1ST INTERNATIONAL ENGINEERING DESIGN DEBATE (EDD'96)
GLASGOW, SEPTEMBER 23-24,1996

Architecture to handle Concurrent Engineering

Christophe Cointe, Nada Matta
INRIA (projet ACACIA)
2004 route des Lucioles BP. 93
06902 Sophia-Antipolis Cedex
e-mail: {ccointe,nmatta}@sophia.inria.fr

ABSTRACT
Concurrent Engineering is a hard task in which several designers collaborate together to build a system given requirements. To guide Concurrent Engineering task modelling, we define a generic model, in which individual designs are combined with coordination. Characteristics and nature of each sub-task are emphasized through this model. In order to guide Concurrent Engineering representation, we define a structure based on design statement representation. Each individual design and communicated information is represented as an alternative from which a solution (design statement with single alternative) is determined.

Key Words: Concurrent Engineering, Conflict, Knowledge Representation, Information Exchange, Task Modelling**.**

INTRODUCTION

In Concurrent Engineering (CE), several experts (designers, managers,... called participants in this paper) in different specialities, collaborate together in order to construct a system (also called artifact), given the customer's requirements. Each designer uses knowledge learned from his experience and also shared knowledge from his collaboration with the designer group. Conflicts can be revealed from disagreements between designers about proposed designs (called propositions) and needs [Klein,95]. Such conflicts are detected in CE and negotiation methods are used to solve them.

To handle CE, different types of knowledge will be distinguished and represented, e.g: Task organization, communication protocols, conflict management, knowledge sharing, task realization,...

Our main objective is to study CE task modelling and representation. We define generic models to help to model this task and specially the conflict management task. These models will provide generic components to be used (by a knowledge engineer[1]) as building blocks in CE and conflict management task modelling for a real application. We also determine a structure to support this task and to represent its particular components. This work is a part of the Genie[2] project, in which the CE task is analysed in order to provide guides for defining corporate memory.

We present in this paper a generic CE task model (I) in which individual propositions are combined with coordination. A structure to implement this model is then described (II), in which a structure is proposed to help propositions formulating and evaluating. Finally, we present how the structure can be used by designers to realize the CE task (III).

I. MODEL FOR CONCURRENT ENGINEERING TASK

The CE task model that we defined, is represented at knowledge level [Newell,82], in which knowledge characteristics and nature are described. There is also a distinction between this level and symbolic level in which implementation choices are made.

In order to define a generic CE task model, we have represented models, we studied from the literature, in a single formalism. This allows to analyse their characteristics and to reveal their respective contributions. The generic components that we defined are presented as an extension of the CommonKADS library [Breuker et al,94]. So, they are described with the CML[3] [Wielinga et al,92], the conceptual language offered by CommonKADS to represent expertise models. The generic models, provided by CommonKADS library for tasks such as diagnosis, design, etc., are described in CML. Therefore, it seemed natural to exploit CML to describe a generic model of the CE task.

I. 1. Concurrent Engineering task modelling research overview

A number of researchers have studied CE tasks and provided models of this task, like Ramesh et al. [Ramesh et al,94], Sycara [Sycara,91], Brazier et al. [Brazier et al,95] and Bond [Bond,90]. Bond's and Brazier et al's models offer a global view upon the main particularities of CE. We abstract as follows the characteristics emphasized in these models.

In this section, figures represent only data flows like in CommonKADS Task structures [Breuker et al,94]. Rectangles show tasks input/output, rectangles with rounded angles present tasks.

a) In Bond's model, two important notions appeared: Shared model and Private models. Each private model belongs to a participant of the concurrent engineering task and describes his knowledge. In fact, a private model is the expertise model of a participant (i.e. an explicitation of the model of his knowledge). These expertise models are private and they are not shared. However, the Shared model describes a shared model of the system to be designed.

First some participants use their private models to propose modifications in the artifact. These modifications are then communicated to the other participants. If there is no conflict, the Shared model and possibly private models are modified, and so on (Figure 1).

1.A Knowledge Engineer is able to acquire knowledge and define a model for real applications.
2.Genie project is a collaborative project between INRIA and DASSAULT AVIATION.
3.CML: (Conceptual Modelling Language) is defined in CommonKADS approach [Breuker et al,94].

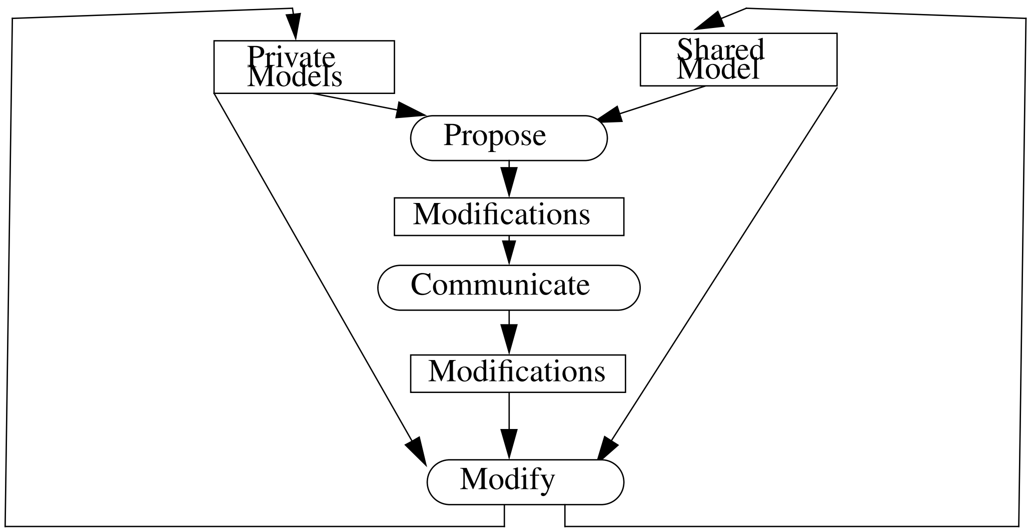

Figure 1. Bond's Concurrent Engineering Model.

b) In their model, Brazier et al. highlight the fact that in Concurrent Engineering, modifications can be made in requirements as well as in the artifact model (Figure 2). So, conflicts can be revealed and managed in requirements as well as in artifact model modifications.

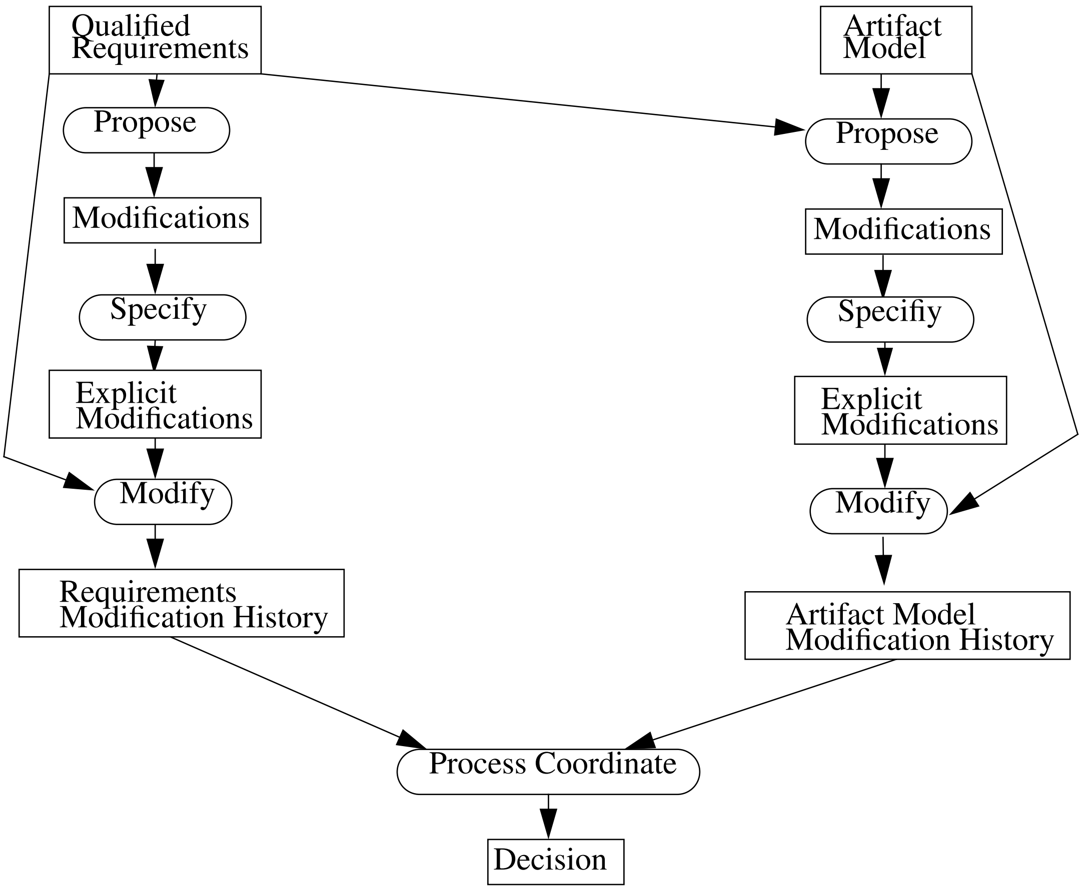

Figure 2. Brazier et al.'s Concurrent Engineering Model.

A coordination task (Figure 2) determines whether requirements or the artifact model must be modified in the next design step, and which part of requirements or the artifact model. This task decides also if the design process can be continued or stopped (if a deadlock cannot be solved, for example).

In each step in this process, conflicts can be detected. For example, a satisfaction conflict can appear when a participant proposes some modifications which do not satisfy requirements. Specifications of propositions may be also in conflict with other participants' objects description and needs. For more details see [Brazier et al,95].

I. 2. Generic Model for Concurrent Engineering task

After studying Concurrent engineering models, we propose a generic model for the concurrent engineering task, in which private propositions are combined with coordination (Figure3).

Three steps at three levels are performed in the CE task (Figure 3):

- Personal level («Design» task). At first, each designer generates some propositions to satisfy given requirements, based on his private knowledge (Private model). His task is identical to a design task. The generic design model represented in the CommonKADS library [Breuker et al,94] can guide the modelling of this task.

- Inter-personal level («Argument» task). To communicate his propositions to the group, a participant arguments them. Assumptions made in the first step, are used to determine arguments.

- Group level («Evaluate» task). The last step is an evaluation task. Propositions may not satisfy participants' needs and conflicts can appear. So, the principal subtask in this evaluation consists of detecting and solving conflicts.

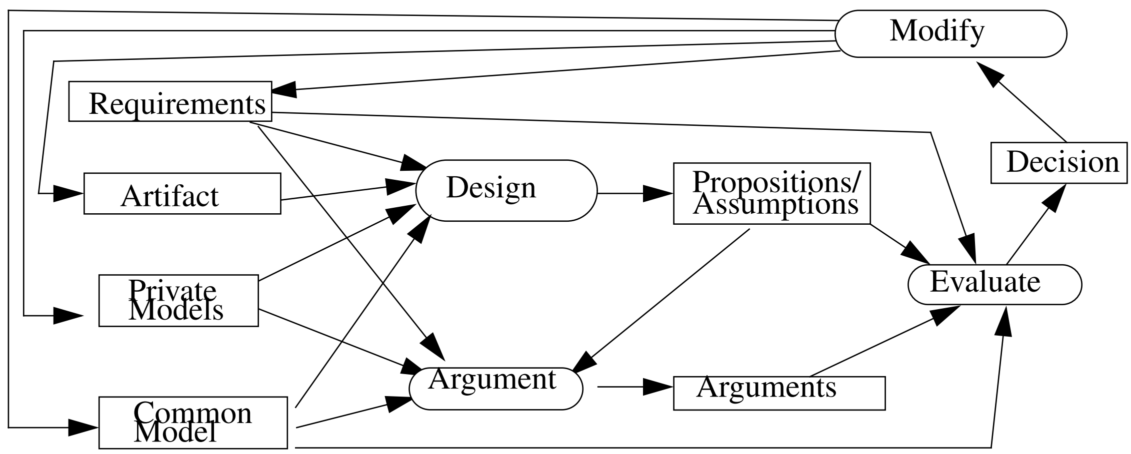

Figure 3. Generic Concurrent Engineering Model.

Propositions can concern the requirements or the artifact model. Decisions about which object (requirements or artifact model) can be modified in the next process cycle and in which part modifications can be made, are taken in the design control (a subtask of "Evaluate").

Sometimes, new knowledge can be learned from this experience. Then, some private models need to be modified by their respective owner. In the same manner, the Shared model can be reorganized and modified, because other shared knowledge can be explicited at each cycle.

In the next sections, we detail the tasks realized in the three levels.

I. 2. 1. Design Task

When a participant generates a proposition, his task is a design task. He must propose a design respecting related requirements. According to the design generic model proposed in CommonKADS library [Breuker et al,94], a participant first specifies the problem, then generates the corresponding subpart of design and evaluates if this subpart satisfies given requirements. Violated requirements cause modifications in the generated subpart of design or in the requirements.

In a CE task, a participant needs also to extract propositions which are more understandable by other participants than the corresponding subpart of design (Figure 4.). These propositions of design are then communicated to all the group of participants.

In CE, a Shared model is needed to facilitate cooperation between participants. The explicitation of shared knowledge enables a global view of the system on the one hand, and helps to determine interdependencies between participants on the other hand. The Artifact represents the current state of the system to be built (Figure 4.).

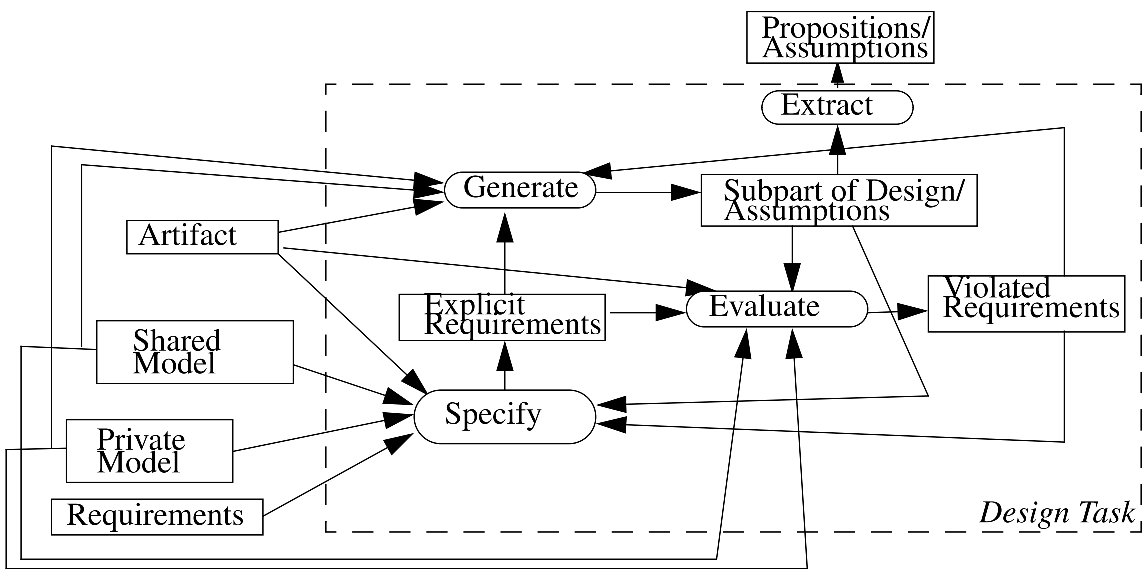

Figure 4. The «Design» task Model.

I. 2. 2. Argument Task

Argumentation is used in CE to defend a proposition and to persuade the group to choose it. Arguments must be explicit either to avoid conflicts if possible or to determine precisely their nature. Propositions are first justified with potential arguments which are then communicated to the group with the corresponding propositions. These arguments may not be sufficient to force the choice of propositions at the next level «Group level» and some of the arguments may be refuted by the group. So, a backtrack can be made and potential arguments are evaluated considering eventual conflict and refuted arguments (Figure 5).

A number of methods are proposed to help a participant to argument his proposition. K. Sycara [Sycara,91] recommends Case-Based Reasoning (CBR) to define arguments by retrieving them from previous cases. She proposes also some strategies using general principles and heuristic rules to define arguments. B. Ramesh [Ramesh et al,94] favours interdependency sharing to detect potential differences in order to consider them in the definition of the arguments.

I. 2. 3. Evaluate Task

Propositions may not satisfy participants' needs and conflicts can appear. So, the principal subtask in this evaluation consists of detecting and solving conflicts. Another subtask of «Evaluate» consists of controlling the design process overall. The group decides which object (requirements or artifact model) can be modified in the next process cycle and in which part modifications can be made («Decision» in Figure 3).

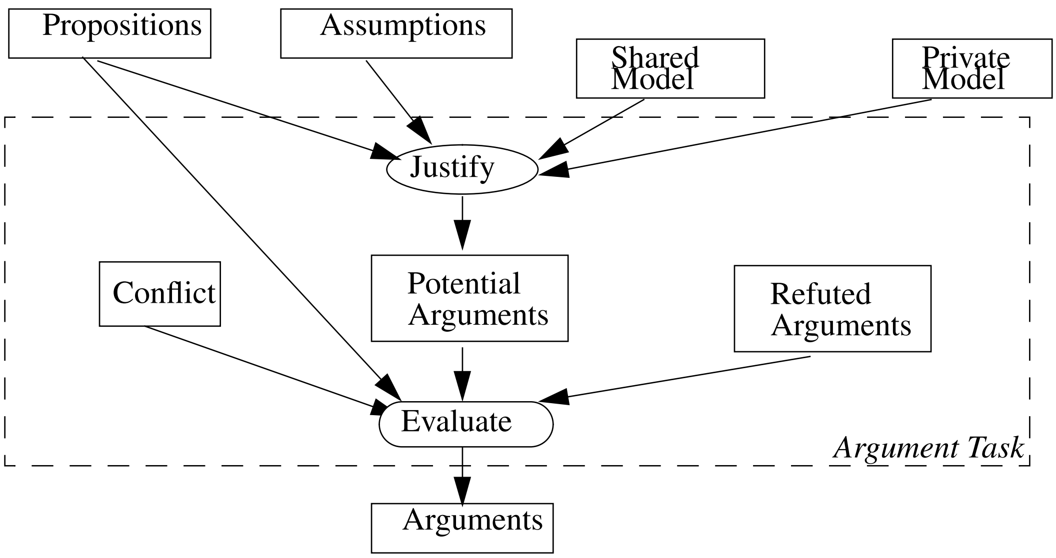

Figure 5. The «Argument» task Model.

Conflict management task consists of first detecting a conflict (its nature) and then solving it. As conflict detection is close to a diagnosis problem, diagnosis methods like those presented in [Benjamins,93] can be used to detect conflicts. Conflict symptoms can be first detected and contributors may be transformed in hypothesis to order to define the elements and the nature of the conflict.

Once detected, to solve a conflict, a solving method is selected and applied. The result is then evaluated to determine if the conflict is solved or not. In the last case, another method can be selected and applied and so on (Figure 6.).

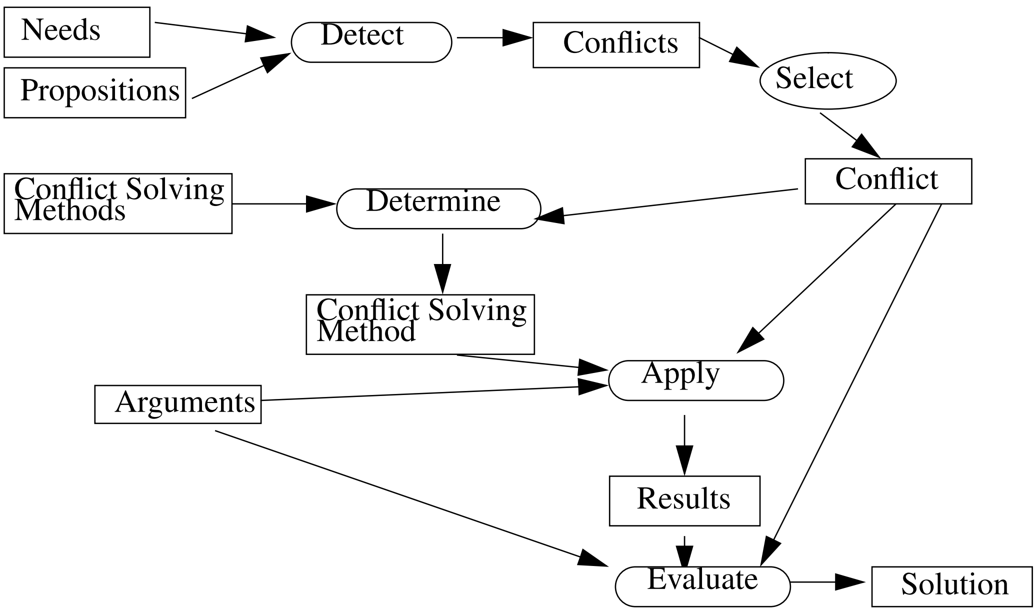

Figure 6. Conflict management task Model.

A large number of negotiation methods can be used to solve detected conflicts. Some of them like «Locate a consensus», where a consensus is located and a solution is then chosen, can be applied to solve mixed-motive conflict [Ramesh et al,94]. Another like «Introduction of a Third party solution», where a third party imposes a solution [Easterbrook et al,93] is defined to solve resource conflicts and also mixed-motive conflicts.

Other methods are proposed in some conditions. For example, to retrieve conflict solution by applying CBR [Sycara,91], first a number of past cases must be stored and second, at least one case similar to the current one must be selected. Another method proposes to restructure the problem, using a Goal Graph Search to add goals or to substitute rejected goals [Sycara,91]. This method needs goal graphs organized as influence trees, to be applied. Defining a «Counter-proposition» or modifying rejected proposition [Sycara,91] are also recommended.

Given this CE task model, a structure to support this task is determined. We describe its main elements in the next sections.

II. A STRUCTURE FOR REPRESENTED AND EVALUATED ALTERNATIVE PROPOSITIONS (SREAP)

In this part, we introduce a definition of a structure to support alternative propositions in CE task. First we define the context of this structure. Then, we describe its basic elements. Finally, we discuss different basic functionalities to use it.

II. 1. Context of the SREAP

Firstly, we have to define what kind of environment we deal with. And, more precisely, where and when SREAP can be used, given the CE task model described above.

Before presenting our structure, let us give an overview of the study in this area. We can note the classification proposed by A. Molina [Molina et al. 95] based on technological requirements to develop CE systems. Four types of requirements are needed:

- *Modelling methodologie*s: There is a need of modelling methodology, which tackles human behaviour modelling.

- *Computer-aided decision support*: Decision support software tools is necessary to guide designers activity in the whole CE life cycle.

- *Information architecture*: An open and distributed architecture is needed to guide task achievement and communication.

- *Framework for CE environment development*: Environment and tools to build a CE system are needed.

As we have said before, we suppose that the product is developed by a team of designers. All the designers are distributed among a network and the group of designers can evolve for the need of a particular design. So, SREAP can be viewed as a service of an Information architecture.

We can find in the literature examples of such architectures. We present shortly some of those architectures:

- MONET (Meeting On The NETwork) [Srinivas et al. 92] is a CERC (Concurrent Engineering Research Center, West Virginia University) development based on DICE (DARPA Initiative in Concurrent Engineering) architecture. MONET provides a support, and makes transparent the access and the exchange of multimedia data (text, audio, graphics, video and CAD data) and shared computer applications among people over a computer network. Those exchanges of multimedia data can be made in real-time, so MONET is a real-time multimedia conferencing system.

- MIT DICE (Distributed and Integrated environment for Computer-aided Engineering) [Sriram et al. 90] has been developed at the IESL (Intelligent Engineering Systems Laboratory) at the MIT (Massachusetts Institute of Technology). MIT DICE proposes a communication system among computer and users and a global database implemented as a blackboard.

- PACT (Palo Alto Collaborative Testbed) [Cutkosky et al. 93] is a concurrent engineering infrastructure developed at the Lockheed Palo Alto Research Laboratories. PACT is based on a multi-agents system. The agents encapsulate engineering tools and framework and communicate through a shared representation of the design.

Those information architectures tend to respond to the requirement of a good communication between the geographical-dispersed users (human or computer), a shared representation of the design, a design history service, tools and framework application integration services and multi-media user interface [Molina et al. 95].

In those information architectures, the object-oriented representation is very helpful to represent the design product and all that is related to it. We do the same for SREAP (i.e., we use the object paradigm for all our descriptions).

As it was notified in [Brazier et al. 95], we also describe the design of a product in three parts: the product, the requirements and the design process.

In our review of CE systems, we think there is a lack of guidelines for the CE task. Information architectures help to share, store and communicate design information, manage the history of the design process, integrate the design tools and provide an interface for the designers, but they do not help designer to choose the pertinent information to communicate to other designers. Those pertinent information will be evaluated by the designer team.

We will now describe the goal of SREAP. We want to provide support to the cooperative activity of designers. The designers will be helped in their choices between design propositions. Our goal is not to propose an automatic choice between propositions, but to help designers in their choice.

SREAP should be able to support the description of some parts of the product design and the notion explained in our CE's model showed in the part I. Our structure helps to formulate a design proposition and to evaluate it against others.

We will first present the basic principles of SREAP and, then, its use. SREAP will be implemented as a multi-agents system, but we do not describe in this article this implementation. We describe here a structure to support the different activities that those agents will have to help the designers in their cooperative activities.

III. DEALING WITH ALTERNATIVE PROPOSITIONS

As described in the CE model (I), several propositions are made by designers and communicated to the group. The group is composed of specialists of several special fields, and each designer has his specialized domain. Our structure can be viewed as a way to describe and represent the alternative propositions defined in order to evaluate them and to select a solution.

One of the goals of SREAP is to represent all the propositions in a similar structure. Each designer generates his private design and the propositions are extracted from sub-parts of this design and communicated to the other designers.

To illustrate the main concepts used and formalised in SREAP, a short example of designing a room is proposed. There are three designers: an architect, an electrician and a windows provider. Suppose the customer want to make a light room with a window. The requirements associated were the size (between 15 and 20 m^2) of the room and the orientation of the window (not at the north). This stage corresponds to a common design statement.

After this common design statement stage, each designer plans his work to respond to this design problem. Based on their private knowledge the designers communicate their proposition(s). For example, the architect designer proposes a $17m^2$ room and a north-east window orientation or a 18.5m2 room and a south window orientation, the electrician proposes electrical fittings for lamps for a room of 15 m^2 and the windows provider proposes a wood casement window for a room of 20 m^2. Each proposition corresponds to an alternative proposition of design for the size of the room.

Alternative propositions must be evaluated against the common design statements. In the evaluation conflicts can appear between alternative propositions. For this evaluation stage, design statements and alternative propositions must be formulated in a standard way, which is what SREAP proposes below.

III. 1. Design Statement and Alternative Proposition

A proposition represents a set of objects proposed by a designer. The proposition is based on the designer's private knowledge (Private Model) and can be evaluated by the other designers. A designer can make several propositions for a design. A proposition is a partial alternative description of a design. In SREAP we call ***alternative proposition*** the representation of a proposition. All alternative propositions corresponding to the same design are linked in SREAP with the associated ***design statement***.

A common decision (within the design team) must be made on what set of objects designers will propose modifications. This set of objects describes the static aspects (represented by objects which describe the product and the requirements) and the dynamic aspects (represented by objects which describe the design process) of the design. We call those objects, ***design statement aspects***.

Here we define what kind of objects are represented in design statement aspects. They can describe the product, the requirements and the design process.

- The description of the design product concerns all the elements which describe and define the product in an intermediate step of design. It is an incremental description, so during the design we have a partial description of the design and by adding product objects we refine and complement the description of the product. For example, the product describes a room which is composed of the window, the electrical fitting, the lamps and so on.

- The description of the requirements concerns all the constraints and elements, that the designer(s) wants to have influence on the product. Requirements are individual or global, they depend on the context of the design description, and they can be hard (indisputable) or express a preference: for example, preference between two orientations of the window in the room.

- The design process deals with the evolution of both descriptions of requirements and product. It characterises the relationship within and between those two descriptions. For example, the design process can provide the requirements which are associated to a particular subset description of design.

Based on those design statement aspects, and his design, the designer communicates an alternative proposition to the designer group. A set of objects composes the alternative proposition. We call this kind of set of objects, an ***alternative proposition description***. The alternative proposition description is represented by a type and the objects which refer to this type. The different types of alternative proposition description are described below.

III. 2. Types of alternative proposition description

An alternative proposition description is represented by the type of elements needed to describe an alternative proposition or to defend it. Each type of alternative proposition description is associated with set of objects. The alternative propositions are described in three general types: aspects, arguments and links.

- ***aspects:*** Alternative proposition description aspects are represented by objects which describe elements of design in order to refine and to complete the description of the design statement aspects. Aspects adding to the proposition can help to make the choice between alternatives or to define an element of a future design statement. Aspects can be static (i.e. describe the product and the requirements) or dynamic (i.e. describe the design process) as in the design statement aspects. Those aspects can be versions of objects representing design statement aspects or of other objects. We call the version of an object a modification or an addition of properties of an object.
- ***arguments***: The goal of arguments is to defend the alternative proposition. They are split in two parts: the facts and the functions. In the following, we illustrate arguments with examples retrieved from [Sycara 90].

Facts: Arguments are represented by objects. There are two kinds of facts, facts which are parameters of argument functions and facts which describe the qualification of the current proposition (for example the evaluate cost of the proposition). For example, facts can express:

- ***Goals***: Possible future states of design if the alternative proposition is chosen by the group (for example, used in the generation of threats or promises as arguments).

- ***Criteria:*** Elements used for qualification of the alternative proposition (for example, used in the appeal of a criteria corresponding to a specific theme or in the prediction of the future by extrapolating currents facts).

- ***Examples / Counter examples***: References to past cases, contradiction points of another alternative proposition (for example, used in definition of counter examples as arguments or in the appeal of exceptions retrieved from minor standard justification).

- ***Believes or Behaviours***: General uses (or habits) of designer group (for example, used in prevailing practice justification).

Functions: Functions describe (alternative) methods to calculate the interest of the proposition. For examples, methods can be:

- ***Case Based Reasoning***: Methods to express arguments based on past cases.

- ***Preference Analysis***: Methods to express an ordering between alternative propositions (for example, used in prevailing practice justification or in the appeal of self-interest argumentation).

- ***Rules/Heuristics***: Methods to express some general principles (for example, used in universal principles justification) or exceptions (for example, used in the appeal of exceptions retrieved from minor standard for the argumentation).

- ***links***: Always, an object (representing aspects or arguments) is linked with others. For example, a structural link (for example the link «A is sub-part of B», «C is composed of D,E,F»), a functional link (for example the link «A is equal to B», «C is greater than D»), a logical link (for example the link «A implies B»), or a composition of links. Thus, we can describe different types of links between objects and for each link, we can have the inverse link. So, we view link as a composition of objects defined later (which depend on another design decision) or already defined somewhere else.

III. 3. A structural view of the SREAP

SREAP can be viewed in structural way like a three levels tree: design statements, alternative propositions and alternative proposition descriptions. In the following, we call it SREAP tree, in which:

- design statement is the root of the tree,

- alternative propositions are sons of the design statement,

- alternative proposition descriptions are the leaves.

We associate a list of objects to a design statement. A design statement is labelled by a reference name or by a name chosen by the designers who have defined the design statement aspects.

An alternative proposition is a set of alternative proposition descriptions. An alternative proposition is labelled by a name selected by the designer who have made this proposition.

The alternative proposition description is composed of a type and a set of objects which defines and characterizes the alternative proposition. An alternative proposition description is labelled by the name of its type.

After defining the three different levels of the tree, we define below the relationship between sons in a tree. There are two kind of relations: alternative and associative. The alternative relation is between alternative proposition nodes and the associative is between alternative proposition descriptions leaves. The alternatives relations can be viewed as an «or» relation between alternative propositions and the associative relations as an «and» relation between alternative proposition descriptions.

An example of an SREAP tree is proposed in Figure 8: the design statement «Luminosity» defines the luminosity of a room, the object in design statement aspect is «Light». Two alternative propositions are associated to this design statement, labelled by «Good» and «Half-light». In the Good alternative proposition, we refer to a design statement named «Artificial_Luminosity» (Design Statement = Artificial_Luminosity), in this case we argue that the artificial light source provides a good luminosity (Fact:= Light.quality = «Good»). In the Half-light alternative proposition, the window must have a size between 1 and 2 m2 and must be defined in alternative «Half-light» of design statement «Window_Luminosity». In this alternative proposition the design statement object Light is refined.

III. 4. A representation view of the SREAP

For a more ergonomic use of SREAP we have associated a graphical representation of each elements of the tree. So, the different elements of SREAP are represented in Figure 7.

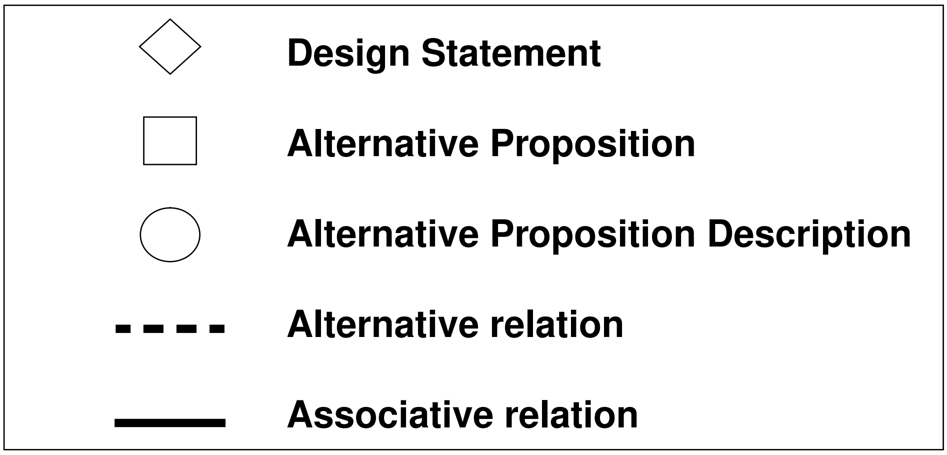

Figure 7. Basic graphical elements for SREAP.

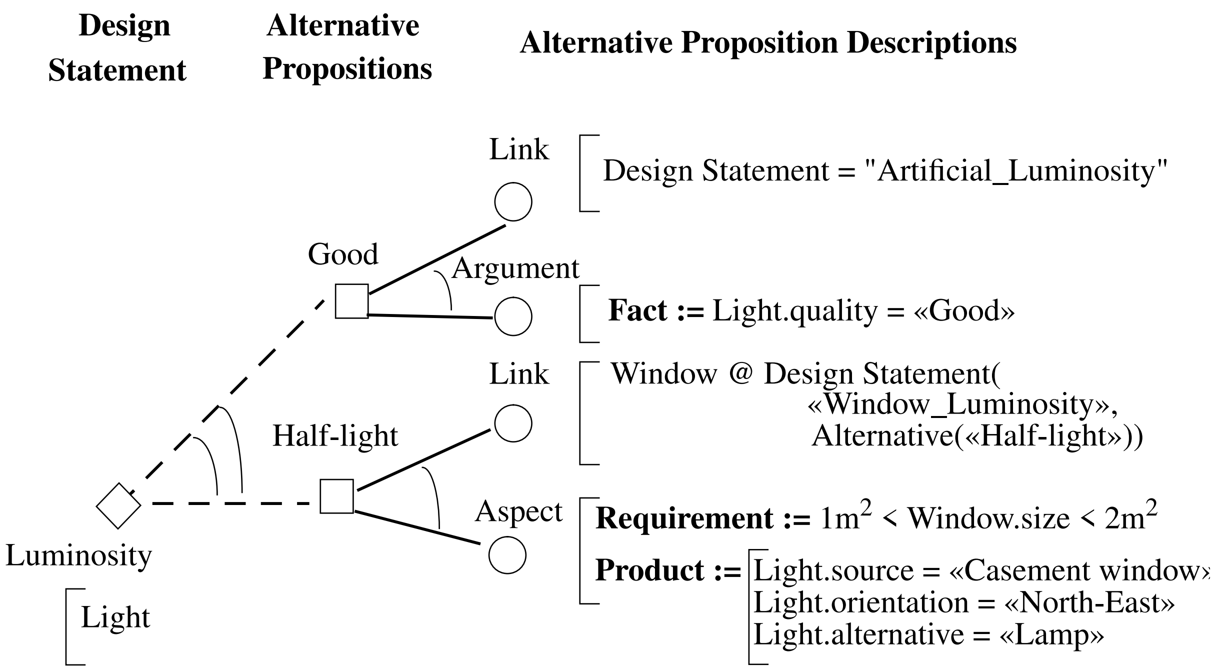

Figure 8. Example of SREAP tree.

IV. USE OF SREAP TO REALIZE THE CE TASK

We will define the meaning and the use of the SREAP elements (design statement, alternative proposition, alternative proposition description and alternative and associative relations) for a designer use. So, we describe the relationship between SREAP and the CE task.

As we said in part (II), SREAP can be viewed as a service of an information architecture, so we dispose of the other services of information architecture. In particular, designers using information architecture can read the design information. So, we will use those information architecture services and we will define the functionalities to manipulate the different elements of SREAP tree.

In our future implementation we will provide an aid to formulate and to send a proposition. This implementation will propose some services as user interface to write and to manipulate the elements of his sub-part of design, read the propositions, communication tools to support proposition and information sending between designers and help to manage conflicts between propositions. In this article, we focus on SREAP functionalities that are needed to describe CE task proposed in part (I). SREAP functionalities must help designers to manage, share, store and communicate alternative propositions and design statements. For this purpose, we define some basic functionalities Read, Write and Send. In the sections below, we show the use of those functions in a CE task. We only specify the general definition of those functions. We do not present in this paper how SREAP can support tasks realization and conflict management.

- **Read**: The elements of a SREAP tree must be accessible to a designer. The designer can: ask elements of private or shared SREAP tree, update elements of his private SREAP tree from the CE environment or his private resource, or read in a shared SREAP tree elements for his private SREAP tree.

- **Write**: A designer must be able to define elements of a SREAP tree. The designer can insert / delete elements in his private SREAP tree.

- **Send**: The elements of a SREAP tree must be exchanged between designers. The designer can: tell to other designers, or insert / delete in a shared SREAP tree elements of his private SREAP tree. Or elements of shared SREAP tree can be communicated to a designer.

In the following, we present the use of the SREAP in communication between designers in a CE task. For each element of a CE task description we define what are the elements of the SREAP used. To sum up the use of SREAP functionalities we provide figures for each CE task.

IV. 1. Design task

In a first step we have to help each designer to form his propositions. For each designer we provide SREAP tree to help him to formalize his proposition, and so on for all designers. Each sub-part of design corresponds to a private and partial part of design. The proposition will correspond to the description of an alternative proposition in a shared SREAP tree.

- ***Artifact, Private model*** and ***Shared model***: In SREAP, we are interested in the exchange of information between the designers, so all the parts which correspond to private or synthesis representation are not represented in SREAP. In particular, *Artifact*, *Private model* and *Shared model* are not represented in an extensive way in a SREAP tree, but elements of *Artifact*, *Private model* and *Shared model* can be used to describe elements of a proposition in a SREAP tree. In the rest of the document, we assimilate elements retrieved from the *Artifact*, the *Private model* and the *Shared model* which are represented in a SREAP tree as, respectively, *Artifact*, *Private model* and *Shared model*.

- ***Requirements***: Description of requirements, which are used in a proposition, are represented by design statement aspect requirements or by alternative proposition description aspects of shared SREAP tree.

- ***Explicit Requirements***: They are represented by alternative proposition description aspects of the actual designer's private SREAP tree, if they are used in a proposition.

- ***Violated requirements***: They are represented by design statement aspect requirements or in alternative proposition description aspects of one, or more, shared SREAP trees, if they are used in a proposition.

- ***Sub-part of Design/Assumptions***: Elements of *Sub-part of Design/Assumptions* are represented in an alternative proposition of an actual private SREAP tree, if they are used in a proposition.

- ***Propositions/Assumptions***: They are represented by alternative propositions of the actual private SREAP tree. *Proposition/Assumptions* are chosen to be sent to the other designer and they are represented in a shared SREAP tree.

- ***Specify***: The goal of the sub-part of design is to provide a more precise and explicit list of requirements from the set of requirements provided to the designers. Designers can be helped by using the shared SREAP tree to formulate the global requirements of the product to design in the design statement. Reading the definition of the global constraints and the common description of the current design helps to express the constraints in order to be more specific and compatible with the global and private (described in the private model) constraints, and description of the past and current description of the product (described respectively in the artefact and in Shared model). SREAP will help to realize this task by providing services of Read of the *requirements*, Read of the *Shared model and* Write *explicit requirements* (cf. Figure 9).

- ***Generate***: To satisfy the explicit requirements, the designer builds a *sub-part of design/ assumptions*. The designer, in his private work, could have several partial descriptions of his final sub-part of design. Or, he can have several sub-parts of design to propose to the other designers. So he can formulate a set of sub-parts of design as alternative propositions of his actual private SREAP tree. SREAP provides a common requirement (satisfied or violated) representation. So, the designer is helped to generate a *sub-part of Design* using some services provided by SREAP like: Read of the violated *requirements,* Read of the *Shared model* Read *explicit requirements* and Write the *sub-part of Design/Assumptions* (cf. Figure 9).

- ***Evaluate***: A designer has to evaluate if a *Sub-part of Design/Assumption* satisfies the *explicit requirements*. The unique representation of *Sub-part of Design* and *requirements* (in alternative proposition description aspects and design statement aspects) guides the designer to evaluate if the *Sub-part of Design* satisfy the *requirements* and to detected violated ones. SREAP will help to realize this task by providing services of Read of the *Shared model*, Read *Sub-part of Design*, Read *explicit requirements* and Write the *violated requirements* (cf. Figure 9).

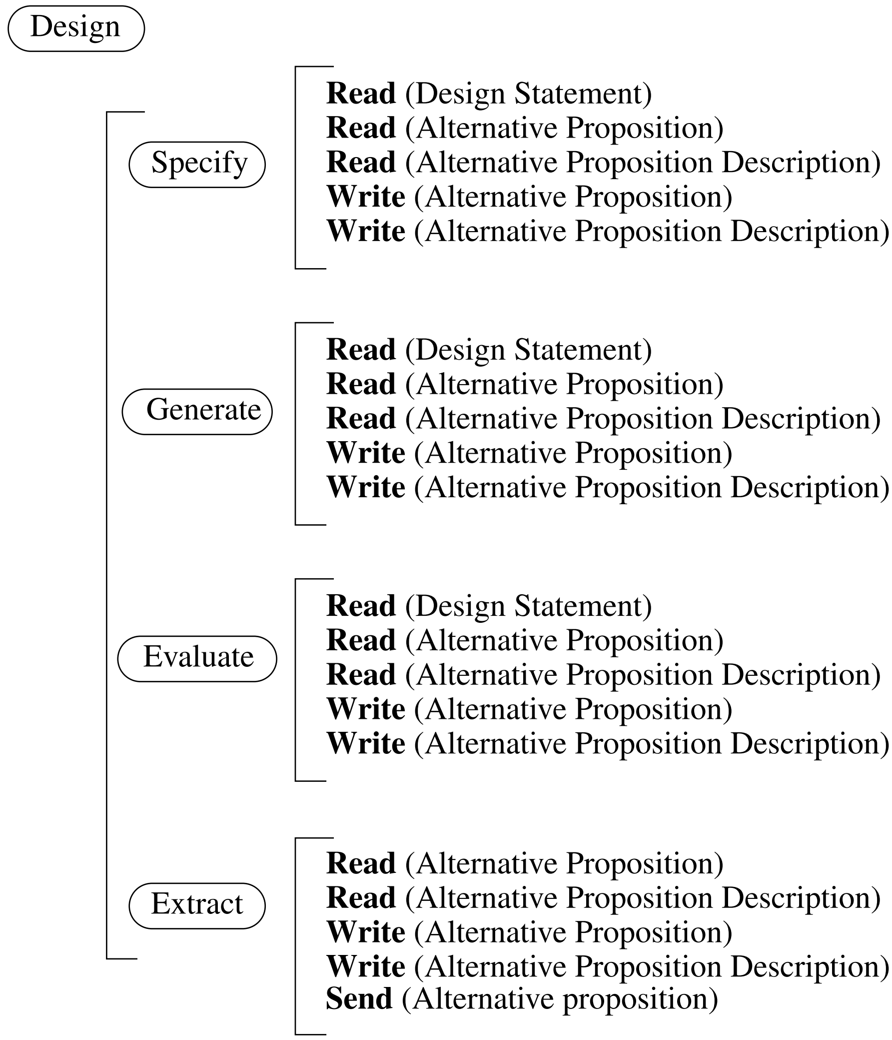

Figure 9. Design task.

- ***Extract***: When a designer has finished to formulate his *Sub-parts of Design*, he has to choose one of his alternative propositions (corresponding to *Sub-parts of Design*) of his actual private SREAP tree and to send a proposition to the group. From a *Sub-part of design* is extracted a proposition and is stored as an alternative proposition in shared SREAP tree. SREAP will help to realize this task by proposing elements which are potentially represented in a proposition as an alternative proposition of a shared SREAP tree. We provide services as: Read *Sub-part of Design*, Write *proposition/Assumptions* and Send of the *Proposition/Assumptions* (cf. Figure 9).

IV. 2. Argumentation task

Communication in the argumentation task (part I) can be handled as follows.

- ***Conflict***: The source of conflicts is represented within the alternative propositions of shared SREAP tree.

- ***Potential Arguments***: They are represented in alternative proposition description arguments of one or more alternative proposition nodes of the actual private SREAP tree.

- ***Refuted Arguments***: They are represented in alternative proposition description arguments of one or more alternative proposition nodes of the actual private SREAP tree.

- ***Arguments***: They are represented in alternative proposition description arguments of the alternative proposition node of the actual private SREAP tree.

- ***Justify***: Based on his *Assumptions* and *Proposition,* represented as alternative propositions in SREAP tree (respectively private and shared), a designer defines, and represents as alternative proposition description arguments, a number of potential arguments in order to justify his proposition. SREAP will help to realize this task by providing services of Read of the *Shared model*, Read *propositions*, Read *assumptions* and Write *potential arguments* (cf. Figure 10).

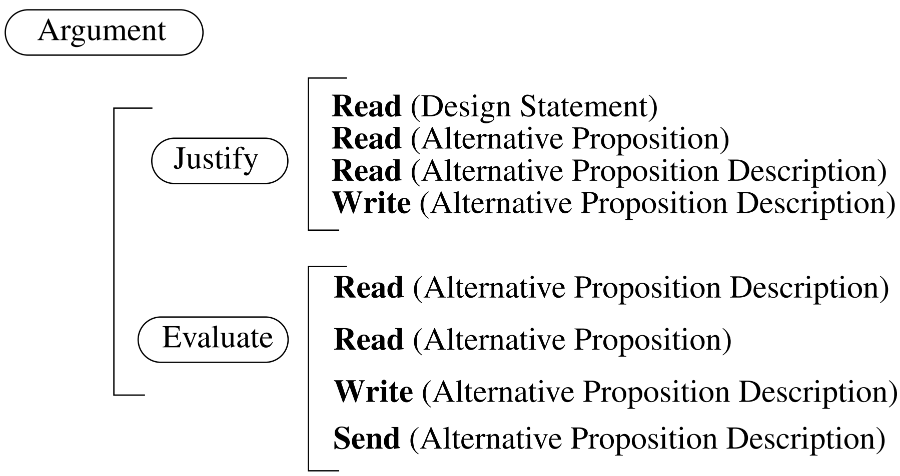

Figure 10. Argument task.

- ***Evaluate***: SREAP will help to realize this task by proposing *Potential or Refuted Arguments* which is selected or modified (represented as an alternative proposition description arguments of a shared SREAP tree) considering conflict sources described in alternative propositions. We provide services as: Read of the *conflict*, Read of the *propositions*, Read of *the potential arguments*, Read of the *Refuted Arguments*, Write the *Arguments* and Send the *arguments* (cf. Figure 10).

IV. 3. Evaluate Task

As described in part I, the evaluate task is decomposed in conflict management and in design process control.

IV. 3. 1. Conflict management task

- ***Needs***: They are design statement aspect requirements or alternative proposition description aspects requirements of shared SREAP tree.

Conflict Management

- Detect
  - **Read** (Alternative Proposition)
  - **Read** (Alternative Proposition Description)
  - **Write** (Alternative Proposition)
  - **Write** (Alternative Proposition Description)
- Select
  - **Read** (Alternative Proposition)
  - **Read** (Alternative Proposition Description)
  - **Write** (Alternative Proposition)
  - **Write** (Alternative Proposition Description)
- Determine
  - **Read** (Alternative Proposition)
  - **Read** (Alternative Proposition Description)
  - **Write** (Conflict solving method)
- Apply
  - **Read** (Conflict solving method)
  - **Read** (Alternative Proposition)
  - **Read** (Alternative Proposition Description)
  - **Write** (Alternative Proposition)
  - **Write** (Alternative Proposition Description)
- Justify
  - **Read** (Alternative Proposition)
  - **Read** (Alternative Proposition Description)
  - **Write** (Alternative Proposition)

Figure 11. Conflict management task.

- ***Conflict solving methods***: They are associated to a SREAP tree. They are not represented in the tree, because they are methods between propositions, and we represent only elements of propositions. In our future work, we plan to use those methods with SREAP and the way to represent those methods is not already defined.

- ***Results***: They are represented by alternative proposition nodes of a shared SREAP tree.

- ***Solution***: They are represented by alternative proposition nodes of a shared SREAP tree.

- ***Detect***: The goal of this task is to detect what element of which proposition violates a need represented as a shared requirement. SREAP represents both requirements and proposition in shared trees. So, the detection of conflict sources is guided; a conflict source is an alternative proposition description which does not satisfy the needs. SREAP will help to realize this task by providing services of Read the *needs*, Read *propositions* and Write the *conflicts* (cf. Figure 11).

- ***Select***: It is a choice of a *conflict*. SREAP will help to realize this task by providing services of Read of the *conflicts* and Write *conflict* (cf. Figure 11).

- ***Determine***: It is a choice of a *conflict solving method*. SREAP will help in this task by providing services of Read the *conflict* and Write a *Conflict solving method* (cf. Figure 11).

- ***Apply***: It is the application of the *conflict solving method*. The possibility to represent arguments in corresponding shared alternative proposition helps to use those arguments in order to help to solve conflict and define solution as results. SREAP will help to realize this task by providing services of Read a *Conflict solving method,* Read *conflict*, Read *arguments* and Write *results* (cf. Figure 11).

- ***Evaluate***: The goal of this task is to evaluate if the results had resolved the conflict selected. So, like the detect task, we search what element of results violate a need. SREAP represents the both requirements and results in shared trees. SREAP will help to realize this task by providing services of Read *results*, Read *conflict*, Read *arguments* and Write *solution* (cf. Figure 11).

IV. 3. 2. Control of design process task

- ***Decision***: The goal of this task is to express which objects (which describe product and requirements) will determine the design statement aspects in the next process cycle. Future design statement aspects will complete and refine the current solution by add of new alternative propositions. SREAP will help in this task by providing services of Read of the *Solution*, Read of the *Requirements*, Read of the *Shared model and* Write *Decision* (cf. Figure 12).

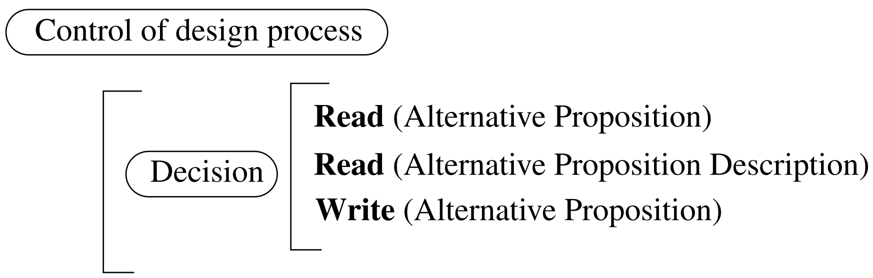

Figure 12. Control of design process task.

We have shown how to represent and share an alternative proposition with respect to the CE model described in part I. All individual actions supported by SREAP, are not static; the designer task can not be reduced to the use of a simple representation of an alternative proposition node in a SREAP tree. In order to help a designer, we have proposed a set of functions corresponding to actions that designers do in respect of our model propose in part I. SREAP propose functionality for load, store and send pertinent information. Those pertinent information are structured in a single feature in order to make conflict detection and solving easier.

CONCLUSION

Concurrent Engineering is a complex task in which individual competencies are confronted. The objective of our work is to provide guides to help the modelling of this task and to support its realization.

We proposed a generic model in which individual competencies and common ones are handled. Our model is general and can be used in different types of pragmatic design cases. The decomposition of a CE task in three level reflects the real-life realization of this task and allows to control methods used to perform it and to manage conflicts detected between designers. Our decomposition makes explicit the operation nature in CE, so a best management of the elements which implements those operations can be reached.

We also define a structure to support CE task representation. Knowledge management and representation are held in this structure.

This presentation of our work corresponds to the first step of the definition of SREAP. In this article we describe a way to represent pertinent information in a part of the CE task. In this representation we have provided a structure to help designers to formulate a proposition in order to be understood by others (we suppose that the designers use the same language and the same ontology, yet we do not manage the problem of misunderstanding due to the different technical vocabularies).

In order to help a collective choice between propositions, SREAP must support collective exchange and represent pertinent information. For the collective actions and representation, in respect of the CE task described in part I, we use the same structure and functionality: we represent only the pertinent information (sent proposition) and use the same set of SREAP functions. We focused on the management of disagreement between propositions. SREAP could help to define sources of conflict (as a part of a proposition) in the process of conflict management. Here we do not express the conflicts management methods nor the conflict types identification.

Our structure, SREAP, proposes help by providing guidelines which orient designers in their CE task through a structured communication and by providing a common representation of the deferents manipulated CE proposition elements. We feel there is a lack of such guides in information architecture.

Some parts of a CE task are not represented in SREAP because they are not pertinent to the exchange of information between designers. The past design descriptions are not represented in SREAP. To represent and read private and past information, we can use technologies used in information architecture.

The need to represent arguments [Sycara,90] generated with propositions concerning the product generated and the requirements manipulated [Brazier et al,95], pushed us to group these elements in a unique structure.

In order to guide CE and conflict management task modelling, we are going to define a generic conflict management methods library and to study conflict types as index of this library. We plan to implement this SREAP in a multi-agents system. We will study the possibility of detection of the different nature of potential conflicts in terms of SREAP and the application of a conflict solving method. A definition of a scenario of CE design and a validation by users will show the interests and problems of SREAP.

ACKNOWLEDGEMENTS

We deeply thank the French «Ministère de l'Enseignement Supérieur et de la Recherche» (Genie Project) that funded this research and the ModelAge Working Group (contract EP:8319) for their partial funding. We thank also Dr. R. Dieng and Dr. O. Corby for their remarks.

REFERENCES

[Benjamins,93] R.Benjamins, Problem Solving Methods of Diagnosis, *Thesis Univeriteit van Amsterdam, Amsterdam,* 1993.

[Bond,90] A.H. Bond, A Computational Model for organizations of cooperating intelligent agents, *Proceedings of the Conference on Office Information Systems*, Cambridge April, 1990.

[Boyera et al,95] S. Boyera, O. Corby, *Etat de l'art sur la multi-expertise, Projet Génie Thème 3*, Rapport de contrat GENIE, Thème 3, Sophia Antipolis, June 1995.

[Brazier et al,95] F.M.T. Brazier, P. H.G. Van Langen, J. Treur, Modelling conflict management in design: An explicit approach, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol.9, N.4, Cambridge University Press, USA 1995, pp.353-366.

[Breuker et al,94] *CommonKADS Library for expertise modelling Reusable problem solving components, Frontiers in Artificial Intelligence and Applications,* J. Breuker and W. Van de Velde (EDS), Amsterdam: IOS.Press 1994.

[Cutkosky et al. 93], Cutkosky MR, Engelmore RS, Fikes RE, Genesereth MR, Gruber TR, Mark WS, Tenenbaum JM and Weber JC, «PACT: An Experiment in Integrating Concurrent Engineering Systems», *IEEE Computer*, pp. 28-37, January 1993.

[Easterbrook et al,93] S. M. Easterbrook, E. E. Beck, J. S. Goodlet, L. Plowman, M. Sharples, C. C. Wood, A Survey of Empirical Studies of Conflict, *CSCW: Cooperation or Conflict?* S. Easterbrook (ed), Springer-Verlag 93.

[Klein,95] M. Klein, Conflict management as part of an integrated exception handling approach, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Vol.9, p.259-267, Cambridge University Press USA 1995.

[Molina et al. 95], Molina A, Al-Ashaab A, Ellis TIA, Young RIM and Bell R, A Review of Computer-Aided Simultaneous Engineering Systems, *In Research in Engineering Design*, vol. 7, number 1, 1995.

[Newell,82] A. Newell, The Knowledge level, *Artificial Intelligence Journal,* 19 (2), 1982.

[Ramesh et al,94] B. Ramesh, K.Sengupta, Managing Cognitive and Mixed-motive Conflicts in Concurrent Engineering, *Concurrent Engineering: Research and Applications* Vol.2, N.3, 1994, pp.223-236.

[Sriram et al 90], Sriram D., Logcher R., Wong A., and Ahnmed S., An Object-Oriented Framework for Collaborative Engineering Design, in D. Sriram, R. Logcher and S. Fukuda (eds.) *Computer-Aided Cooperative Product Development*, Springer pp 51-92, 1990.

[Srinivas et al. 92], Srinivas K, Reddy R, Babadi A., Kamana S. Kumar V. and Dai Z, CERC-TR-

RN-91-009 MONET: a Multi-media System for Conferencing and Application Sharing in Distributed Systems, *CERC Technical Report Series*, CERC-TR-RN-91-009, Morgantown, WV: CERC, West Virginia University, 1992.

[Sycara,90] K. P. Sycara, Persuasive Argumentation in Negotiation, *Theory and Decision,* 28: 203-242, Kluwer Academic Publishers, Netherlands, 1990.

[Sycara,91] K. P. Sycara, Cooperative Negotiation in Concurrent Engineering Design, *Computer aided cooperative product development. Proceedings of MIT-JSME workshop*. D. Sriram, R. Logcher, S. Fukuda (Eds), Cambridge, MA 1991.

[Wielinga et al,92] B. Wielinga, W. Van de Velde, G. Schreiber, H. Akkermans, *The CommonKADS Framework for Knowledge Modelling,* Report of ESPRIT KADS-II project p.5248: KADS-II/1.1/pp/UvA/35/1.0, September 1992.