

Modelling conflict management in design: an explicit approach

FRANCES M.T. BRAZIER, PIETER H.G. VAN LANGEN, AND JAN TREUR

Artificial Intelligence Group, Department of Mathematics and Computer Science,

Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands.

E-mail: {frances, langen, treur}@cs.vu.nl.

This paper focusses on how conflicts that arise during a design process and the management of conflicts can be modelled. A number of possible conflict types are distinguished and it is described how each of them can be detected during the design process, using an explicit meta-representation. Furthermore, it is shown how these conflict types can be analyzed and managed by means of strategic meta-knowledge about design processes.

Keywords: Conflict Management; Design Process; Formal Specification; Generic Task Model; (Strategic) Meta-Knowledge

1. INTRODUCTION

Conflicting interests, requirements, design possibilities: conflicts are inherent to design. Detecting and resolving conflicts requires knowledge of strategies that can be used for this purpose. In addition, however, explicit conflict knowledge is required. In design practice, the interests, designs, and requirements are often considered simultaneously and explored in parallel, and at some point resolved. This holds in particular for collaborative design efforts in which designers collectively design a complex artifact. The coordination of an individual designer's design processes requires extensive knowledge of conflict management strategies: when should conflicting options coexist, when should they be resolved. Design, therefore, includes not only detection and resolution, but also management of conflicts.

To model the strategic knowledge involved in conflict detection, resolution, and management, explicit knowledge of the design process itself and of the subtasks is required. Brazier, Van Langen, Ruttkay, and Treur (1994) present a framework for the development of intelligent design support systems, in which the conceptual design of design support systems is based on a generic task model of design and specified in a formal specification language. Within the framework, formal specifications can be automatically translated to executable prototype implementations.

Brazier, Van Langen, and Treur (1994) discuss the formal semantics of static and dynamic aspects depicted in this model of design. Reasoning patterns in design are highly dynamic and non-monotonic: detecting, resolving and managing conflicts is an essential ingredient in the

control of dynamic reasoning patterns. Strategic knowledge is essential and modelled explicitly within the framework, that is based on the notion of a meta-level architecture. The framework offers a rich (meta-) language to explicitly express strategic knowledge for conflict management. In this paper, the generic task model of design is used to structure a taxonomy of conflict types encountered during design. Detection, resolution, and management of these types of conflicts are formally specified within the framework.

The generic task model of design is introduced in Section 2. This generic task model can be seen as a model for a single designer's design process, but also as a model for the coordination of more than one design process. In Section 3 conflict types are distinguished in connection to the generic task model and illustrated by means of examples adopted from building design. Section 4 presents examples of the types of strategic knowledge required for the detection, resolution, and management of conflicts, together with examples of specifications. In Section 5 the results of this approach are discussed and future research is proposed.

2. A GENERIC TASK MODEL OF DESIGN

The generic task model of design resulted from the analysis and synthesis of a number of design tasks, including:

- curriculum design;
- elevator design (see Brazier, Van Langen, Treur, Wijngaards, and Willems (1995));
- chemical process design;
- financial portfolio design;
- environmental policy design (see Brumsen, Pannekeet, and Treur (1992));
- office plan design (see Geelen, Ruttkay, and Treur (1991));
- financial routing design (see Geelen and Kowalczyk (1992)); and
- emission inventory protocol design.

The generic task model of design incorporates the most generic aspects of design and is discussed below.

2.1. Outline of the generic task model

The generic task model of design outlined in Brazier, Van Langen, Ruttkay, and Treur (1994) assumes the existence of a problem statement and a more specific list of requirements and requirement qualifications, along with knowledge of the domain objects and knowledge of design strategies including conflict management strategies. The requirements, the knowledge concerning the necessary and desired properties of the design object (within a given context), clearly influence the process of design. Requirements can often be grouped into sets of

requirements that are directly related to a specific view of the object to be designed. Preferences exist between requirements: individual requirements may be hard requirements, as may sets of requirements, but they may also be (soft) requirements between which preference relations exist. This also holds for sets of requirements: preference relations often exist between sets of requirements. These preferences are expressed in a meta-language as qualifications of requirements.

The strategies that coordinate the different viewpoints, but that also determine which set of requirements to consider at a particular point (on its own or in parallel with another set of requirements), are clearly related to the state of the (partial) design object description, but are not the same. Requirements often change (e.g., they are decomposed into more specific ones) during the process of design, when they are evaluated against the (partial) description of a design object. Strategies that are employed for the creation of the design artifact itself, necessarily consider different types of knowledge. A description of the same design object made from two points of view will often differ. The design object description is most often partial: it is extended during design, on the basis of additional knowledge and integration of sub-solutions.

The distinction between requirement qualification manipulation and design object description manipulation is shown in Figure 1. Design process coordination, the design process evaluation subtask, is responsible for the coordination between those two manipulation subtasks. The result of the design task, a design object description, fulfills a set of requirements and complies with the fixed knowledge of the domain, known to the design object description manipulation component.

The formal specification of this model in a compositional architecture includes specification of the task structure and the control structures. Refinement of the generic task model for specific domains (see Brazier, Treur, Wijngaards, and Willems (1994)) results in a task model in which the following types of knowledge are specified:

- a decomposition of the task structure;
- a decomposition of the task's control structure;
- a decomposition of the (domain) knowledge;
- information exchange between subtasks; and
- delegation of roles.

Within this document examples of specifications of knowledge will be used to illustrate the way in which such knowledge is specified.

A short description of the model will be given below in three sections, each describing a main subtask of the model: manipulation of requirement qualification sets, manipulation of design object descriptions, and coordination of the design process.

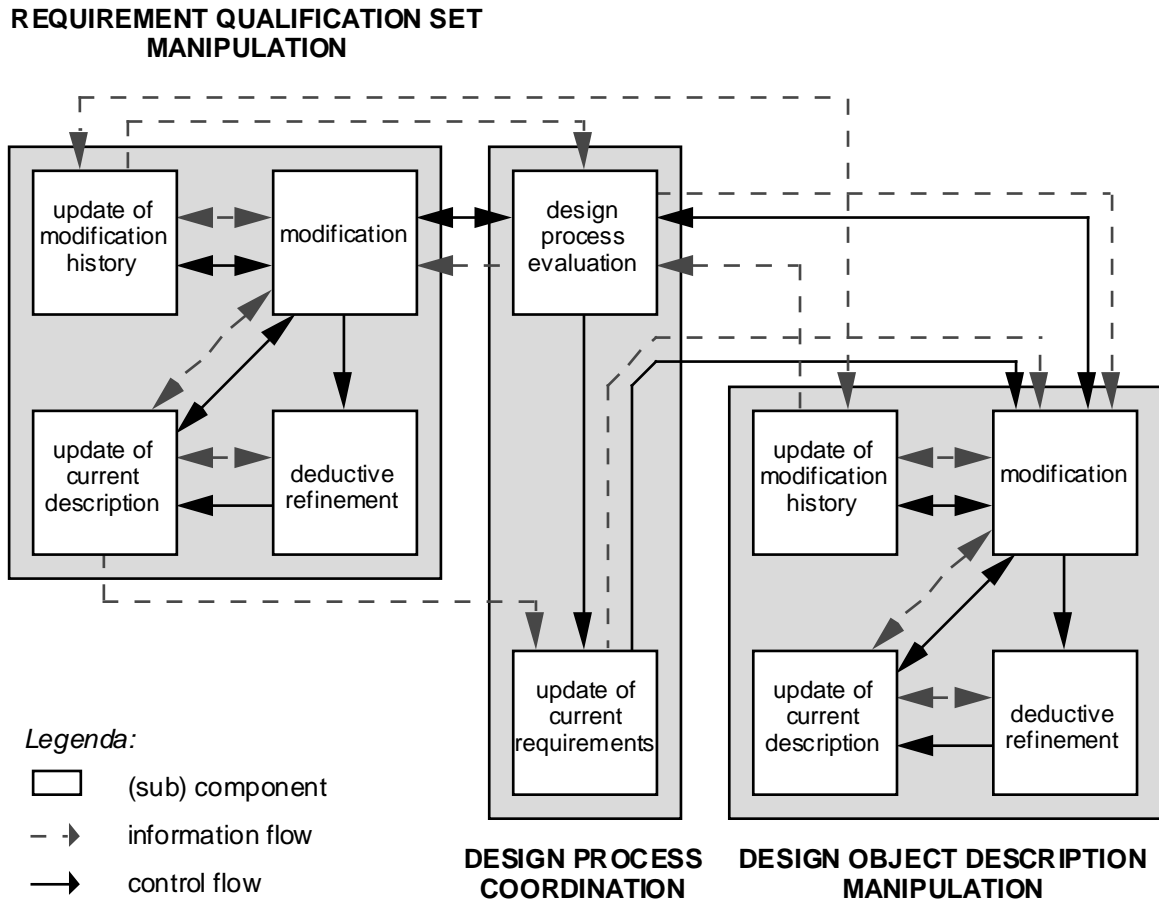


Fig 1. Generic task model of design.

2.2. Requirement qualification sets

To choose the most relevant subset of requirements out of the current set of requirements, entails consideration of the relevance/importance/strength of the individual requirements and the relations between requirements. Hard requirements, for example, must, by definition, hold for the final design object description but are not necessarily continually imposed during design. A set of related hard requirements (a view), however, may be grouped together during design. The choices made, the strategy chosen for the determination of the set of requirements to be considered, are based on knowledge of preferences between requirements. Furthermore, additional requirements may be specified by the participating agents (client, management, designer) with respect to evaluation criteria for the design object description.

Within the task model, the modification component determines the modification of the current requirement set, the deductive-refinement component determines which requirements follow directly from the resulting requirement set, the update-of-current-description component keeps track of the updated requirement qualification set, and the update-of-modification-history component keeps track of the requirement sets considered during the design process as a whole.

To determine which requirements to consider first, which to ignore, and which to modify or add (e.g., by decomposing requirements), possible modifications need to be considered. Explicit ranking criteria between preferred sets of requirements is sometimes available, but most often strategic knowledge is required. One global strategy often used to determine which modifications are most relevant is based on the distinction between the sources of a requirement: requirements based on user preferences are given higher priority than requirements based on default assumptions, which in turn are given higher priority than requirements which were the deductive consequence of previous requirements (similar to the approach described in Haroud, Boulanger, Gelle, and Smith (1994)).

In some domains the task of determining the most relevant modification to a set of requirements, entails determining which sets of requirements could possibly be considered, selecting a set on which to focus, determining which modifications are applicable, and selecting the most appropriate. This refinement of the requirement qualification set modification task is shown in Figure 2.

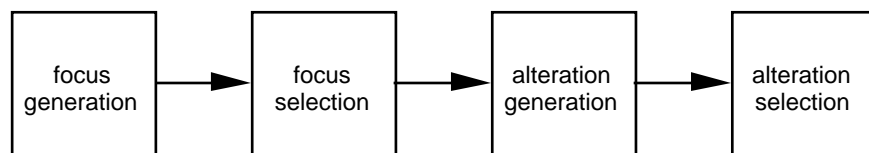


Fig. 2. Generation and selection of focus and alteration.

2.3. Design object descriptions

Creating a design object description on the basis of the requirements imposed, again entails determining a strategy for design object description construction. This process is similar to the requirement manipulation process, although the knowledge differs considerably. Again a possible strategy may be to focus on a given number of related elements of the design object (a view), on the basis of the related requirements, using the domain knowledge available to adapt the (partial) design object description, resulting in a new (partial) design object description. This process may be repeated for another view of the design object, and the resulting design object descriptions compared. The knowledge required for the comparison is part of the knowledge of the modification component.

Within the task model, the modification component determines which elements of the current design object description should be modified, the deductive-refinement component determines, on the basis of its domain knowledge which other elements follow directly from the modification, the update-of-current-description component keeps track of the updated design object description, and the update-of-modification-history component keeps track of the design object descriptions considered during the design process as a whole. The link between the

update-of-modification-history components of the requirement qualification set manipulation component and the design object description manipulation component is clearly defined.

In some domains the task of determining the most relevant design object description modification, entails determining on which part of the design object description the system could possibly focus and then selecting the most appropriate, determining which modifications are possible and again selecting the most appropriate modification. The refinement of the design object description modification task is comparable to the decomposition shown in Figure 2 for requirement qualification modification.

2.4. Coordination of the design process

The substructure for coordination of the design process is dedicated to determining whether to continue the design process or not, and if so, to continue with what subject. It consists of two components, named design-process-evaluation and update-of-current-requirements, and a number of interactions.

The component design-process-evaluation determines (from a strategic point of view) how the design process proceeds. For this purpose, it monitors the progress of the design process (by inspecting the modification histories with the help of the update-of-modification-history components), applies a coordination strategy to decide whether to continue or not and if so, where (either modification of the current requirement qualification set or modification of the current design object description), and informs the modification components about its strategic decision.

The task of update-of-current-requirements is to keep track of the current set of requirements which the current (partial) design object description has to satisfy. These requirements are assumed to be uniquely determined by the current requirement qualification set.

3. AN OVERVIEW OF CONFLICT TYPES

As discussed in the introduction, conflicts exist within each and every design process, both for an individual agent's design and for a collaborative design effort. The three components of the generic task model provide a basis for the distinction of three types of conflicts encountered in a single agent's design process. Conflicts are found within the design object description manipulation component, the requirement qualification set manipulation component, and the design process coordination component. Conflicts between agents, between an individual agent's design processes, are the fourth type of conflict distinguished. Each of these categories will be discussed below in more detail. A summary of the possible types of conflict that can occur in a design process is shown in Table 1. In the sequel, *design object description* will sometimes be abbreviated *DOD* and *requirement qualification set* *RQS*.

Table 1. Conflict types in a design process

Conflict Types	<u>Manipulation</u>		<u>Coordination</u>	
	<i>RQS</i>	<i>DOD</i>	<i>Process</i>	<i>Agent</i>
Intraview	•	•		
Interview	•	•		
Focus Generation	•	•		
Focus Selection	•	•		
Alteration Generation	•	•		
Alteration Selection	•	•		
Deductive Refinement	•	•		
Alteration	•	•		
Satisfaction			•	
Process Coordination Generation			•	•
Process Coordination Selection			•	•
Language				•
Beliefs				•
Goals				•
Responsibility				•
Cooperation Level				•

3.1. Conflicts within the requirement qualification manipulation component

Important (sub)components within the requirement qualification manipulation component are the components update-of-current-description, deductive-refinement, and modification. Again, conflicts can arise both within and between each of these components.

3.1.1. Conflicts between requirement qualifications

Requirement conflicts can occur within one view (*intraview requirement conflict*) or between two different views (*interview requirement conflict*). Moreover, conflicts are possible between externally generated requirement qualifications defining the problem statement and internally generated requirements that are of a heuristic nature (*external/internal conflicts*).

EXAMPLE 3.1.1.1:

If the customer wants a bungalow with a second-floor area of 50 m², this causes an intraview requirement conflict, because, by definition, a bungalow only has one floor. →

3.1.2. Requirement qualification set modification conflicts

Conflicts related to the modification (determination of foci and alterations) of requirement qualification sets can occur when determining the most appropriate focus on a part of the

requirement qualification set (when generating and selecting possible foci), or when determining the most appropriate alteration of the requirement qualifications set (when generating and selecting possibilities). As for conflicts between requirement qualifications, different views can play an important role. The conflict of having more than one alternative for foci is called a *requirement qualification set focus generation conflict*.

EXAMPLE 3.1.2.1:

A customer wants to have a dormer with a round roof, but the architect has made clear that, although the idea is appealing, it may cost too much, given the amount the customer is prepared to pay. The question is whether to take the dormer-related requirements into account right away or only in a later stage of the design process, when a better prediction about the costs can be given. This is an RQS focus generation conflict—the customer now may (temporarily) drop the idea of having a round roof for the dormer or may even abandon the idea of having a dormer altogether. →

The conflict of generating more than one alternative for modifications is called a generation conflict: a *requirement qualification set alteration generation conflict*.

EXAMPLE 3.1.2.2:

Suppose that a preliminary design of the house has been made, but without any indication of the position of the front door. Given the location where the house is to be built, the customer wants the front door of the house to be positioned on the west-side of the house (because then the front door can be easily accessed by visitors), while the architect, knowing that the prevailing wind comes from the west, wants the front door to be on the south-side. This is an RQS alteration generation conflict. →

The selection subtasks, to resolve generation and alteration conflicts, respectively, often make use of different selection criteria. Comparing these criteria provides a type of conflict: a *requirement qualification set focus selection conflict* and a *requirement qualification set alteration selection conflict*, respectively.

EXAMPLE 3.1.2.3:

Assume the same RQS focus generation conflict as described above. The selection of one of the mentioned options can be guided by the criterion of estimated cost or by the criterion of estimated light conditions. According to the first criterion, the best option for RQS focus alteration would be the focus *without* any requirements for a dormer at all. According to the second criterion, the best option would be the focus *with* requirements for a dormer (but without any for a round roof). This is an RQS focus alteration selection conflict. →

EXAMPLE 3.1.2.4:

Assume the same RQS alteration generation conflict as described above. The selection of one of the mentioned options can be guided by the criterion of accessibility or by the criterion of protection against out-door conditions. According to the first criterion, the best option for RQS alteration selection would be the one including the requirements for the front-door at the west-side of the house. According to the second criterion, the best option would be the one including the requirements for the front-door at the south-side. This is an RQS alteration selection conflict. →»

3.1.3. Conflicts between requirement qualifications and deductive refinement

Conflicts between the current requirement qualification set and deductive refinement knowledge also occur. A given requirement qualification set itself may not contain conflicting elements, but may conflict with elements that are implied, using knowledge of the deductive-refinement component. This type of conflict is called *deductive requirement qualification set refinement conflict*. Also here the views can play a role, in a similar way as above.

EXAMPLE 3.1.3.1:

Suppose the customer wants a two-story house with an area of 50 m² (geometric view), which may cost at most 90,000 guilders (economic view). Given that the usual height of a floor is 2.60 m, and that the cost of building a house is between 400 and 600 guilders per m³, this would mean that the house would cost at least 104,000 guilders. The geometric view and the economic view on requirements are in conflict, given the available deductive refinement knowledge. →»

3.1.4. Conflicts between requirements qualifications and possible alterations

Conflicts between requirements qualifications and possible alterations occur when alterations to a given requirement qualification set are generated (e.g., based on default knowledge), that are in conflict with other elements of the current requirement qualification set.

3.2. Conflicts within the design object description manipulation component

Important (sub)components within the design object description manipulation component are the components update-of-current-description, deductive-refinement, and modification. Again, conflicts can arise within components or between them.

3.2.1. Conflicts within a design object description

Conflicts between parameter values within a design object description, *design object description conflicts*, can occur in the update-of-current-description component. If different views of the same design object description are modelled, a design object description conflict

can be a conflict within one view (*intraview*), or a conflict between different views (*inter-view*).

EXAMPLE 3.2.1.1:

In the current design object description, the water supply system and the sewage discharge system occupy the same space in the bathroom. This is an interview design object description conflict. →»

3.2.2. Design object description modification conflicts

Conflicts related to the modification of the design object description can occur, for example, when trying to determine the most appropriate focus on a part of the design object description (when generating and selecting possible foci), or when trying to determine the most appropriate alteration of design object description elements (generated and selected). Different views can also play an important role. The conflict of having more than one alternative for foci is called a *design object description focus generation conflict*:

EXAMPLE 3.2.2.1:

Suppose, as earlier, that in the current design object description, the water supply system and the sewage discharge system occupy the same space in the bathroom. During modification at least two options exist: either the design process focusses on the water supply system, or it focusses on the sewage discharge system (the third option is to explore both foci in parallel). →»

The conflict of generating more than one alternative for alterations is called a *design object description alteration generation conflict*. An alteration can involve deletion (retraction) of some design object description elements, addition of other elements, or a combination of both (where deletion is assumed to precede addition).

EXAMPLE 3.2.2.2:

Suppose the customer wants the house to have a volume of approximately 260 m³. Then one possible design is that of a bungalow with an area of 100 m², while another possible design is that of a two-story house with an area of 50 m² (in both cases assuming a floor height of 2.60 m). Thus, there is a DOD alteration generation conflict. →»

The selection subtasks, meant to resolve generation and alteration conflicts, respectively, often make use of different selection criteria. Comparing these criteria again provides a type of conflict: a *design object description focus selection conflict* and a *design object description alteration selection conflict*, respectively.

EXAMPLE 3.2.2.3:

In the same DOD alteration generation conflict as described above. One of the criteria for selection of one of the mentioned options is average room area; another criterion is (warmth) insulation value. According to the first criterion, the best option for DOD alteration would be the bungalow (since the bungalow's floor area can be fully occupied as room area, whereas in the two-story house space must be reserved for stairs). According to the second criterion, the best option would be the two-storey house (since the sum of the wall area and the roof area of the two-story house is less than that of the bungalow). This is a DOD alteration selection conflict. →»

3.2.3. Conflicts between the design object description and deductive refinement

Conflicts between the current design object description and deductive refinement knowledge also occur. A given design object description may not contain conflicting elements, but may be in conflict with elements that are implied, using knowledge of the deductive-refinement component. This type of conflict is called a *deductive design object description refinement conflict*. Also here the views can play a role, in a similar way as above.

EXAMPLE 3.2.3.1:

Suppose that according to the design object description, there is a cold-water tap in the garage (water system view) but no central heating radiator (heating system view). In winter, the water system in a garage without heating may freeze. Thus, the water system view and the heating system view on the design object description are in conflict, given deductive refinement knowledge about what may happen in winter (interview conflict between design object description and deductive refinement). →»

EXAMPLE 3.2.3.2:

Suppose that according to the design, the furnace of the central heating system is connected to an *air* discharge channel. This is an intraview conflict between design object description and deductive refinement, because there is knowledge that the furnace may only be connected to a *gas* discharge channel. →»

3.2.4. Conflicts between the design object description and possible alterations

A conflict between design object description and possible alterations occurs when alterations to elements within the current design object description are generated (for instance, based on default knowledge) that are in conflict with other elements of the current design object description.

3.3. Design process coordination conflicts

Two types of conflict can be distinguished which relate to the coordination between the previous two components.

3.3.1. Satisfaction conflicts

Satisfaction conflicts are the conflicts which arise if a design object description does not satisfy one or more of the requirements in the update-of-current-requirements component. This violation of requirements can occur in one or more views.

EXAMPLE 3.3.1.1:

Suppose the customer wanted the house to have a volume of approximately 195 m³, but the architect has designed a house with a volume of 260 m³. Clearly, a satisfaction conflict exists between the customer's requirements and the architect's design object description. →»

3.3.2. Design process coordination conflicts

The component design process coordination can generate more than one alternative for continuation of the design process. Here a *design process coordination generation conflict* occurs. Selection of one of the alternatives can involve multiple selection criteria. This implies a *design process coordination selection conflict*.

EXAMPLE 3.3.2.1:

Suppose that the architect reports to have failed so far in producing a design object description that satisfies the customer's requirements, and that s/he is not sure whether a design object description exists which will satisfy all requirements. The architect wants to know if s/he has to stick to the customer's requirements or if the requirements can be relaxed a little bit before going on with trying to make a design. This is a design process coordination generation conflict. →»

3.4. Conflicts related to agent coordination

Given that an agent performs a design task, whether partial or complete, coordination between agents is essential with respect to conflicts. An example of an interagent conflict is if the languages used by both agents differ and are not explicitly linked to each other (*language conflicts*). Conflicts also occur with respect to the agents' beliefs, goals, responsibilities, levels of cooperation and the strategies which are used in a collaborative effort. These aspects, although recognized and modelled within specific applications (see for instance Brazier and Ruttkay (1993) and Brazier, Dunin-Kę plicz, Jennings and Treur (1995)), have yet to be included within the taxonomy of conflicts.

4. EXPLICIT REPRESENTATION AND MANAGEMENT OF CONFLICTS

In this section, the detection, resolution, and management of the various types of conflicts distinguished in Section 3 are discussed within the framework of the generic task model of design. The general approach is to use explicit representations of conflicts in a meta-level language. For example, `conflicting_values_for(p)`, where `p` is a parameter of the design object description. Conflicting values occur in the design object description as

```
value_of(p, c)
value_of(p, d)
```

for some distinct values (constants) `c` and `d`. By upward reflection, epistemic meta-information on the design object description is provided. In this case:

```
true_for_current_DOD(value_of(p, c))
true_for_current_DOD(value_of(p, d))
```

Based on this epistemic meta-information, it can be explicitly concluded by meta-level reasoning that there is a conflict of this type (conflict detection); for example, by deriving the meta-statement `conflicting_values_for(p)`, based on meta-knowledge such as:

```
if true_for_current_DOD(value_of(P: Parameters, V: Values))
  and true_for_current_DOD(value_of(P: Parameters, W: Values))
  and not equal(V: Values, W: Values)
then conflicting_values_for(P: Parameters);
```

Once this has been concluded it can be the starting point for meta-level reasoning to manage the conflict: by strategic reasoning to control the design process.

A number of examples of conflict types distinguished in Section 3 are used below to illustrate the general approach.

4.1. Requirement conflicts

The first example to be considered is the following intraview requirement conflict.

EXAMPLE 4.1.1:

If the customer wants a bungalow with a second-floor area of 50 m², this causes an intraview requirement conflict, because by definition a bungalow only has one floor. →

Note that information about requirements already is at a meta-level with respect to the object level of the design object description: imposing requirements requires reasoning about the

design object description. Detection of conflicts requires reasoning about the requirements and is therefore represented one meta-level higher: at the meta-meta-level. In addition to object-level components for reasoning about the domain of application, one component is needed at the meta-level for drawing deductive conclusions from the current requirement qualification set and one at the meta-meta-level for detection of conflicts in the current requirement qualification set. The example can be specified as follows:

Input for deductive requirement qualification set refinement

```
requirement(value_of(house-type, bungalow))
requirement(value_of(floor-area(floor(2)), 50-m2))
```

By deductive refinement a number of logically related requirements can be derived:

Deductive requirement qualification set refinement knowledge

```
if requirement(value_of(floor-area(floor(N: Values)), V: Values))
then requirement(value_of(minimum-number-of-floors, N: Values));

if requirement(value_of(house-type, bungalow))
then requirement(value_of(number-of-floors, 1));

if requirement(value_of(number-of-floors, N: Values))
then requirement(value_of(minimum-number-of-floors, N: Values));
```

Output of deductive requirement qualification set refinement

```
requirement(value_of(minimum-number-of-floors, 2))
requirement(value_of(number-of-floors, 1))
requirement(value_of(minimum-number-of-floors, 1))
```

These results can be reflected upwards to the RQS modification component where the subtask of requirement conflict detection can be performed:

Input for requirement conflict detection

```
true_for_current_RQS(requirement(value_of(house-type, bungalow)))
true_for_current_RQS(requirement(value_of(floor-area(floor(2)), 50-m2))
true_for_current_RQS(requirement(value_of(minimum-number-of-floors, 2)))
true_for_current_RQS(requirement(value_of(number-of-floors, 1)))
true_for_current_RQS(requirement(value_of(minimum-number-of-floors, 1)))
```

Requirement conflict detection knowledge

```
if true_for_current_RQS(requirement(value_of(P: Parameters, V: Values)))  
  and true_for_current_RQS(requirement(value_of(P: Parameters, W: Values)))  
  and not equal(V: Values, W: Values)  
then conflict_in(requirements_on(P: Parameters));
```

The resulting outcome of the requirement conflict detection subtask is an explicit representation of the conflict at the meta-meta-level:

Output of requirement conflict detection

```
conflict_in(requirements_on(minimum-number-of-floors))
```

An example strategy for managing this conflict can start by focussing on the requirements regarding the number of floors and by asking a user about the requirement conflict.

```
if conflict_in(requirements_on(P: Parameters))  
then to_be_focussed_on(P: Parameters);  
  
if to_be_focussed_on(P: Parameters)  
then to_ask_user_about(P: Parameters);
```

To ask the user is just one of the possible resolution strategies. Our approach enables specification of various alternative strategies, as shown in subsequent examples. Next we discuss an interview requirement conflict.

EXAMPLE 4.1.2:

Suppose the customer wants a two-story house with an area of 50 m² (geometric view) which may cost at most 90,000 guilders (economic view). Given that the usual height of a floor is 2.60 m, and that the cost of building a house is between 400 and 600 guilders per m³, this would mean that the house would cost at least 104,000 guilders. Thus, the geometric view and the economic view on requirements are in conflict. →

In addition to object-level components for reasoning about the domain of application, one component is needed at the meta-level for drawing deductive conclusions from the current requirement qualification set and one at the meta-meta-level for detecting conflicting views on the current requirement qualification set.

Input for deductive requirement refinement

```
requirement(value_of(number-of-floors, 2))
```

```

requirement(value_of(house-area, 50-m2))
requirement(value_of(maximum-cost, 90000-guilders))

```

Deductive requirement refinement knowledge

```

default_requirement(value_of(floor-height, 2.60-m));
default_requirement(value_of(minimum-volume-cost, 400-guilders-per-m3));
default_requirement(value_of(maximum-volume-cost, 600-guilders-per-m3));

```

```

if requirement(value_of(house-area, A: Values))
  and default_requirement(value_of(floor-height, H: Values))
  and equal(times(A: Values, H: Values), V: Values)
then requirement(value_of(floor-volume, V: Values));

```

```

if requirement(value_of(number-of-floors, N: Values))
  and requirement(value_of(floor-volume, V-floor: Values))
  and equal(times(N: Values, V-floor: Values), V-house: Values)
then requirement(value_of(house-volume, V-house: Values));

```

```

if requirement(value_of(house-volume, V: Values))
  and default_requirement(value_of(minimum-volume-cost, C-min: Values))
  and equal(times(V: Values, C-min: Values), C-total: Values)
then requirement(value_of(minimum-cost, C-total: Values));

```

```

if requirement(value_of(maximum-cost, C-max: Values))
  and not less_than(C-max: Values, C: Values)
then requirement(value_of(realizable-cost(C: Values), yes));

```

```

if requirement(value_of(maximum-cost, C-max: Values))
  and less_than(C-max: Values, C: Values)
then requirement(value_of(realizable-cost(C: Values), no));

```

```

if requirement(value_of(minimum-cost, C-min: Values))
  and not less_than(C: Values, C-min: Values)
then requirement(value_of(realizable-cost(C: Values), yes));

```

```

if requirement(value_of(minimum-cost, C-min: Values))
  and less_than(C: Values, C-min: Values)
then requirement(value_of(realizable-cost(C: Values), no));

```

Modelling Conflict Management in Design 17

Output of deductive requirement refinement

```
default_requirement(value_of(floor-height, 2.60-m))
requirement(value_of(floor-volume, 130-m3))
requirement(value_of(house-volume, 260-m3))
default_requirement(value_of(minimum-volume-cost, 400-guilders-per-m3))
requirement(value_of(minimum-cost, 104000-guilders))
requirement(value_of(realizable-cost(90000-guilders), yes))
requirement(value_of(realizable-cost(90000-guilders), no))
requirement(value_of(realizable-cost(104000-guilders), yes))
requirement(value_of(realizable-cost(104000-guilders), no))
```

Input for requirement conflict detection

```
true_in_view(economic-view, requirement(value_of(maximum-cost, 90000-guilders)))
true_in_view(economic-view, requirement(value_of(realizable-cost(90000-guilders), yes)))
true_in_view(economic-view, requirement(value_of(realizable-cost(104000-guilders), no)))
true_in_view(geometric-view, requirement(value_of(number-of-floors, 2)))
true_in_view(geometric-view, requirement(value_of(house-area, 50-m2)))
true_in_view(geometric-view, requirement(value_of(minimum-cost, 104000-guilders)))
true_in_view(geometric-view, requirement(value_of(realizable-cost(90000-guilders), no)))
true_in_view(geometric-view, requirement(value_of(realizable-cost(104000-guilders), yes)))
```

Requirement conflict detection knowledge

```
if true_in_view(V1: views, requirement(value_of(P: Parameters, V: Values)))
  and true_in_view(V2: views, requirement(value_of(P: Parameters, W: Values)))
  and not equal(V: Values, W: Values)
then conflict_between_views(V1: Views, V2: Views, P: Parameters);
```

Output of requirement conflict detection

```
conflict_between_views(economic-view, geometric-view, realizable-cost(90000-guilders))
conflict_between_views(geometric-view, economic-view, realizable-cost(104000-guilders))
```

Conflicts between externally generated requirements may be initially detected but left unresolved until a design object description has been created that can be evaluated with respect to these requirements. After evaluation, a resolution strategy may be applied, for instance by asking the user to choose between the requirements, based on what has been achieved in the design process.

4.2. Design object description conflicts

The following example shows the explicit representation of an interview conflict in the current design object description.

EXAMPLE 4.2.1:

Suppose that according to the design object description, there is a cold-water tap in the garage (water system view) but no central heating radiator (heating system view). In winter, the water system in a garage without heating may freeze. Thus, the water system view and the heating system view on the design object description are in conflict. →»

One component is needed at the object-level for drawing deductive conclusions from the current design object description and one at the meta-level for detecting conflicting views on the current design object description.

Input for deductive design object description refinement

```
value_of(contents(garage, cold-water-tap), yes)
value_of(contents(garage, central-heating-radiator), no)
```

Deductive design object description refinement knowledge

```
if value_of(contents(garage, central-heating-radiator), no)
then value_of(possible-temperature(garage, cold), yes);

if value_of(contents(garage, cold-water-tap), yes)
then value_of(possible-temperature(garage, cold), no);
```

Output of deductive design object description refinement

```
value_of(possible-temperature(garage, cold), yes)
value_of(possible-temperature(garage, cold), no)
```

Input for design object description conflict detection

```
true_in_view(water-system-view, value_of(contents(garage, cold-water-tap), yes))
true_in_view(heating-system-view, value_of(contents(garage, central-heating-radiator), no))
true_in_view(water-system-view, value_of(possible-temperature(garage, cold), no))
true_in_view(heating-system-view, value_of(possible-temperature(garage, cold), yes))
```

Design object description conflict detection knowledge

```
if true_in_view(V1: views, value_of(P: Parameters, V: Values))
and true_in_view(V2: views, value_of(P: Parameters, W: Values))
```

```
and not equal(V: Values, W: Values)
then conflict_between_views(V1: Views, V2: Views: P: Parameters);
```

Output of design object description conflict detection

```
conflict_between_views(
    water-system-view, heating-system-view, possible-temperature(garage, cold))
```

Next, the explicit detection and management of an intraview design object description conflict is specified.

EXAMPLE 4.2.2:

Suppose that according to the design, the furnace of the central heating system is connected to an *air* discharge channel. This is an intraview DOD conflict, because the furnace may only be connected to a *gas* discharge channel. →»

One component is needed at the object-level to draw deductive conclusions from the current design object description and one at the meta-level to detect conflicts in the current design object description.

Input for deductive design object description refinement

```
value_of(encompassing-system(furnace), central-heating-system)
value_of(channel-connection(furnace), air-discharge-channel)
```

Deductive design object description refinement knowledge

```
if value_of(encompassing-system(F: Furnaces), central-heating-system)
then value_of(channel-connection(F: Furnaces), gas-discharge-channel);
```

Output of deductive design object description refinement

```
value_of(channel-connection(furnace), gas-discharge-channel)
```

Input for design object description conflict detection

```
true_for_current_DOD(value_of(encompassing-system(furnace), central-heating-system))
true_for_current_DOD(value_of(channel-connection(furnace), air-discharge-channel))
true_for_current_DOD(value_of(channel-connection(furnace), gas-discharge-channel))
```

Design object description conflict detection knowledge

```
if true_for_current_DOD(value_of(P: Parameters, V: Values))
and true_for_current_DOD(value_of(P: Parameters, W: Values))
```

```

and not equal(V: Values, W: Values)
then conflicting_values_for(P: Parameters);

```

Output of design object description conflict detection

```

conflicting_values_for(channel-connection(furnace))

```

In this example one could use the strategy to propose to focus on retracting the value for the parameter *p*:

```

if conflicting_values_for(P: Parameters)
then retraction_to_be_focussed_on(P: Parameters);

```

Further meta-knowledge can be used to generate possible retractions of the values *air-discharge-channel* and *gas-discharge-channel*:

```

if retraction_to_be_focussed_on(P: Parameters)
and true_for_current_DOD(value_of(P: Parameters, W: Values))
then possible_alteration(delete_from_DOD(value_of(P: Parameters, W: Values)));

```

By generation of these possible retractions a new conflict may be created, this time a generation conflict: managing conflicts often implies translating a conflict of one type into a conflict of another type. The alteration generation conflict can be detected using the following generic knowledge:

```

if possible_alteration(add_to_DOD(value_of(P: Parameters, V: Values)))
and possible_alteration(add_to_DOD(value_of(P: Parameters, W: Values)))
and not equal(V: Values, W: Values)
then alteration_generation_conflict_on([P: Parameters]);

```

```

if possible_alteration(add_to_DOD(value_of(P: Parameters, V: Values)))
and possible_alteration(add_to_DOD(value_of(Q: Parameters, W: Values)))
and not equal(P: Parameters, Q: Parameters)
then alteration_generation_conflict_on([P: Parameters, Q: Parameters]);

```

```

if possible_alteration(delete_from_DOD(value_of(P: Parameters, V: Values)))
and possible_alteration(delete_from_DOD(value_of(P: Parameters, W: Values)))
and not equal(V: Values, W: Values)
then alteration_generation_conflict_on([P: Parameters]);

```

```
if possible_alteration(delete_from_DOD(value_of(P: Parameters, V: Values)))
  and possible_alteration(delete_from_DOD(value_of(Q: Parameters, W: Values)))
  and not equal(P: Parameters, Q: Parameters)
then alteration_generation_conflict_on([P: Parameters, Q: Parameters]);

if possible_alteration(add_to_DOD(value_of(P: Parameters, V: Values)))
  and possible_alteration(delete_from_DOD(value_of(Q: Parameters, W: Values)))
then alteration_generation_conflict_on([P: Parameters, Q: Parameters]);
```

To manage the alteration generation conflict, selection knowledge can be used. If this selection knowledge involves comparison of the generated alterations on different criteria, a new conflict is created: a selection conflict. To manage this, again explicit conflict management knowledge can be used.

After these selection and generation conflicts have been resolved, one of the possible alterations of the design object description can be selected and effectuated by retraction of the value of the channel-connection(furnace) parameter. Retraction is supported in the DESIRE framework by compositional truth maintenance capabilities; for details, see Pannekeet, Philipsen, and Treur (1991).

Note that the conflict management approach specified for design object descriptions can also be adapted for requirements qualification sets.

4.3. Coordination conflicts

The following example shows how to represent and manage a satisfaction conflict.

EXAMPLE 4.3.1:

Suppose the customer wanted the house to have a volume of approximately 195 m³, but the architect has designed a house with a volume of 260 m³. Clearly, a satisfaction conflict exists between the customer's requirements and the architect's design object description. →»

One component is needed at the object-level to draw deductive conclusions from the current design object description and one at the meta-level to check the current requirements against epistemic information about the current design object description.

Input for design object description deductive refinement

```
value_of(volume-of-house, 260-m3)
```

Design object description deductive refinement knowledge

```

if value_of(P: Parameters, V: Values)
  and not equal(V: Values, W: Values)
  then not value_of(P: Parameters, W: Values);

```

Output of design object description deductive refinement

```

not value_of(volume-of-house, 195-m3)

```

Input for requirement checking

```

current_requirement(value_of(volume-of-house, 195-m3))
false_for_current_DOD(value_of(volume-of-house, 195-m3))

```

Requirement checking knowledge

```

if current_requirement(F: Formulas)
  and false_for_current_DOD(F: Formulas)
  then violated(F: Formulas);

```

Output of requirement checking

```

violated(value_of(volume-of-house, 195-m3))

```

Here the upward reflection transforms object level information of the form

```

not value_of(volume-of-house, 195-m3)

```

into meta-level information of the form

```

false_for_current_DOD(value_of(volume-of-house, 195-m3))

```

By the requirement checking rule it can be derived that

```

violated(value_of(volume-of-house, 195-m3))

```

The conflict management strategy can be to further analyse (at the meta-level) the modifiable parameters of the design object description on which the violation depends (e.g., the width and length of the house) and then to determine on which of these parameters to focus, for example:

```

if violated(value_of(P: Parameters, V: Values))
  then in_violation(P: Parameters);

```

```
if depends_on(P: Parameters, Q: Parameters)
  and in_violation(P: Parameters)
  then to_be_focussed_on(Q: Parameters);
```

Note that the strategic choice here is the decision to explore possible alternatives for the design object description. If all possibilities have been explored, or if other reasons exist to assume that the chance of finding a design object description which meets the current requirements is almost nonexistent, the strategic choice would have been to explore possible alternatives for the requirements.

5. DISCUSSION AND CONCLUSIONS

Conflicts are inherent to design: design processes most often are based on conflicting requirements, interests, beliefs, goals, responsibilities, and domain knowledge. Conflict management entails not only detecting conflicts and exploring possibilities to resolve conflicts, but also coordinating design processes in which conflicting options are explored in parallel. Conflicts often evolve in the sense that the resolution of one conflict requires or results in the creation of a new conflict, which in turn, may (have to) be resolved immediately, or may be ignored for a while. In the end, however, all conflicts need to be resolved. The complex reasoning processes involved in this process are highly dynamic and non-monotonic. As conflicts are not pathological exceptions to be avoided but an essential characteristic of design, they need to be explicitly modelled during the design and development of decision support systems.

As shown in this paper the generic task model of design introduced in Brazier, Van Langen, Ruttkay and Treur (1994), specified in the framework DESIRE for formal specification of compositional architectures, is a framework within which an effective structure exists within which different types of conflicts within a single design process can be identified (for instance almost all conflicts distinguished in Oh and Sharp (1994) are covered by this classification). Some of these conflicts are implicit; for example, only by closely observing the design process will conflicts between the information on which reasoning about requirements is based and the information on which reasoning about the design object description is based, be discovered. However, to formally model conflict detection, resolution, and management, knowledge about conflict occurrence must be made explicit. This can be achieved by using meta-level strategic reasoning to guide the analysis, resolution, and management of conflicting information. The notion of meta-level architecture that defines the basic underlying structure of a compositional architecture plays a crucial role in the approach to modelling conflicts described in this paper. It offers an expressive language to specify conflict detection, management, and resolution strategies.

Formal semantics of the compositional approach to complex and dynamic reasoning processes as described can be found in formal models for the behaviour of the reasoning based on temporal logic; see Engelfriet and Treur (1994), Gavrila and Treur (1994), and Treur (1994). Current research addresses the possibilities for validation and verification of dynamic patterns of reasoning (see Treur and Willems (1994a), Treur and Willems (1994b)), such as those found in design processes.

To model conflicts between individual design processes (agents), a new design process is required: namely to design the collaboration between the individual processes. The requirements imposed upon collaboration between these processes must be made explicit and knowledge about the characteristics, competencies, responsibilities, beliefs, goals, must be used to guide conflicting design processes (for some of these aspects, see Petrie, Cutkosky, Weber and Conru (1994)). Current research addresses the use of the generic task model to model such multi-agent systems and to formally specify the conflicts which arise: a promising endeavour.

REFERENCES

- Brazier, F.M.T., Dunin-Kę plicz, B., Jennings, N.R., & Treur, J. (1995). Formal specification of multi-agent systems: a real-world case. *Proc. First Int. Conf. on Multi-Agent Systems, ICMAS '95*.
- Brazier, F.M.T., Langen, P.H.G. van, Ruttkay, Zs., & Treur, J. (1994). On formal specification of design tasks. *Proc. Artificial Intelligence in Design '94*, (Gero, J.S., and Sudweeks, F., Eds.), pp. 535-552. Kluwer Academic Publishers, Dordrecht.
- Brazier, F.M.T., Langen, P.H.G. van, & Treur, J. (1994). A logical theory of design processes. *Proc. AID '94 Workshop Nature and Role of Theory in AI in Design Research*, (Smithers, T., Ed.).
- Brazier, F.M.T., Langen, P.H.G. van, Treur, J., Wijngaards, N.J.E., & Willems, M. (1995). Modelling a design task in DESIRE: the VT example. *Journal of Human Computer Interaction*, Special Issue on Sisyphus (to appear).
- Brazier, F.M.T., & Ruttkay, Zs. (1993). Modelling collective user satisfaction. *Proc. HCI International '93*, pp. 672-677. Elsevier Science Publishers, Amsterdam.
- Brazier, F.M.T., Treur, J., Wijngaards, N.J.E., & Willems, M. (1994). A formalisation of hierarchical task decomposition. *Proc. ECAI '94 Workshop Formal Specification Methods for Knowledge-Based Systems*, (Aben, M., Fensel, D., Harmelen, F. van, and Willems, M., Eds.), 97-112.
- Brumsen, H.A., Pannekeet, J.H.M., & Treur, J. (1992). A compositional knowledge-based architecture modelling process aspects of design tasks. *Proc. Twelfth Int. Conf. Artificial Intelligence, Expert systems and Natural Language, Avignon '92*, Vol. 1, pp. 283-293. EC2, Nanterre.
- Engelfriet, J., & Treur, J. (1994). Temporal Theories of Reasoning. *Logics in Artificial Intelligence: Proc. Fourth Europ. Workshop Logics in Artificial Intelligence, JELIA '94*, (MacNish, C., Pearce, D., and Pereira, L.M.,

- Eds.), pp. 279-299. Springer-Verlag, Heidelberg. Extended version to appear in *Journal of Applied Non-classical Logic*, special issue with selected papers from JELIA '94.
- Gavrila, I.S., & Treur, J. (1994). A formal model for the dynamics of compositional reasoning systems. *Proc. Eleventh Europ. Conf. Artificial Intelligence, ECAI '94*, (Cohn, A.G., Ed.), pp. 307-311. John Wiley & Sons, Chichester.
- Geelen, P.A., & Kowalczyk, W. (1992). A knowledge-based system for the routing of international blank payment orders. *Proc. Twelfth Int. Conf. Artificial Intelligence, Expert systems and Natural Language, Avignon '92*, Vol. 2, pp. 669-677. EC2, Nanterre.
- Geelen, P.A., Ruttkay, Zs., & Treur, J. (1991). *Logical analysis and specification of an office assignment task*. Technical Report IR-283, AI Group, Dept. of Math. and Comp. Science, Vrije Universiteit, Amsterdam.
- Haroud, D., Boulanger, S., Gelle, E., & Smith, I.F.C. (1994). Strategies for conflict management in preliminary engineering design. *Proc. AID '94 Workshop Conflict Management in Design*, (Smith, I.F.C., Ed.).
- Oh, V. & Sharp, J. (1994). Managing conflicts in an interdisciplinary design environment. *Proc. AID '94 Workshop Conflict Management in Design*, (Smith, I.F.C., Ed.).
- Pannekeet, J.H.M., Philipsen, A.W., & Treur, J. (1991). *Designing compositional assumption revision*. Technical Report IR-279, AI Group, Dept. of Math. and Comp. Science, Vrije Universiteit, Amsterdam.
- Petrie, C.J., Cutkosky, M.R., Weber, T., & Conru, A.B. (1994). Next-Link: An experiment in coordination of distributed agents. *Proc. AID '94 Workshop Conflict Management in Design*, (Smith, I.F.C., Ed.).
- Treur, J. (1994). Temporal semantics of meta-level architectures for dynamic control of reasoning. *Proc. Fourth Int. Workshop Meta-Programming in Logic, META '94*, (Turini, F., Ed.), Lecture Notes in Computer Science, Vol. 883. Springer-Verlag, Heidelberg.
- Treur, J., & Willems, M. (1994a). A logical foundation for verification. *Proc. Eleventh Europ. Conf. Artificial Intelligence, ECAI '94*, (Cohn, A.G., Ed.), pp. 745-749. John Wiley & Sons, Chichester.
- Treur, J., & Willems, M. (1994b). On verification in compositional knowledge-based systems. *Proc. ECAI '94 Workshop Validation of Knowledge-Based Systems*, (Preece, A., Ed.), 4-20.