# A Unified Framework for Modeling Cooperative Design Processes and Cooperative Business Processes

Colette Rolland, Selmin Nurcan, Georges Grosz
*Université Paris 1 - Panthéon - Sorbonne, Centre de Recherche en Informatique*
*17, rue de Tolbiac, 75013 Paris, FRANCE, email : {rolland, nurcan, grosz}@univ-paris1.fr*

**Abstract**

*We look at any cooperative design process as a decision making process, i.e. a non deterministic process. The process is performed by responsible agents having the freedom to decide how to proceed according to their assessment of the situation they are faced to. However, the cooperative design process cannot be an ad-hoc and chaotic process. We look at it as a repeatable process made of steps resulting each of the application of the same pattern for decision making. The pattern views a decision as a choice of the way to proceed in a given situation to achieve an intention. An intention can be fulfilled in different ways depending on the situation being considered. We propose a process meta-model for describing such a pattern of decision making. This meta-model can also be used for modeling cooperative business processes whether well structured or ill-structured. In order to deal with a wide range of cooperative processes, we propose a single process meta-model which provides the structuredness of the predefined models and the flexibility of ill-structured processes.*

## 1. Introduction

It is traditional to look to any engineering activity from both the product point of view and the process points of view. The product is the desired result, the process is the route followed to reach the result. Methods were classically focused on the product aspect of systems development and have paid less attention to the description of formally defined ways-of-working which could be supported by CASE environments. Clearly, there is a high need for methods and tools which offer *process guidance* to provide advice on which activities are appropriate under which situations and how to perform them [1 ,2 ,3 ,4]. We propose a way-of-working which intends to provide such guidance.

Our approach is composed of three complementary elements :

(1) a set of models used for describing the system to be constructed and the organization in which it will operate,

(2) a way-of-working supporting the usage of concepts,

(3) a set of tools supporting the way-of-working.

This paper focuses on the presentation of the way-of-working which enables the cooperative design process management on a basis of a method rather than on intuition. It aims at organizing and structuring the design process. It provides advice on what should be considered during this process (goals, roles, etc.), why and how it should be analyzed (goal decomposition, role dependency study, etc.) following some relevant techniques (brainstorming, SWOT analysis, etc.). It also suggests which problem should be tackled next and provides some argument to help in the making of the most appropriate design decision. Finally, it includes means to support cooperative design processes and cooperative business processes including brainstorming, exchange and emergence of ideas. Due to the tool support, some process automation is possible and tracing facilities emphasize the recording of the rationale and argumentation provided throughout the process.

This approach, called EKD (Enterprise Knowledge Development), is currently being applied, in the ESPRIT project ELEKTRA[1] [5] for reorganizing electricity companies and designing new solutions.

This paper is organized as follows: Section 2 presents the cooperative process meta-model. Section 3 focuses on EKD way-of-working for cooperative design. Section 4 presents the guidance offered by the EKD process. Section 5 illustrates the use of the presented meta-model for describing both cooperative design processes and cooperative business processes.

## 2. The cooperative process meta-model

We distinguishes three levels of process modeling :

-At the instance level, process traces are recorded. A process is a description of the route followed to construct a product or a service. The output of a process is a *product*, it can be a requirements specification, a conceptual schema, a service to a client in an organization or a set of business goals. A process and its related product are specific to an application.

-At the type level, ways-of-working are defined. A way-of-working is a process model, i.e. a description of process. It is an anticipation of what the process will look like. A process is then, an instantiation of a process model. A process model is specific to a method.

-At the meta-type level, we define the generic concepts used to represent any process model. Along with their relationships, those concepts constitute the process meta-model. Ways-of-working are therefore, instances of the process meta-model. A process meta-model is method independent.

[1] This work is supported by the ESPRIT project ELEKTRA (N° 22927) founded by the EEC in the context of the Framework 4 Programme.

As we shall see in section 3.1, these three levels can be used to provide guidance at three different levels of abstraction, the domain specific level, the method-specific level, and the generic level, respectively.

We propose a *meta-model* as a basis for cooperative process model definition [1, 6, 7]. The process meta-model allows us to deal with many different situations in a flexible, decision-oriented manner. Moreover, it can support different levels of granularity in decision making as well as non-determinism in process performance. It identifies a decision in context as the basic building block of ways-of-working and permits their grouping into meaningful modules. Parallelism of decisions and ordering constraints are also supported.

In this section, we present first the basic process model developed in [1]. In paragraph 2.4 we introduce extensions to deal also with cooperative processes.

## 2.1. The concept of context

The central concept of the process meta-model is the one of *context* (figure 1) which associates a situation with an intention. A *situation* is a part of the product it makes sense to take a decision on. Situations can be of various granularity levels; they can be either atomic like an attribute of an object class or they can be coarse-grained like the whole object schema. A decision reflects a choice that a user can make at a given moment in the process. A decision refers to an *intention*. An *intention* expresses what the user wants to achieve.

A *context* is the association of a *situation* and an *intention*. Acting in a context corresponds to a step in the process. That is, when placed in a given situation, and in order to progress in the process, the user has to take a decision referring to an intention (figure 1).

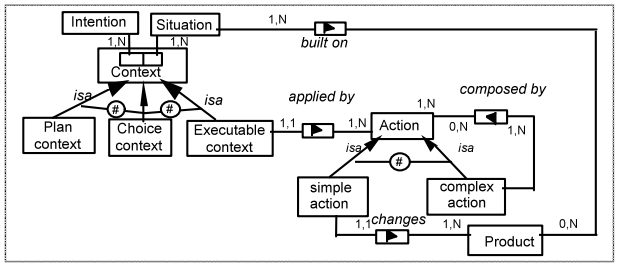


**Figure 1 : The concept of Context**

## 2.2. The different types of contexts

A situation exists at different levels of granularity. Further, decisions have consequences which differ from one granularity level to another. The different contexts are classified (figure 1) according to their consequences in the meta-model into *executable contexts*, *plan contexts*, and *choice contexts*.

**2.2.1. Executable context.** At the most detailed level, the execution of any process can be seen as a set of transformations performed on the product, each transformation resulting from the execution of a deterministic action. Such an action is a consequence of a decision made in a certain context. This leads to the introduction of the concept of an *executable context*.

< (Reservation request+resources); *Create reservation* >

**Figure 2. Example of an executable context**

An *executable context* (figure 2) implements a decision, its intention is realized by an action. Therefore, in the meta-model (figure 1), an executable context is associated with an action. An *action* performs a transformation of the product, it is the implementation of a decision. Performing an action changes the product and may generate a new situation which is itself, subject to new decisions. The *action* can be complex or simple. A *complex action* is composed of *actions*. A *simple action* changes the product by creating, updating or deleting it.

**2.2.2. Choice context.** A user may have several alternative ways to fulfill a decision. Therefore, he/she has to select the most appropriate one among the set of possible choices. In order to model such a piece of process knowledge, we use a second specialization of the concept of context, namely the *choice context* (figure 3).

A *choice context* corresponds to a situation which requires the exploration of alternatives in decision making. Each alternative is an approach or a strategy for the resolution of the issue being faced by the user in the current situation. By definition, a choice context offers a choice among a set of strategies, all of them achieving the same intention.

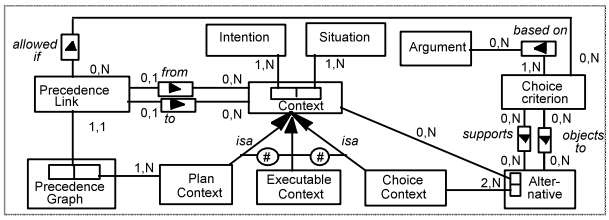


**Figure 3. The representation of the concept of context**

In the process meta-model, the various alternatives of a choice context are represented in the *alternative relationship* (figure 3). They are associated to choice criteria based on arguments.

*A choice criterion* is a combination of arguments which *supports* or *objects to* an alternative of a choice context. It may provide priority rules to select one *alternative* among several depending on the *arguments*.

Since alternatives of a choice context are also contexts, contexts may share an *alternative* relationship (figure 3), leading to *alternative-based hierarchies of contexts*. The alternative-based relationship among contexts allows the refinement of large-grained decisions into more fine-grained ones. This is one of the means by which the process meta-model handles the granularity problem (figure 4).

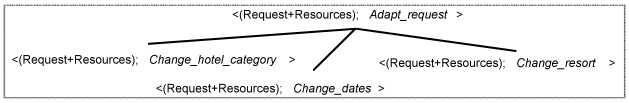

**Figure 4. Example of a choice context**

The notions of *alternatives* and *choice criteria* allow the way-of-working to support the user in exploring and selecting the most appropriate strategy to resolve an issue.

**2.2.3. Plan context.** In order to fulfill an intention associated to a certain situation, a user may be required to take a set of decisions on corresponding situations; he/she has to follow a plan. To this end, a third specialization of context, namely, *plan context* is introduced. A *plan context* is an abstraction mechanism by which a context viewed as a complex issue can be decomposed in a number of sub-issues. Each sub-issue corresponds to a sub-decision working on a sub-situation. The decomposition of context is another means provided by the meta-model to solve the granularity problem.

The component contexts can be of any type, i.e. executable, choice or plan. For example, for the intention named "*Process_Request*" to be fulfilled, the three intentions "*Analyse_Request*", "*Adapt_Request*" and "*Create_Reservation*" must be satisfied. This is modeled (fig. 5) by a plan called : "<(Request + Resources), *Process_Request*>", it is decomposed into three contexts: "<(Adapted Request + Resources), *Analyse_Request*>", and "<(Analysed Request + Resources), *Create_Reservation*>" executable contexts, and "<(Request + Resources), *Adapt_Request*>" choice context.

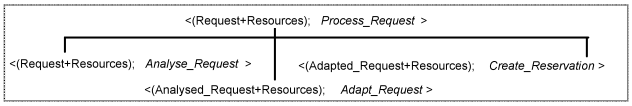


**Figure 5. Example of a plan context**

In the process meta-model the decomposition of a plan context into its more elementary contexts is represented (figure 3) by the relationship *precedence graph* between *context* and *plan context*. The ordering of the contexts, within a plan, is defined by the *precedence graph*. The nodes of this graph are contexts while the links -called *precedence link*s- 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. Flexibility is introduced by allowing several sets of possible parallel or ordered transitions to be defined in the same graph.

Decomposition of contexts can be made iteratively leading to hierarchies of contexts linked by the *decomposition* link. Notice that this link corresponds in figure 3 to the composition of the *precedence graph* relationship with the *from* and *to* relationships.

Each type of context influences the on-going process in a different manner: an executable context affects the product and generates a new situation, which itself becomes the subject of decisions; a choice context does not change the product but helps to further the decision making process through the refinement of an intention; a plan context provides the means to manage the complexity of an intention by providing a decomposition mechanism. Performing decomposition and refinement iteratively allows the users to reach executable intentions and thus, to act on the product.

## 2.3. The structure of way-of-working

Contexts in the meta-model have hierarchical relationships of two types, decomposition and refinement. In the way-of-working, we suggest a grouping based upon these links. The modules resulting from this grouping are hierarchies of contexts called *trees*. A way-of-working can be composed of several trees. This leads to the final vision of a way-of-working as a *forest* of trees (figure 6).

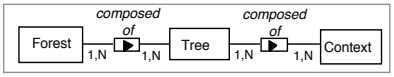


**Figure 6. The way-of-working structure**

## 2.4. Extensions to deal with cooperative processes

According to the process meta-model, acting in a context should correspond to a step in the cooperative process. Since there are a number of participating users in cooperative processes who discuss and work with one another, there is need to have specific provision for the conversational action in the process meta-model. Additionally, since users play different roles in organizations and participate in the cooperative process from the point of view of this role, it is necessary to explicitly bring the notion of a role in the meta-model.

**2.4.1. The concept of role.** A role is the definition of an organizational intention shared by a collection of users, all of whom have the same privileges and obligations to a set of work processes in an organization. For example, the role of a reservation service clerk, that of an accounts officer, etc. In a given situation, a user has an intention (because of his/her role in this process), and that makes him/her progress in the cooperative process.

To this end, we introduce the concept of *role*, and then specialize it into *individual role* and *group role* (figure 7). For example, the reservation service clerk is an individual role whereas public relations team is a group role. A group role *contains* several individual roles.

We attach the context of the process meta-model to a role. This captures knowledge about which decision can be taken by which role. Therefore, the basic division of responsibility in cooperative processes is imposed on the set of decisions of the meta-model. This helps us in representing coordination of roles, providing access control, and in giving more appropriate guidance which is completely tailored to the role.

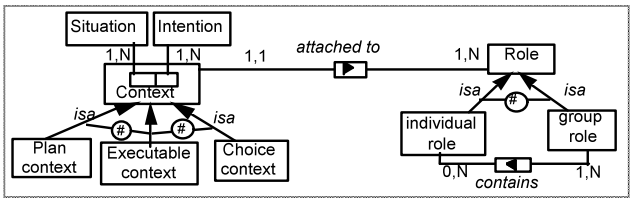


**Figure 7. The context is attached to a role**

**2.4.2. Specialization of the action and product concepts.** We classify actions into two types (figure 8): *individual action* and *conversation action*.

Individual actions perform transformations of artifacts while conversation actions create messages. Therefore, we classify the concept of *product* into *artifact* and *message* (figure 8). *Artifact* represents the objects of the information system. To keep track of conversations, we introduce the *message* concept as the basic component of the conversational activity. A *message* may *concern* several *artifacts*.

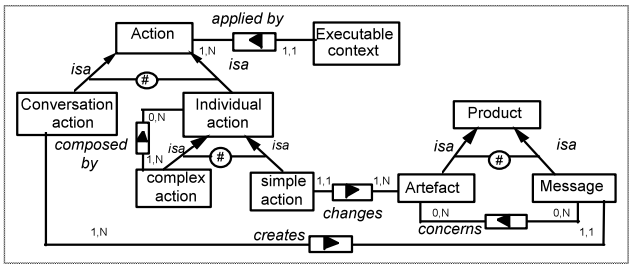


**Figure 8. Actions and products that they transform**

Figure 9 shows an executable context which is applied by an individual action. An *individual action* is performed by an *individual role* (figure 10).

< (Reservation request+resources); *Create reservation ;* Reservation_service_clerk >

**Figure 9. An executable context according to the cooperative process meta-model**

We want also to deal with group activities, in the sense that several participants can synchronously act in the same activity by exchanging messages. We represent this type of cooperation by the *conversation action*. The *conversation action* is *performed by* a *group role*. It creates several messages, each message being *produced by* an individual role *contained by* the previous group role (figure 10). New *contexts* may *emerge* from any *conversation action* (figure 10). These contexts can be executable and associated to actions, which may themselves be conversa-tional. These, in turn, trigger new contexts and so on.

## 2.5. Highlights

For many organizations, well-structured and ill-structured procedures coexist in work processes and must be managed in the final solution which describes the organization [6]. Allowing the description of both ill and well-structured procedures within a single process meta-model aims to make transition between different types of group activities transparent. This requires homogeneity and coherence of handled concepts.

An instantiation of the cooperative process meta-model results in a cooperative process model allowing to deal with a large variety of situations in a decision-oriented manner.

The concept of plan context enables the cooperative process meta-model to deal with well-structured cooperative processes which require the use of a control model [6, 7]. The corresponding precedence graph defines the ordering of the component contexts and, consequently, the coordination of the various roles.

The alternative-based guidance of the choice context leaves freedom to users who can make a choice which may not even be one of the predefined alternatives proposed by the way-of-working. This feature allows the cooperative process meta-model to deal with exception handling in cooperative processes.

The concept of conversation action allows the cooperative process meta-model to deal with ill-structured cooperative processes and the unstructured component of globally well-structured cooperative processes.

We believe that what we just introduced consists in the minimal necessary set of concepts required to describe cooperative processes. Indeed, these concepts constitute the basic and could extended toward specific requirements or use (e.g. : role hierarchy).

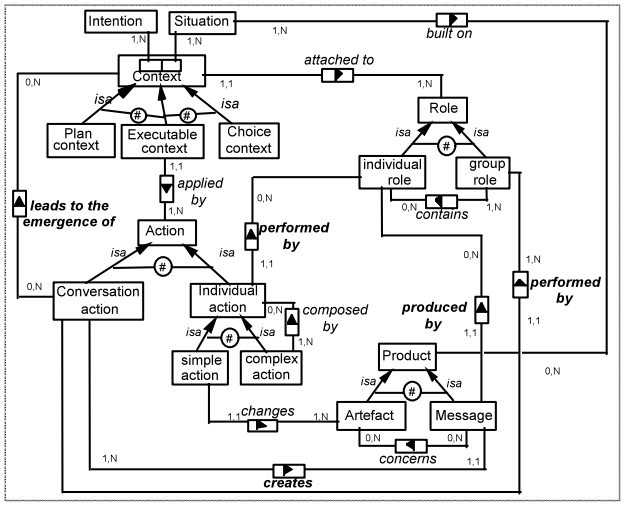


**Figure 10. The cooperative process meta-model**

# 3. The way-of-working for cooperative design

## 3.1 The process is guided

This section introduces our view of cooperative design processes, the concepts and paradigm underlying the EKD way-of-working and how processes are guided. First, we look to any cooperative design process as a *decision making process*, i.e. a non deterministic process. The process is performed by responsible agents having the freedom to decide how to proceed according to their assessment of the situation they are faced to. Agents do not necessarily follow a predefined plan of actions. Cooperative design requires a number of decisions to be made, on what to consider in the existing organization, on the study of alternative solutions, on the selection of the most appropriate solution, etc..

Secondly, the cooperative design process cannot be an ad-hoc and chaotic process. We look at it as a repeatable process made of steps resulting each of the application of the same *pattern for decision making*. The proposed EKD way-of-working is entirely based on this pattern.

Third, the pattern views a decision as the choice of the *way to proceed* in a given *situation* to achieve an *intention*. In terms of the cooperative process meta-model, a decision is contextual in the EKD approach. An intention can be fulfilled in different ways depending on the situation being considered.

Therefore, if we visualize the pattern (figure 11) as having an input, a body and an output, then *input* is a *couple <situation, intention>* , i.e. a context.

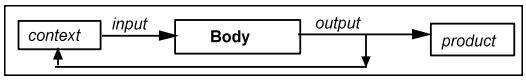


**Figure 11. The EKD decision making pattern**

Each application of the pattern in a specific EKD process will deal with a specific *input*, i.e. a specific *context*. The *output* of the EKD decision making pattern is either a modified *product* or new *contexts*.

The *body* of the decision making pattern provides the knowledge to make the decision In other words, the pattern is intended to provide guidance on how to proceed to achieve the intention in the given situation. The body makes use of the different types of knowledge as follows:

*Domain specific knowledge:* heuristical knowledge which partly constitutes the know-how of EKD engineers.

*EKD knowledge:* knowledge independent of any particular domain but *specific to EKD*. For instance, while classifying a goal, an engineer refers to some existing and well understood classifications, the elements guiding his/her selection of the appropriate class are known "a priori", they are reused for the classification of every goal. This type of knowledge is specific to EKD and can be used in any organizational setting.

*Generic knowledge:* when an engineer has to solve a new design problem, he/she could structure his/her reasoning by looking for alternative ways to solve the problem or by decomposing the problem into smaller problems. This type of knowledge is fully *generic* and not tailored to EKD.

The decision making pattern is tailored to *always provide guidance*: the domain specific knowledge is the more accurate whereas the generic knowledge is the default guiding option; the EKD knowledge providing support dedicated to the use of the specific EKD models.

These three types of guidance can be related to the levels of abstraction introduced in section 2. Generic guidance is at the meta-model level; EKD guidance is at the model level and domain specific guidance relates to the process level [8] .

Decision making might require emergence of ideas, exploration of choices, argumentation of various alternatives and perhaps deliberation among the stakeholders involved in the process. Section 4.1 shall demonstrate how the generic decision making pattern takes these aspects into account .

### 3.2. The process is incremental and dynamic

The suggested way-of-working makes the EKD process *cyclic*, each step of the process repeats the EKD decision making pattern. As a consequence, the product which is the target of the process (i.e. the new company organization of its business processes) is *incrementally constructed*. In addition, the sequencing of steps is not fixed a priori. Steps *dynamically* follow one another. The dynamicity is brought by the decision making pattern which does not impose any predefined ordering in the decision making process but allows the EKD engineers to switch from one context to another depending on changed situations and intentions.

### 3.3. The process is supported by software tools

The way the EKD environment provides guidance in the performance of the process can be explained using the Dowson's framework [9]. The framework introduces three interacting domains: *process modeling, process performance* and *process enactment*. The *process modeling* domain captures all activities performed for modeling software development processes: process model definition, process model specialization, etc.. The *process enactment* domain encompasses what takes place in a process to support process performance based on the process definitions. This is essentially an interpretation of an instantiated process model that guides, enforces or partly automates process performance. The relationship between the process modeling and the process enactment domains is the instantiation of the process model. The instantiated model is then used within the process enactment domain for supporting process performance. The *process performance* domain is defined as the set of activities conducted by human agents and non human agents (e.g. computer). The relationship between the process performance and the process enactment domain is twofold. On the one hand, the process enactment domain supports, controls and monitors the activities of the process performance domain. On the other hand, the process performance domain provides feedback on process performance, to enable process adjustment.

The process model supporting the EKD way-of-working comprises three classes of process model fragments; each of them being adapted to the three types of guidance introduced in section 3.1. We call them *method chunks*, and therefore the cooperative design environment uses *generic method chunks*, *EKD method chunks* and *domain specific method chunks*. All chunks are stored in the library of the EKD environment and are accessible at any moment during process performance.

## 4. Guidance in EKD process

### 4.1. Generic guidance

The method knowledge library comprises only one generic guideline that we refer to as the *generic method chunk* or simply *generic chunk*.

The chunk is applicable in *situations* where the two other types of guidelines do not hold. The guideline aims to fulfill the goal called "*progress*". It proposes a help strategy for progressing in the EKD process which offers four options: first, *do, plan* and *choose,* each of them corresponding to a given type of context, *executable*, *plan* and *choice context,* respectively (see section 2).

- The *do* option corresponds to a straight-forward resolution strategy. It should be chosen when the method engineer knows exactly what needs to be done in order to fulfill the context's intention. The engineer is required to specify the design action(s) and their effects on the design

product. We call this type of context *executable*, its intention can be directly implemented through actions.

• The *choose* option corresponds to a resolution strategy which requires the exploration of alternative paths. It should be selected when the method engineer thinks about different alternative ways for progressing with regard to the input context but has not make up his/her mind about the one to select. The generic chunk proposes to specify all possible alternative paths and to elaborate an argument for each of them in order to choose. Based on the proposed arguments, the enactment leads to the selection of the alternative path which looks the most appropriate. The initial context is said a *choice* context.

• The *plan* option follows a planning strategy. The method engineer has in mind a plan for achieving the context's intention and will progress by building a plan of decisions to be made. The enactment consists of plan execution. The initial context is called a *plan* context.

Note that the two last options correspond to the classical reduction operator in the problem reduction approach to problem solving [10].

However, parts of the EKD process are dealing with wicked and ill-defined problems for which even the generic guidance provided by the decision making pattern might be found too inflexible. The discovery of goals is an example. Setting the opportunities, weaknesses, threats and strengths for a design process to occur is another example. As pointed out in [11, 12, 13], finding goals is very hard and no efficient way of solving this problem is known. Organizing cooperative work and brainstorming sessions is probably the most adapted approach to deal with this kind of highly creative activity in order to make ideas emerge. The problem is therefore, to be able within the EKD way-of-working, to support both ill-structured and well (or better)-structured processes.

In order to take into account the cooperative work, we complete the generic chunk by a fourth strategy called *brainstorm* (figure 12). This strategy is supported by the argument "the current situation requires cooperative brainstorming". The associated alternative is an executable context <input context, *use the brainstorm strategy*> leading to the execution of the conversation action within the required group role and having the initial input context as situation. The EKD engineer selects this strategy when he/she cannot achieve alone the input intention, for example, "*Operationalize goal*".

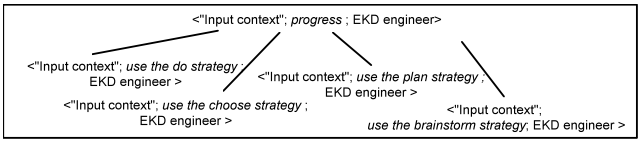


**Figure 12. The generic method chunk**

## 4.2. EKD guidance

*EKD guidance* is based on *EKD knowledge*, i.e. knowledge specific to EKD. This is knowledge for supporting EKD engineers to specifically undertake the cooperative design process in an organization using the EKD models.

The EKD knowledge supports for example, the construction of the different models representing the initial enterprise state (the initial product) as well as the future enterprise state of the organization (the design product), the expression of alternative strategies for design, the evaluation of these strategies, as well as other kinds of activity such as brainstorming, cooperative work, etc..

We express this knowledge as we did for generic knowledge, i.e. using the process meta-model and the different types of context. However there is one major difference : the EKD knowledge is expressed at the type level, i.e. the level of specific classes of EKD phenomena such as "identifying goals", "operationalizing goals", "finding design models meeting specific goals", etc..

More generally, the EKD knowledge can be reused for decision based guidance in many different cooperative design processes within different companies. An EKD method chunk is reusable any time the situation type matches elements of a specific product and the intention type matches the current intention of the EKD engineer.

The enactment of the decision making pattern at the EKD level is similar to what we have illustrated for generic guidance. The main difference lies in the retrieval of the method chunk. The retrieval of an EKD chunk is based on matching: Assuming that the engineer has chosen the input context, he/she has to select an EKD method chunk where (1) the situation type matches the input context's situation and (2) the intention of the method chunk matches the input context's intention. This selection is greatly facilitated by the use of a software tool. The remaining part of the reasoning loop associated to the application of the EKD decision making pattern is similar to what we presented for using the generic knowledge but the EKD engineer is more guided:

- the tactics is provided by the EKD method chunk. The EKD engineer is only required to instantiate the method chunk. He does not have himself to find the way of resolving the issue he is faced to, but he is just required to follow the predefined resolution approach provided by the chunk. If the method chunk is a choice context, the EKD engineer will have to instantiate the alternative contexts whereas he/she will do the same for the component contexts of a plan; he does not have anything to do at this stage if the context is executable.

- the enactment is identical to what we presented earlier. The EKD engineer makes decisions according to the predefined tactics, i.e. selecting the most appropriate alternative based on the arguments provided by the method chunk (again the arguments are predefined and do not need to be formulated by the engineer) in case of a choice context; selecting the adequate path in the precedence graph to execute a plan context and performing the action(s) of an executable context to modify the design product accordingly to the decision made.

We use a *matrix presentation* to overview the collection of chunks included in the EKD knowledge library. The *columns* of the matrix are *intentions* which arise during the EKD process, the *rows* of the matrix are techniques used in the guidelines and the *chunks* are the *matrix elements*.

There are 5 main intentions [14]: (1) Model the current enterprise state ; (2) Acquire goal ; (3) Operationalize goal ; (4) Generate design models ; (5) Validate design models. Some of them are decomposed into a number of

sub-intentions which are visualized in the matrix as sub-columns. For example, "Find goal", "Classify goal", "Prioritize goal", "Detect goal conflict", "Solve goal conflict" are sub-intentions of "Acquire goal".

The same technique can be used in different ways in different chunks. For example, brainstorming strategy is a technique which might be used for satisfycing the intention of "Detect goal conflict" and for "Solve goal conflict" as well. The SWOT analysis might be used for satisfycing the intention of "Analyze the context of cooperative design" and for "Argument alternative design models".

The essential benefit the EKD engineer gains in using EKD method chunks relates to guidance. By following the heuristical knowledge embedded in the method chunk, the engineer is constantly guided. Part of the solution he/she has to find is provided by the chunk. Suggestions are made on the alternative strategies he/she can follow, predefined arguments supporting or objecting to these strategies are provided, predefined plans he/she can use for reaching his/her decision are ready-made, he/she is told what to do next, etc..

### 4.3. Domain specific guidance

*EKD domain specific guidance* is based on *EKD domain specific knowledge*. *The domain specific knowledge* aims at providing guidance to EKD engineers for solving very well focused problems related to a specific domain. It is based on experience and suggests to reuse and possibly to adapt already tested solutions of the same domain. The use of domain specific knowledge within the EKD decision making pattern is very close to what has been presented in section 4.2 for EKD guidance. The difference lies in the fact that domain specific method chunks are defined at the instance level and therefore do not have to be instantiated while being used.

The reasoning loop starts with the retrieval of the domain specific chunk matching the input context. This can be done manually by the EKD engineer who browses through the knowledge library and looks for a chunk whose situation is the input context's situation and the intention is the input context's intention. It is more easily done with the use of the EKD tool environment which realizes the matching automatically. If there exists such a matching chunk, the EKD engineer can decide to use it.

The chunk provides the tactics for making the decision. Because domain specific chunks are defined at the instance level, there is no need for context instantiation (as for EKD method chunks). Then the engineer has just to enact the guideline provided.

The following section illustrates the use of our framework for modeling a cooperative design process and the cooperative business process resulting from it as its product.

## 5. Modeling cooperative design processes and the resulting cooperative business processes

Figure 13 illustrates a chunk for "goal reduction" as a tree of contexts in the EKD knowledge library (see section 4.2) guiding the reduction at different level of details. For the sak eof clarity, the name of the role is not mentioned, it is always « EKD engineer ».

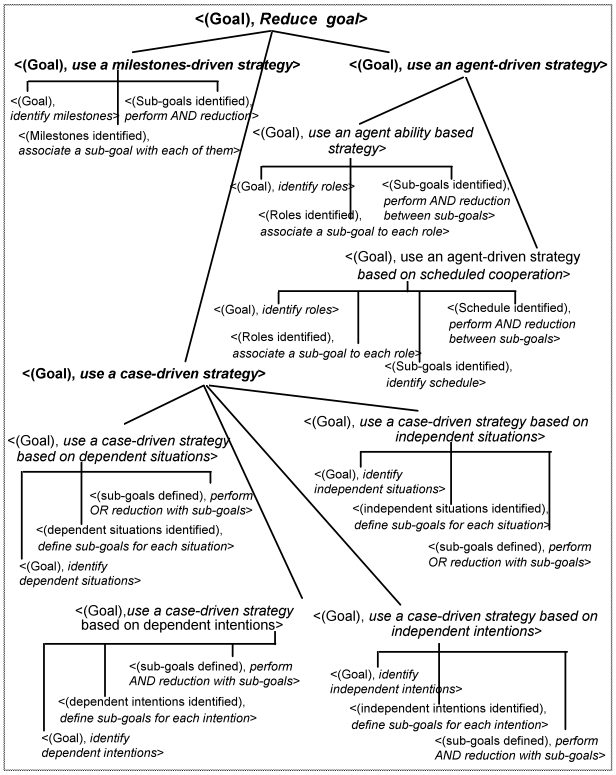


**Figure 13. The chunk for goal reduction**

At the first level, the chunk proposes three alternative strategies in order to reduce a goal [12]:

- a milestones-driven strategy (1),
- a case-driven strategy (2),
- an agent-driven strategy (3).

(1) The milestones-driven strategy is applicable when the satisfaction of the goal is associated with milestones. The goal reduction consists in identifying the milestones, the corresponding sub-goals and to perform an AND reduction with these sub-goals. This is modeled as a plan context.

(2) The case-driven strategy offers choices which correspond to its possible alternative sub-strategies. It is therefore, modeled as a choice context. This strategy is applicable when an exhaustive list of cases in which the goal has to hold can be identified and the work to be achieved is different for each case.

Each alternative is modeled as a plan context describing the goal reduction using :

- dependent situations; this reduction is based on the analysis of the organizational situation associated to the goal and should be selected when sub-situations dependent one from the other can be identified. These situations often correspond to possibly successive states of one of the objects involved in the situation.
- independent situations; in this case the situation as mentioned before can be decomposed into independent sub-situations, often exclusive states of the involved object.

- independent intentions; the reduction in this case is based on the goal itself and its target and on the identification of independent sub-intentions which are required for the goal to be satisfied. Another view of this alternative is to reason in terms of parallel work steps to achieve the goal.
- dependent intentions; as in the previous case, the reasoning supporting the reduction looks to the goal itself and identifies ordered work steps with dependent intentions.

(3) The agent-driven strategy is modeled as a choice context. Indeed, there are several alternative ways to apply this strategy.

Each alternative is a plan context describing the goal reduction using :
- scheduled cooperation; this strategy is applicable when contributions of the agents have to be structured according to a schedule so that each sub-goal concerns disjoint roles.
- agent abilities; this strategy is applicable when the goal can be decomposed into sub-goals each of them fitting some ability of a specific role without any need for scheduling.

Let us illustrate the use of the "Reduce goal" method chunk using the following air traffic control example :
<(Goal: "*No passenger should miss his/her connection*"), Reduce goal>

Figure 14 illustrates the goal reduction structure obtained by applying iteratively the method chunk "Reduce goal" for the input context <(Goal: "*No passenger should miss his/her connection*"), Reduce goal>. An indication in bold mentions the strategy whose selection led to the goal structure. Let us now comment the step by step enactment of the chunk leading to this goal reduction structure.

Assume first, that the EKD engineer selects the alternative <(Goal: "*No passenger should miss his/her connection*"), use a case-driven strategy> supported by the argument "one can identify an exhaustive list of cases in which the goal has to hold".

Secondly, the enactment of this choice context leads to the selection of the most adequate alternative which is a plan context <(Goal: "*No passenger should miss his/her connection*"), use independent situations> supported by the argument "the intention of this goal must be achieved in independent situations (corresponding to the exclusive states of landing)".

Thirdly, the enactment of the plan context <(Goal: "*No passenger should miss his/her connection*"), reduce the goal using independent situations> leads to reduce the goal using a choice context with two alternatives: *no passenger should miss his/her connection when the plane lands on time*, and *no passenger should miss his/her connection when the plane is late*.

These two new goals are to be reduced in turn.

The goal "*No passenger should miss his/her connection when plane on time*" is reduced using an agent-driven strategy based on scheduled cooperation; this is supported by the argument "contributions of the agents are structured according to some schedule so that each sub-goal concerns disjoint roles". The consequence is an AND reduction between the following sub-goals : *inform passenger during flight*, *set up ground staff support*, and *speed-up check-in*.

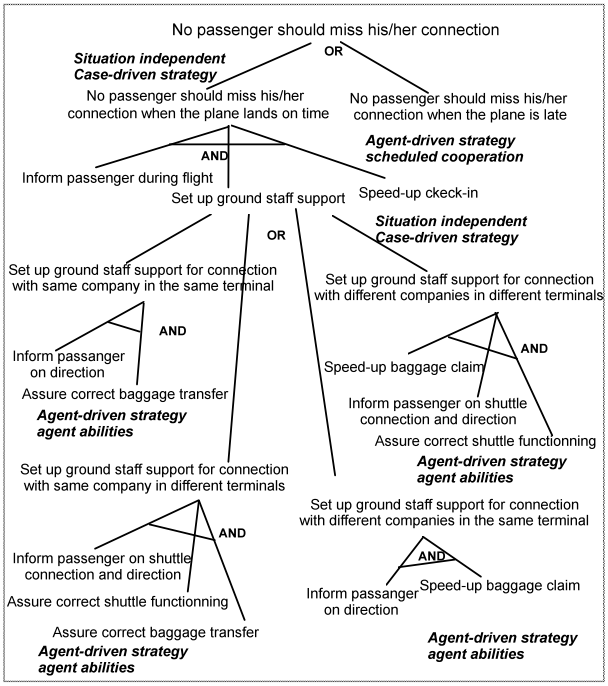


**Figure 14. The goal reduction structure for the goal « No passenger should miss his/her connection »**

Thus, there are three new goals to be reduced in turn.

The goal "*set up ground staff support*" is reduced using a case-driven strategy based on independent situations supported by the argument "the intention of this goal must be achieved in different and independent situations (exclusive states of landing)". The consequence is an OR reduction between the following sub-goals :
*set up ground staff support for connection with same company in the same terminal*
*set up ground staff support for connection with same company in different terminals*
*set up ground staff support for connection with different companies in the same terminal*
*set up ground staff support for connection with different companies in different terminals*

Then, four new goals to be reduced in turn. Each of them is reduced using an agent-driven strategy based on agent abilities.

The goal "*set up ground staff support for connection with same company in the same terminal*" is reduced in two sub-goals by an AND reduction : i*nform passenger on direction,* and *assure correct baggage transfer*.

The goal "*set up ground staff support for connection with same company in different terminals*" is reduced in three sub-goals by an AND reduction : i*nform passenger on shuttle connection and direction, assure correct shuttle functioning,* and *assure correct baggage transfer*.

The goal "*set up ground staff support for connection with different companies in the same terminal*" is reduced in two sub-goals by an AND reduction : i*nform passenger on direction* and *speed-up baggage claim*.

The goal "*set up ground staff support for connection with different companies in different terminals*" is reduced in three sub-goals by an AND reduction : s*peed-up*

*baggage claim*, *inform passenger on shuttle connection and direction*, and *assure correct shuttle functioning*.

This structure obtained by the reduction of the goal "*No passenger should miss his/her connection*" provides at the same time the high level description of a business process called "connection" in an airport. This process (see figure 15) has a well-known goal "*No passenger should miss his/her connection in this airport*". We do not include roles in the corresponding process model in order to not load too much the figure.

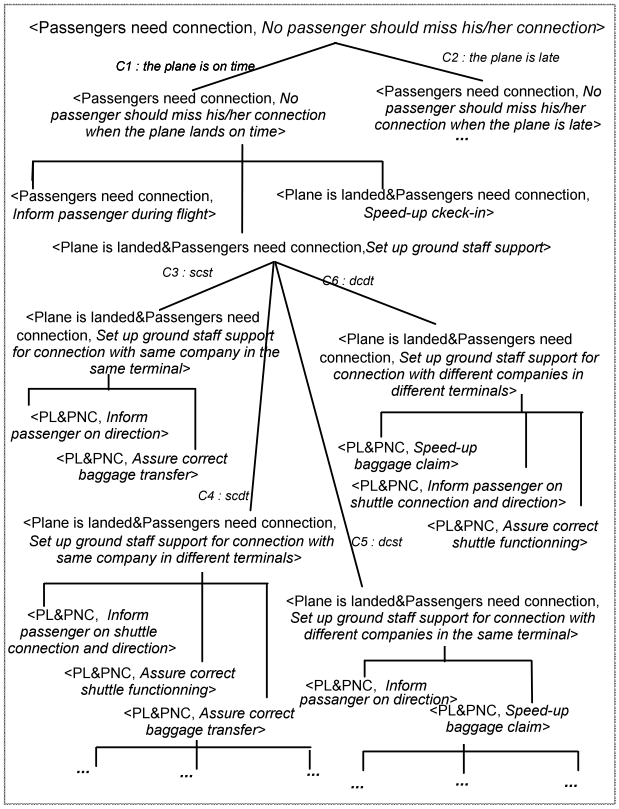


**Figure 15. The model of the cooperative business process "connection"**

## 6. Related work

Our proposal subsumes within it, at least six different approaches to process modeling:
(1) Activity-oriented approaches: Some activity-oriented process models (e.g. Waterfall [15], Spiral [16], Fountain [17]) are coarse-grained and aim to organize the software life-cycle. This is simply represented by a plan context. The precedence graph of the plan enables the definition of several alternative paths in this organization.
(2) In process centred environments (e.g. [18]), new activity based process models are fine-grained. The process meta-model allows through decomposition and refinement to model the process at different levels of granularity and therefore can handle the fine-coarse descriptions supported by software centred environments. However, unlike process models of these environments which concentrate on modeling the part of the process where activities are mainly executed by tools, the EKD process meta-model aims at offering means for supporting creative activities performed by human beings as well. It seems that purely activity centred models are not semantically powerful enough to achieve such a goal.
(3) Similarly to decision oriented process models (e.g. [19]), we look upon the cooperative design process as a decision making process. Like in IBIS [20], decisions are motivated by positions which are themselves supported or objected by arguments. The concepts of arguments and choice criteria we propose are borrowed to this approach. We extend the concept of decision to the one of context by relating decisions to the situations in which they are made. Further more, in DAIDA [19], the only mechanism proposed to describe a process model is by decision decomposition. The alternative ways of fulfilling a decision cannot be modeled as choice contexts do.
(4) In workflow models (e.g.[21,22]), a production process is described as a sequence of activities (component processes) constrained with a control flow. However, component processes are modeled as black boxes. No direct action on a product is undertaken. In our approach, we organize the content of processes by ordering the component of processes into a dependency graph which allows to describe iteration, back track as well as parallelism between processes.
(5) In situated process modeling [23], processes are described as sets of chunks whose invocation is not fixed in advance but based on the specification status. In the EKD process, we extend the notion of EKD product in order to provide a number of strategies for guiding the EKD engineer. For instance, guidance can be based on past decisions.
(6) Finally, we can relate the classic AND/OR graphs for planning used in the artificial intelligence community [10] to hierarchies of contexts. The "AND graphs" are similar to plan contexts whereas the "OR graphs" are similar to choice contexts. We enrich these graphs with the concepts of arguments and choice criteria and therefore provide a means for guiding the requirements engineer to make the appropriate decision.

## 7. Conclusion

In order to deal with a wide range of cooperative processes, we proposed a single process meta-model which provides the structuredness of the predefined models and the flexibility of ill-structured processes. The cooperative process meta-model allows:
• to represent cooperative design processes,
• to represent cooperative business processes,
• to improve their modeling by introducing heuristics that facilitate knowledge expression,
• to integrate conversations between actors into modeling and to make new goals emerge.

The EKD decision making pattern is a *reasoning mechanism* supporting decision making by providing a *set of predefined concepts*, a *library of guidelines* and a *set of predefined rules*. The concepts identify the elements supporting the reasoning; the two key concepts are the ones of context and product. The rules play a double role. First, they help in the retrieval of the appropriated

guideline from the library supporting decision making at that particular stage of the process, i.e. in the current situation at hand. Second, rules are used to guide the decision making according to the guideline.

The decision making pattern is tailored to provide guidance in all cases. In some cases, the pattern offers a domain specific guidance. This happens when the library contains knowledge about the domain of the project which matches the current context of work.

The library contains EKD specific guidelines which are tailored to the way the EKD approach suggests to work with the different EKD models. Such guidelines suggest, for instance, different techniques for supporting the emergence of goals, the operationalization of goals, the classification of goals etc.. These guidelines are independent of any particular domain but are based on EKD method knowledge.

Finally, if none of the two previous types of guidelines matches the current context of work, the generic guideline may operate. It is a generic rule for supporting decision making in cooperative design processes when neither EKD specific guidelines nor domain specific guidelines apply. Clearly, the more specific the guidance provided is, the more efficient it is. However, the generic guideline, by offering a general frame for decision making, makes the EKD process entirely based on guidance.

## References


[1] C. Rolland, et Al., "*An Approach for Defining Ways-of-Working*", Information Systems Journal, Vol. 20, No 4,1995.

[2] C. Rolland, "*Understanding and Guiding Requirements Engineering Processes*", invited talk, IFIP World Congress, Camberra, Australia, 1996

[3] M. Dowson, and C. Fernstrom, "*Towards requirements for Enactment Mechanisms*", Proc. of the th European Workshop on Software Process Technology, 1994.

[4] J. D. Wynekoop, N. L. Russo, "*System Development methodologies: unanswered questions and the research-practice gap*", Proc. of 14th ICIS (eds. J. I. DeGross, R. P. Bostrom, D. Robey), Orlando, USA, 1993, pp. 181-190.

[5] "*ELectrical Enterprise Knowledge for TRansforming Applications*", The ELEKTRA Project Programme, ELEKTRA consortium, 1996.

[6] S. Nurcan, C. Gnaho, and C. Rolland, "*Defining Ways-of-Working for Cooperative Work Processes*", Proc. of the First International Conference on Practical Aspects of Knowledge Management (PAKM) Workshop on Adaptive Workflow, Basel, Switzerland, October 30-31, 1996.

[7] S. Nurcan, and C. Rolland, "*Meta-modeling for cooperative processes*", Proc. of the 7th Euro-Japanese Conf. on Information Modellling and Knowledge Bases, Toulouse, France, May 27-30, 1997.

[8] C. Rolland, S. Nurcan, and G. Grosz, "*Guiding the participative design process*", Asso. for Information Systems Americas Conf., Indianapolis, Indiana, Aug. 1997, pp. 922-924.

[9] M. Dowson, "*Consistency Maintenance in Process Sensitive Environments*", Proc. of the Process Sensitive SEE Architecture Workshop, Boulder, CO. September 1992.

[10] N. Nilsson, "*Problem Solving Method in Artificial Intelligence*", McGrawHill, 1971.

[11] C. Potts, "*A Generic Model for Representing Design Methods* ", Proc. 11th Int. Conf. on "Soft. Engineering", 1989.

[12] A. Dardenne, A.v. Lamsweerde, and S. Fickas, "*Goal-directed Requirements Acquisition*", Science of Computer Programming, Vol. 20, 1993, pp. 3-50.

[13] A. Anton, "*Goal-Based Requirements Analysis*", ICRE '96, IEEE, Colorado Springs, Colorado USA, 1996, pp. 136-144.

[14] C. Rolland, S. Nurcan, and G. Grosz, "*A way-of-working for change processes*", Int. Research Symposium: Effective Organisations, Dorset, UK, September 4-5, 1997, pp. 201-204.

[15] Royce W. W. : "*Managing the Development of Large Software Systems*"; Proc. IEEE WESCON 08/1970

[16] B. Boehm, "*A Spiral Model of Software Development and Enhancement*", IEEE Computer 21, 5, 1988.

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

[18] L. Jacherri, J. O. Larseon, R. Conradi, "*Software Process Modeling and Evolution in EPOS*", in Proc. of the 4th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'92), Capri, Italy, 1992, pp574-589.

[19] M. Jarke, J. Mylopoulos, J. W. Schmidt, Y. Vassiliou, "*DAIDA - An Environment for Evolving Information Systems*"; ACM Trans. on Information Systems, Vol. 10, No. 1, 1992.

[20] Conklin, E.J. and Begeman, M. : *gIBIS: A Hypertext Tool for Exploratory Policy Discussion*, ACM Transactions on Office Information Systems, Vol. 6, No. 4, 1988, pp. 303-331.

[21] Winograd T. : "*A Language/Action Perspective on the Design of Cooperative Work*", In Human Computer Interaction, 3 (1°, pp 3-30, 1987-88.

[22] Gulla J.A., Lindland O.I. : "*Modeling Cooperative Work for Workflow Management*", in Proc. of the Int. Conf CAiSE94, Utrecht, The Netherlands, pp53-65, 1994.

[23] Rolland C., Cauvet C. : "*ALECSI : An Expert System for Requirements Engineering*", in "Advanced IS Engineering", R. Andersen, J Bubenko, A. Solvberg (Eds), Springer Verlag, 1991.