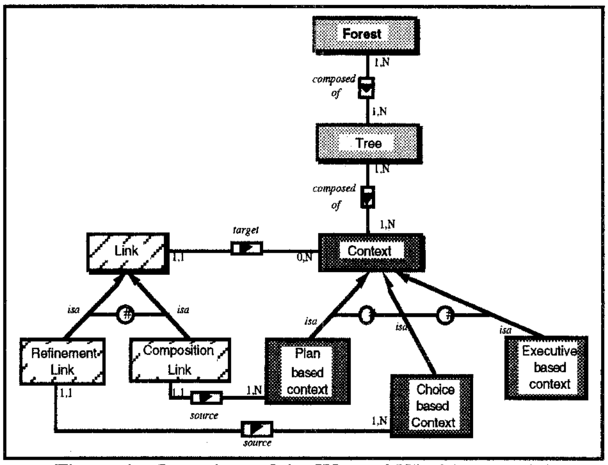


Figure 1 : Overview of the Way-of-Working Model

A *way-of-working* is represented as a collection of hierarchies of contexts that we refer to as a *forest.* A hierarchy of contexts, called a *tree*, represents a structured piece of knowledge for supporting decision making in the RE process; it is a process chunk which aims at supporting the application engineers in making the most appropriate decision according to the situation at hand. A tree is composed of *contexts* and *links*. Links allow to relate contexts in a recursive manner to construct hierarchies of contexts. As shown in figure 1, links are of two kinds : *refinement links* which allow to refine a large-grained context into a finer ones and *composition links* allowing to decompose a context into component contexts.

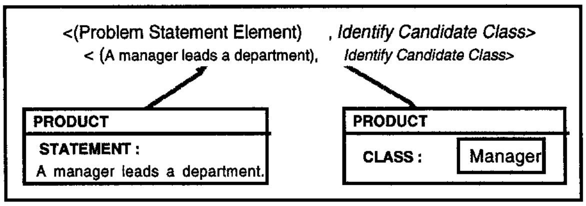
We detail in turn the notions of context, tree and forest and exemplify them with the OMT methodology.

2.2 The Notion of Context

A context is defined as a couple (situation, decision), where the *decision* represents a choice that an Application Engineer (AE) can make at a given point in time of the development process and where the *situation* is defined as the part of the product it make sense to make a decision on. A *decision* corresponds to an intention, a goal that the application engineer wants to fulfil while the situation is a part of the product under development the AE is working on. A context has also a type, namely executive, choice or plan. As we will see in the following, each type of context plays a specific role in a tree.

2.2.1 Executive-based Context

An executive-based context corresponds to a decision which is directly applicable through an action inducing a transformation of the product under development. For instance, in the OMT methodology, the creation of a candidate class is considered as an *action,* it materialises the implementation of the *executive-based context* <(Problem Statement Element), *Identify Candidate Class*>. Figure 2 is an instantiation of this *executive-based context* (<(A manager leads a department), *Identify Candidate Class*>) showing the consequence of its execution on the product (the creation of the class Manager).



<(Problem Statement Element) , *Identify Candidate Class*>
< (A manager leads a department), *Identify Candidate Class*>
PRODUCT
STATEMENT :
A manager leads a department.
PRODUCT
CLASS :
Manager

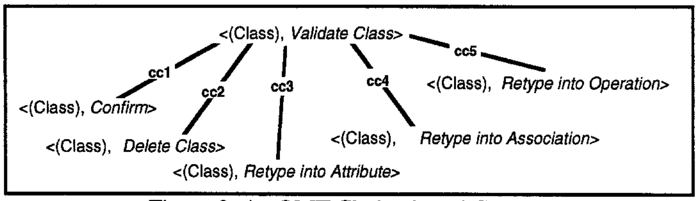
Figure 2 : An OMT Executive-based Context

Executive-based contexts are the atomic blocks of our ways-of-working. Non atomic blocks are built over contexts using refinement and composition links. Let us present first, choice-based contexts which are built with a refinement structure, explain their semantic and features.

2.2.2 Choice-based Context

When building a product, an application engineer may have several alternative ways to modify the product. Therefore, he/she has to select the most appropriate one among the set of possible choices. In order to model such a piece of knowledge, we introduce a second type of context, namely the *choice-based context.* The execution of such a context consists of choosing one of its alternatives, i.e. selecting a context representing a particular strategy for the resolution of the issue raised by the context.

For instance in the OMT methodology (figure 3), there are several ways to validate a class: by confirming, deleting or retyping the class into an attribute, an association or an operation.



<(Class), *Validate Class*>
cc1
cc2
cc3
cc4
cc5
<(Class), *Confirm*>
<(Class), *Delete Class*>
<(Class), *Retype into Attribute*>
<(Class), *Retype into Association*>
<(Class), *Retype into Operation*>

Figure 3: An OMT Choice-based Context

In our way-of-working model, arguments and choice criteria are defined to support or object to the various alternatives of a choice-based context. They are very helpful to support the application engineer while choosing the most appropriate strategy.
Table 1 gives a description of the arguments and choice criteria associated to the example of choice-based context introduced in figure 3.

arguments :
a1: The candidate class explicitly refers to a real world phenomenon relevant to the system.
a2: The candidate class expresses the same information than an existing object class and its name is less descriptive than the other.
a3: The candidate class is not in the scope of the problem.
a4: The candidate class is ill defined and not precise according to the scope of the problem.

a5: The candidate class expresses an implementation construct.
a6: The name of the candidate class describes an individual object, the class has no specific behaviour and represents an aspect or a property of an existing class.
a7: The candidate class represents something which is applied to the object.
a8: The candidate class expresses a role that is played in an association.
choice criteria :
ccl = a1 **cc2** = a2 ∨ a3 ∨ a4 ∨ a5 **cc3** = a6 **cc4** = a7 **cc5** = a8

Table 1: An example of Textual Description for a Choice -Criteria

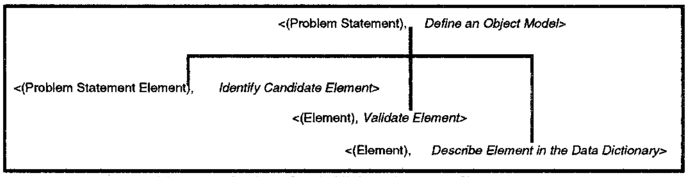
As illustrated in the table above, *arguments* serve as a basis to define *choice criteria*. Each *choice criterion* is a condition for selecting one alternative, it is represented by a logical formula combining arguments. For instance, CC2 expresses that if a2 or a3 or a4 or a5 are true, then the context <(Class), *Delete Class*> should be chosen.

There are two major differences between the *choice-based context* and the *executive-based context* : the first one lies in the absence of any alternatives in the latter and the second is that a choice-based context has no direct consequence on the product under development.
Finally it is important to notice that the alternatives of a choice-based context are contexts too. In the case of figure 3 alternatives are executive-based contexts. But they could be choice-based contexts introducing what is referred to as a refinement link between contexts.

2.2.3 Plan-based Context

In order to represent situations requiring a set of decisions to be made to fulfil a certain intention (for instance to build the object model in the OMT methodology) the way-of-working model includes a third type of context called plan-based context. A plan-based context can be looked upon as a macro issue which is decomposed in sub-issues, each of them corresponding to a sub decision associated to a component decision of the macro one. As alternatives for a choice-based context, components of a plan-based context are contexts too related through a *composition link.*

For instance, *the definition of an Object Model* (figure 4) corresponds to a complex issue, for two reasons. On one hand, the Object Model is a complex object composed of a set of elements which are instances of concepts characterising the structure of the target system and, on the other hand, the "*Define an Object Model*" decision requires a set of decisions for identifying, validating and describing these elements.



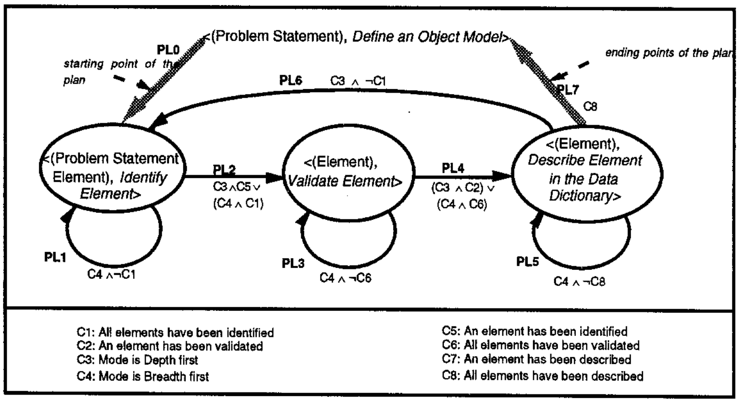
<(Problem Statement), Define an Object Model>
<(Problem Statement Element), Identify Candidate Element>
<(Element), Validate Element>
<(Element), Describe Element in the Data Dictionary>

Figure 4: An OMT Plan-based Context

In the above example, the plan-based context <(Problem Statement), *Define an Object Model*> is decomposed into three choice-based contexts, namely: <(Problem Statement), *Identify Candidate Element*>, <(Element), *Validate Element*> and

<(Element), *Describe Element in the Data Dictionary*>. For achieving the intention of defining an Object Model, the application engineer should iterate several times on the three decisions "*Identify Candidate Element*", "*Validate Element*" and "*Describe Element in the Data Dictionary*".

The ordering of the component contexts within a plan is defined by a graph named *precedence graph*. There is one graph per plan-based context. The nodes of this graph are the component contexts while the links -called *precedence links (PL)*- define either the possible *ordered transitions* between contexts or their possible *parallel enactment*. Based on arguments, a choice criterion may be assigned to a link. The choice criterion defines when the transition can be performed. Figure 5 illustrates the notion of precedence graph for the OMT plan <(Problem Statement), *Define an Object Model*>.


<(Problem Statement), Define an Object Model>
PL0
starting point of the plan
ending points of the plan
PL6 C3 ∧ ¬C1
PL7
C8
<(Problem Statement Element), Identify Element>
PL2
C3∧C5∨ (C4∧C1)
<(Element), Validate Element>
PL4
(C3∧C2)∨ (C4∧C6)
<(Element), Describe Element in the Data Dictionary>
PL1
C4∧¬C1
PL3
C4∧¬C6
PL5
C4∧¬C8

C1: All elements have been identified
C2: An element has been validated
C3: Mode is Depth first
C4: Mode is Breadth first
C5: An element has been identified
C6: All elements have been validated
C7: An element has been described
C8: All elements have been described

Figure 5: The Precedence Graph of the Plan <(Problem Statement), *Define an Object Model*>

One may observe in this graph that the validation of an element (i.e. the context <(Element), *Validate Element*>) is executed either if all the elements have been identified and if the mode chosen by the AE is *breadth first*, or if an element has been identified and if the AE wants to follow a *depth first* mode. This remark leads us to introduce the notion of *an execution path,* the AE can follow in a precedence graph, to execute a plan. A path comprises a set of precedence links and proposes a way to execute the plan which is most often, related to a particular mode (namely, breath first or depth first). Figure 5 defines two paths for the Object Model construction :
- If the AE chooses the *Depth first* mode, each time an element is identified, it is immediately validated and described in the Data Dictionary. Therefore, to build the complete Object Model, the AE should iterate on the execution of three decisions, namely "*Identify Element*", "*Validate Element*" and "*Describe Element in the Data Dictionary*". This path corresponds to a loop composed of the three component contexts of the plan and their relations PL2, PL4 and PL6.
- If the AE chooses the *Breadth first* mode, all the elements are first identified, by iterating on the context <(Problem Statement Element), *Identify Element*>. Following the precedence link labelled PL1, all the identified elements are then validated (PL3) and finally, all the validated elements are described in the Data Dictionary (PL5). In this path, Links PL2 and PL4 are used only once, whereas PL1, PL3, and PL5 are used as many times as there are elements to be created in the Object Model.

Thus, it is possible to evaluate the steps needed to create an Object Model. Apparently in the above description of paths, the depth first path requires less steps than the one defined for the breadth first mode. In fact it is possible to demonstrate that the size of both paths is the same. The component contexts are executed the same number of times, precedence links used in each case are different, but, as a link relates two component contexts, the same number of PL is used in both cases.

Let us illustrate this on a concrete example. Assuming the AE has to create an Object Model made of five elements, he will need to execute five times the three components of the plan. The total number of precedence links the paths are made of is the same : 16. The Depth First Path is equal to PL0+5PL2+5PL4+4PL6+PL7, and the Breadth First Path is equal to PLO+4PL1+PL2+4PL3+PL4+4PL5+PL7.

More generally, choice criteria impact the ordering of components, not the size of the path. Note that these measures have been done assuming that the components of the plan were executive-based contexts. But as components can be non atomic contexts, the metrics needs to be extended in order to take into account all the levels of deepness of the graph. Every component of a plan which is itself a plan-based context must be valued by the length of the path of its sub graph. For instance, if we assume that a plan-based context P1 is composed of a plan P2 and an executive-based context E1, the calculation will be : length(P1)= length(P2)+ length (E1) If P2 is composed of three executive-based contexts E2, E3 and E4, then : length(P2) = length(E2) + length(E3) + length(E4).
A more formal description of the precedence graph of the plan is provided in table 3, where <ctxt1> (resp. <ctxt2> and <ctx3>) represents <(Problem Statement Element), *Identify Element*> (resp. <(Element), *Validate Element*> and<(Element), *Describe Element in the Data Dictionary*>).

{PL0. { ([<ctxt1> . PL2/(c3∧c5) . <ctxt2> . PL4/(c3 ∧c2) . <ctxt3> . PL6/(c3 ∧ (not c1))]*[<ctxt1> . PL2/(c3 ∧c5) . <ctxt2> . PL4/(c3 ∧c2) . <ctxt3> . PL7/c8 ..]) ∪ ([<ctxt1> . PL1/(c4 ∧ (not c5)) . <ctxt1>]*. PL2/(c4 ∧c1) . [<ctxt2> . PL3/(c4 ∧ (not c6)) . <ctxt2>]*. PL4/(c4 ∧c6) . [<ctxt3> . PL5/(c4 ∧ (not c8)) . <ctxt3>]*. PL7/c8 ..) } }
*: 0 to n times ∪: union . : Sequence ..: End of the Graph

Table 3 : An Example of Textual Description for a Precedence Graph

This description of a precedence graph shows the use of logical operators to represent all paths of the graph, and to combine precedence links and nodes of the graph. This particular example does not show the use of the "shuffle" operator required to describe parallel execution paths in a plan.

To sum up we can say that plan-based contexts allow to prescribe the way to solve complex issues by decomposing them in sub-issues. The precedence graph associated to each plan provides a detailed description of the different possible execution paths and make possible the measurement of the process performance. The metrics associated to precedence graphs help in the evaluation of process performance costs.

2.3 The Notion of Tree

As introduced in the two previous paragraphs, contexts are related through refinement and composition links. These links can be recursively used to define relations between contexts. This recursive building results in a hierarchy of contexts called a tree. Trees represent meaningful hierarchies of contexts which may be independent one from the others.

A *tree* has a root which is, either a choice-based context, or a plan-based context. It has 'n' levels ($n \geq 2$). There is one level between a choice (resp. plan) based context and its alternatives (resp. components). The more global the root intention is, the more levels are required to support its decomposition and/or refinement. One may notice that the nodes of a tree are contexts of any of the three types : executive, choice or plan. This results from the fact that any component of a plan-based context or any alternative of a choice-based context may be a plan, an executive or a choice-based context. Moreover, whereas the intermediate nodes of trees can be either choice or plan-based contexts, executive-based contexts are always leaves of trees. The type of the leaves of the tree enable to verify whether the tree is complete or not: if all the leaves are executive, the tree is complete otherwise it is not. Figure 6 is an example of an OMT tree : the one which intends to guide the application engineer in constructing the Object Model.

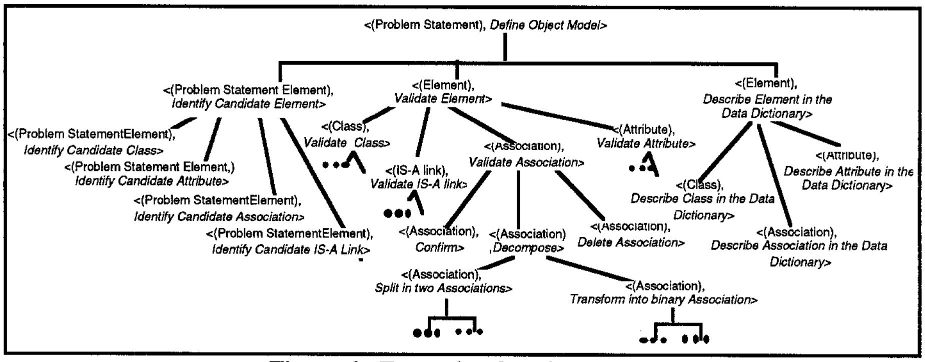

<(Problem Statement), *Define Object Model*>
<(Problem Statement Element), *Identify Candidate Element*>
<(Element), *Validate Element*>
<(Element), *Describe Element in the Data Dictionary*>
<(Problem StatementElement), *Identify Candidate Class*>
<(Problem Statement Element,) *Identify Candidate Attribute*>
<(Problem StatementElement), *Identify Candidate Association*>
<(Problem StatementElement), *Identify Candidate IS-A Link*>
<(Class), *Validate Class*>
<(IS-A link), *Validate IS-A link*>
<(Association), *Validate Association*>
<(Attribute), *Validate Attribute*>
<(Class), *Describe Class in the Data Dictionary*>
<(Attribute), *Describe Attribute in the Data Dictionary*>
<(Association), *Confirm*>
<(Association) *,Decompose*>
<(Association), *Delete Association*>
<(Association), *Describe Association in the Data Dictionary*>
<(Association), *Split in two Associations*>
<(Association), *Transform into binary Association*>

Figure 6 : Example of an OMT Tree

The issue of the above tree is to achieve the complete description of the static aspects of the Information System to be built. As shown in figure 6, several levels of refinement/decomposition of the root context, namely <(Problem Statement), *Define an Object Model*> are needed to achieve this complex issue. The complete definition of the Object Model is achieved by executing the components of the upper plan for each elements.

The first decomposition level of this tree have been discussed in the previous paragraph. It prescribes to organise the Object Model (OM) definition around three main decisions that the AE should make for all the OM elements. The way-of-working description suggests therefore to act in a plan composed of three contexts, namely, <(Problem Statement), *Identify Candidate Element*>, <(Element), *Validate Element*> and <(Element), *Describe Element in the Data Dictionary*>. The three component contexts are choice-based ones.

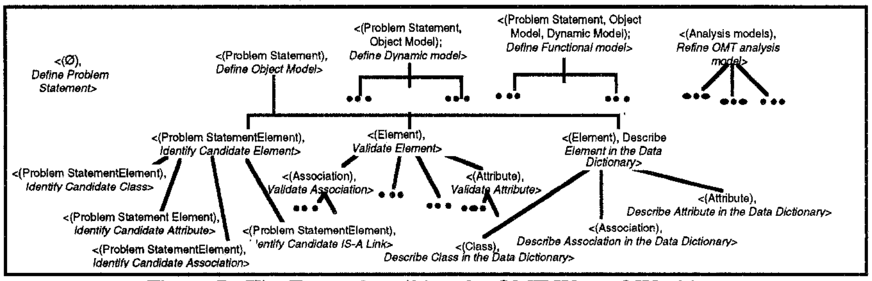
Each of their alternatives is associated to a specific kind of Object Model Element. For instance in the choice-based context <(Element), *Validate Element*>an element can be a class, an attribute, an association or an IS-A link, each of these concepts is related to an alternative of the above choice-based context. Thus, the alternatives are <(Class), *Validate Class*>, <(Attribute), *Validate Attribute*>, <(Association), *Validate Association*> and <(IS-A Link), *Validate IS-A Link*>.

Alternatives of the first component <(Problem Statement), *Identify Candidate Element*> are executive-based contexts and leaves of the tree. Similarly, alternatives of the third component, namely <(Element), *Describe Element in the Data Dictionary*> are executive-based contexts whereas the second component, namely <(Element), *Validate Element*> has alternatives which are choice-based contexts. The sub-trees of the tree have a variable depth. For instance, in the definition of an Object Model, the sub-tree dealing with Identification of Elements has two levels of depth, whereas the one dealing with Validation has five levels.

2.4 The Notion of Forest

A process model is usually composed of a collection of trees, that we call a *forest of trees.* The existence of multiple trees in a unique way-of-working comes from the independence of some contexts due to the independence of either situations or decisions. For example, a number of decisions which can be made from scratch, participate to contexts which are roots of trees. Moreover a way-of-working is difficult to define in one shot but will, more reasonably, be defined progressively by constructing fragments represented by trees.

For instance, the OMT methodology can be considered as a forest of trees, where a tree is associated to the definition of each analysis model, (the Object Model, the Dynamic Model and the Functional Model), to the definition of the Problem Statement, and to the refinement of the OMT schema.



<(Ø), Define Problem Statement>
<(Problem Statement), Define Object Model>
<(Problem Statement, Object Model); Define Dynamic model>
<(Problem Statement, Object Model, Dynamic Model); Define Functional model>
<(Analysis models), Refine OMT analysis model>
<(Problem StatementElement), Identify Candidate Element>
<(Element), Validate Element>
<(Element), Describe Element in the Data Dictionary>
<(Problem StatementElement), Identify Candidate Class>
<(Problem Statement Element), Identify Candidate Attribute>
<(Problem StatementElement), Identify Candidate Association>
<(Problem StatementElement), 'entify Candidate IS-A Link>
<(Association), Validate Association>
<(Attribute), Validate Attribute>
<(Class), Describe Class in the Data Dictionary>
<(Association), Describe Association in the Data Dictionary>
<(Attribute), Describe Attribute in the Data Dictionary>

Figure 7 : The Forest describing the OMT Way-of-Working

To achieve the definition of an OMT schema, the application engineer needs to go through the forest depicted figure 7. The application engineer starts by describing the Problem Statement with users. Then, he/she creates the three analysis models and finally improves these models applying the refinement guidelines provided by the methodology. As the elaboration of the analysis models can be done in any order, the building of each analysis model is represented by an independent tree. In [13], the authors recommend however to start with the Object Model, because it is easier to elaborate and helps in the definition of the Dynamic and the Functional models.

The construction of the Object Model which aims to describe the structure of the future information system has already been presented in the paragraph 2.3. We focus now on the way-of-working to build the Dynamic Model. Figure 8 summarises the OMT guidelines to construct this model which aims at describing the system behaviour as a set of state diagrams.

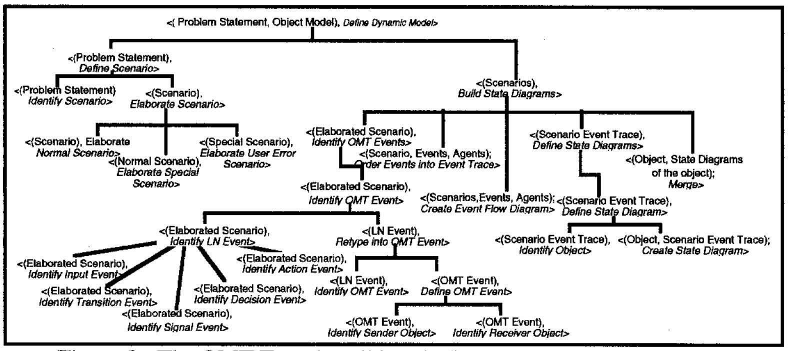

<(Problem Statement, Object Model), *Define Dynamic Model*>
<(Problem Statement), *Define Scenario*>
<(Problem Statement) *Identify Scenario*>
<(Scenario), *Elaborate Scenario*>
<(Scenario), Elaborate *Normal Scenario*>
<(Normal Scenario), *Elaborate Special Scenario*>
<(Special Scenario), *Elaborate User Error Scenario*>
<(Scenarios), *Build State Diagrams*>
<(Elaborated Scenario), *Identify OMT Events*>
<(Scenario, Events, Agents); *Order Events into Event Trace*>
<(Elaborated Scenario), *Identify OMT Event*>
<(Scenarios,Events, Agents); *Create Event Flow Diagram*>
<(Scenario Event Trace), *Define State Diagrams*>
<(Object, State Diagrams of the object); *Merge*>
<(Scenario Event Trace), *Define State Diagram*>
<(Scenario Event Trace), *Identify Object*>
<(Object, Scenario Event Trace); *Create State Diagram*>
<(Elaborated Scenario), *Identify LN Event*>
<(LN Event), *Retype into OMT Event*>
<(Elaborated Scenario), *Identify Input Event*>
<(Elaborated Scenario), *Identify Transition Event*>
<(Elaborated Scenario), *Identify Signal Event*>
<(Elaborated Scenario), *Identify Decision Event*>
<(Elaborated Scenario), *Identify Action Event*>
<(LN Event), *Identify OMT Event*>
<(OMT Event), *Define OMT Event*>
<(OMT Event), *Identify Sender Object*>
<(OMT Event), *Identify Receiver Object*>

Figure 8 : The OMT Tree describing the Dynamic Model Elaboration

To achieve the building of *State diagrams*, the authors suggest to start with the elaboration of *scenarios* as descriptions of sequences of *events* expressed in natural language. *Scenarios* are built in an incremental way. Normal *scenarios* are defined first and then *special* cases are taken into account; finally the exceptions which could happen are also added to improve the *special scenarios.*

In our approach, the elaboration of a *scenario* is represented as a plan where each context deals with a specific aspect of the construction or the refinement of a scenario. The refinement can be done either immediately after the description of the *Normal scenario* or when the three analysis models have been created.
Scenarios are used as a basis to identify the *events* happening in the system life cycle and to build an *Event Trace Diagram* describing events, their ordering and impacts on actors. The knowledge presented in the *Event Trace Diagram* is summarised into the *Event Flow Diagram* where events and actors are related in a static manner (without representing the events ordering).
These diagrams are then used to build *State Diagrams*. A state diagram describes the *state transitions,* of an object, each transition comprising two states and an event with the action or activity it triggers. As the consequence of the event is a modification of the state of the object, the event is represented by an arrow linking the initial state of the object to the final state it will be transformed in. A *State Transition* is created for each event identified in the *Event Trace Diagram,* it can be nested in order to express complex behaviours. The nested State Diagrams are obtained through several refinements. When all the needed *State Diagrams* have been elaborated, they are merged to define a unique Dynamic Model.

3 Discussion

In the IS community, quality criteria have been defined for evaluating conceptual schema, [9], but very few efforts have been made for defining quality criteria of IS processes. Method comparisons [6], [7] focus, for the process aspects, on the number of activities a process model is made of, on how in depth the decomposition of activities is done, etc. This helps to compare the nature of activities, but does not contribute to defining quality criteria applicable to process models. The Software Engineering (SE) community has been more involved with the definition of quality criteria [1], [4] to evaluate SE process models.
Based on the criteria established by the SE community, taking into account the various shortcomings of methods which have been reported in the literature [3], [16], [5] and also the specificity of guidance centred ways-of-working, we propose a set of five criteria, namely modularity, genericity, comprehensibility, granularity and guidance, to evaluate IS process models. In the following we show how our approach meets these criteria.

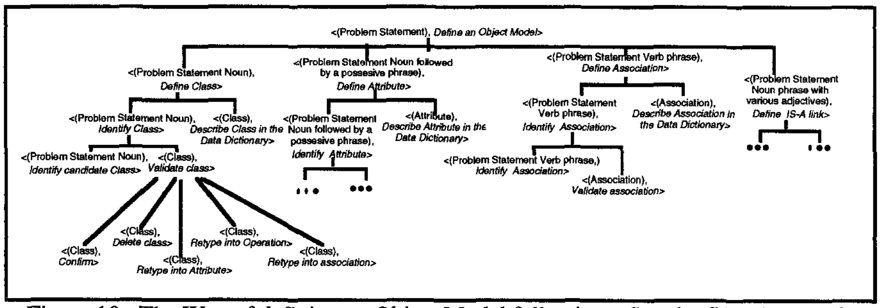
3.1 Modularity and Genericity : Towards Process Knowledge Capitalisation

The proposed approach takes the position that ways-of-working should have a theoretical foundation which is provided in our case by the way-of-working model. This to ensure properties such as *modularity* and *genericity* .

Modularity has proved to be a key issue in many aspects of information and computer science. We believe that *modularity* of ways-of-working descriptions is useful to ensure an easy evolution and improvement of the method. It facilitates training of the methodology, makes easier the construction of a way-of-working by composing existing fragments and facilitates automation in a tool based environment.
Our approach structures a way-of-working into units such as contexts and trees which ensures modularity of process models descriptions. Modules of a way-of-working are obtained by applying the decomposition and refinement mechanisms to contexts. These notions are generic i.e. independent of a particular methodology and powerful enough to represent many various and complex situations in an homogeneous fashion.

The application of the approach for describing a number of existing ways-of-working [11] has demonstrated its genericity : similar process patterns appear several times in the same methodology and even in different methodologies. The properties of modularity and genericity introduce the idea of using *factorisation means* to allow both a more concise and meaningful description of a given way-of-working and to facilitate the reuse of generic process patterns.

In order to optimise the description of a way-of-working, contexts and trees sharing common features are factorised: If several contexts of the same type share common properties such as an intention or a product part of the same type we suggest to factorise them and to introduce a new higher abstraction level in the way-of-working description where guidelines are expressed at the factorised components level. This factorisation approach is illustrated in the following with the OMT methodology example. The stepwise approach recommended by the OMT methodology [13] corresponds to the plan sketched in the figure 10. The plan suggests to proceed in turn to the identification of each object model element.



<(Problem Statement), *Define an Object Model*>
<(Problem Statement Noun), *Define Class*>
<(Problem Statement Noun followed by a possesive phrase), *Define Attribute*>
<(Problem Statement Verb phrase), *Define Association*>
<(Problem Statement Noun phrase with various adjectives), *Define IS-A link*>
<(Problem Statement Noun), *Identify Class*>
<(Class), *Describe Class in the Data Dictionary*>
<(Problem Statement Noun followed by a possesive phrase), *Identify Attribute*>
<(Attribute), *Describe Attribute in the Data Dictionary*>
<(Problem Statement Verb phrase), *Identify Association*>
<(Association), *Describe Association in the Data Dictionary*>
<(Problem Statement Noun), *Identify candidate Class*>
<(Class), *Validate class*>
<(Problem Statement Verb phrase,) *Identify Association*>
<(Association), *Validate association*>
<(Class), *Confirm*>
<(Class), *Delete class*>
<(Class), *Retype into Attribute*>
<(Class), *Retype into Operation*>
<(Class), *Retype into association*>

Figure 10 : The Way of defining an Object Model following a Step by Step Approach

The definition of each element, namely class, attribute, association and is-a link, is described in a two levels hierarchy of plans whose component decisions are of the same type, namely for the upper plan *"Identify ...", "Describe ... in the Data Dictionary"*, and for the lower one *"Identify candidate..."* and *"Validate ..."*.
We suggest to apply a factorisation mechanism to reorganise this hierarchical description as shown in the figure 6. This transformation leads first, to create the "factorised" contexts <(Problem Statement), *Identify Candidate Element*>, <(Element) *Validate Element*> and <(Element) *Describe in a Data Dictionary*>, second to relate them at a new level in the hierarchy to express things at the factorised level and third, to introduce choice-based contexts allowing to select the alternative appropriate to the element to be defined as a refinement of the factorised contexts. These alternatives are then refined as in the initial description using the two level plan based description.
This factorisation attitude leads to a more synthetic description of a way-of-working but also helps in identifying generic process chunks that can be reused in other methodologies. For instance, the generic context <(Problem Statement Element), *Identify Element*> has been instanciated for the definition of other ways-of-working (e.g. O*, E/R., OOA).

3.2 The Granularity Feature

The ability to describe ways-of-working *at different levels of granularity*, in a uniform manner is provided by the recursive application of refinement and decomposition mechanisms on contexts. High level intentions such as *"the definition of an OMT schema "* or lower level ones such as *"the identification of an event of the OMT dynamic model "* are modelled using the same concept (*context*) and can be hierarchically connected using the refinement and decomposition mechanisms. We believe that this is a powerful feature for modelling complex process situations which are not supported by most existing process models. As alternatives of choice-based contexts and components of plan-based contexts are contexts too, the way-of-working model provides an elegant solution to solve the granularity problem by describing hierarchies of contexts.

3.3 The Comprehensibility Capability

The *graphical representation* used to represent trees and precedence graphs leads to an easy understanding of the methodological process. It enhances the *comprehensibility* of the method. The highest levels of the hierarchy describe globally the process by presenting an overview of the way-of-working. The lowest levels of the hierarchy provide a fine-grained description of the executable decisions. The representation of the precedence graph clearly presents how a plan is executed.

3.4 The Guidance Feature

The model has been primarily designed to capture formal and heuristics knowledge able to support the application engineer in his/her activities. Each type of context provides a specific guidance. Executive-based contexts automate the execution of actions. Arguments and choice criteria advise the AE when he/she has to make a choice. Modes and approaches make guidance within a plan more flexible. Finally the various levels of granularity enable to guide the AE from the beginning to the end of his/her work.

4 Conclusion and Future Work

In this paper, an approach for defining ways-of-working in a structured and systematic manner has been presented, exemplified with the OMT methodology and evaluated against five quality criteria. The approach leads to construct ways-of-working able to represent a large variety of situations in a decision oriented manner, with a reasonable level of genericity, at various levels of detail, in a modular and hierarchical manner. The graphical representation of the way-of-working makes its understanding by the application engineer easier and enhances the guidance capabilities. Finally the factorisation mechanism helps optimising the way-of-working description and the discovery of reusable process chunks. In addition, metrics can be defined to measure the cost of using a given way-of-working. As illustrated in section 2.2.3, metrics can be used to compare different paths in a precedence graph. It can be defined for the whole hierarchy in order to calculate the time needed to build an application and to compare two paths in a given hierarchy or to compare several methodologies.

We are looking for the introduction of new features such as dynamic change which has been identified as a key issue of SE process models. We are currently implementing the approach in *a CAPE Environment* able to monitor method engineers in the definition of ways-of-working, and to guide application engineers in the use of ways-of-working for elaborating high quality products. Another topic of research work is to define *a complete set of metrics* for evaluating and comparing methodological processes. Finally, we are populating a *library of reusable process chunks* which will be the basis of a *meta-methodology* for defining a new way-of-working within a specific methodology.

References

1. P. Armenise, S. Bandinelli, C. Ghezzi, A. Morzenti, *A survey and assessment of software process representation formalisms*, Int. Journal of Software Engineering and Knowledge Engineering,Vol. 3, No. 3, 1993.
2. Brinkkemper S, "Formalisation of Information systems modelling", Ph.D. Thesis, University of Nijmegen, Thesis Publishers, Amsterdam, 1990.
3. Bubenko J.R., Bubenko J.A., "Information System Methodologies - A Research View", in Olle T.W., Sol H.G., Verrijn-Stuart A.A. (Eds.) : "Information Systems Design Methodologies : Improving the practice, North-Holland, Amsterdam, The Netherlands, 1986, pp 289-318.
4. M. Dowson, *Software Process Themes and Issues*, IEEE int. conf. , 1993.
5. L.J.B. Essink, *A Conceptual Framework for Information System Development Methodologies*, Memorandum INF-86-34, Univ. Twente, Enschede, The Netherlands, 1986.
6. Goor G. van den, Brinkkemper S., Hong S.; "Formalization and comparison of six Object Oriented Analysis and Design Methods", Report Method Engineering Institute, University of Twente, 1992.
7. Iivari J., "Object-oriented information sytems analysis: A comparison of six object-oriented analysis methods", IFIP Int. Conf. on "Methods and associated Tools for the Information Systems Life Cycle" (A-55), North Holland, 1994.
8. Kumar K., Welke R. J., "Methodology Engineering : A prposal for situation-specific Methodology Engineering", Challenges and strategies for research in Systems Development , J. Wiley & sons Ltd., 1992, pp. 257-269.
9. Lindland O.I., Sindre G., Solvberg A., "Understanding Quality in Conceptual Modeling", IEEE software, March 1994.
10. Lyytinen K., Smolander K., Tahvanainen V.-P., "Modelling CASE Environ -ments in sytems Work", CASE89 conference papers, Kista, Sweden, 1989.
11. Plihon V., "The OMT,The OOA ,The SA/SD,The E/R ,The O* , The OOD Methodology" NATURE Deliverable DP2, 1994.
12. Prakash N., "A Process View of Methodologies", Proc 6th Int. Conf. on "Advanced Information Systems Engineering" (CAISE), Sprg. Vg,1994.
13. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Loresen : "Object-oriented modeling and design", Prentice Hall international, 1991.
14. Schwer S. R. , Rolland C. ,"Formalization of the Information System Design Process", Internal Report C.R.I., 1994.
15. Seligmann P.S., Wijers G. M., Sol H.G., "Analyzing the structure of I.S. methodologies, an alternative approach", in Proc. of the 1st Dutch Conference on Information Systems, Amersfoort, The Netherlands, 1989.
16. Sol H.G.: "Kennis en ervaring rond het ontwerpen van informatiesystemen", Informatie, Vol. 27, N°3, 1985.
17. Wynekoop J.d., Russo N.L., "System Development methodologies: unanswered questions and the research-practice gap", Proc. of 14th ICIS Orlando, USA, 1993, pp. 181-190.