Int. J. Human–Computer Studies (1996) **44,** 469–520

Modelling an elevator design task in DESIRE: the VT example

FRANCES M. T. BRAZIER, PIETER H. G. VAN LANGEN, JAN TREUR, NIEK J. E. WIJNGAARDS AND MARK WILLEMS

Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081a, NL-1081 HV Amsterdam, The Netherlands

An elevator configuration task, the VT task, is modelled within DESIRE as a design task. DESIRE is a framework within which complex reasoning tasks are modelled as compositional architectures. Compositional architectures are based on a task decomposition, acquired during task analysis. An existing generic task model of design, based on a logical analysis and synthesis of task models devised for diverse applications, has been refined for the elevator configuration task. The resulting task model includes a description of the ontology of the elevator domain and a description of the task model.

1. Introduction

Design is a complex process, in which reasoning with different types of knowledge plays an important role. Knowledge of requirements which can be imposed on an artifact (a design object), knowledge of the domain principles and theories involved, but also knowledge of perspectives taken on the artifact, are interwoven. Of equal importance is knowledge of design strategies: knowledge of strategies to manage conflicts between the different interests of the parties involved (e.g. customer, designer), knowledge of strategies to reason about partial designs and partial sets of requirements, etc. To model design, the design process must be made explicit. During conceptual design (including knowledge acquisition) of a design support system, knowledge of both design strategies and design objects must be obtained.

Configuring an elevator, as described by Marcus, Stout and McDermott (1988), Marcus and McDermott (1989), and Yost (1994), is a specific type of design task. Elevator configurations are the design objects, customer specifications and relevant building information define the (initial) requirements, and constraints define design object knowledge. Compiled modification knowledge (default and fix knowledge) guides the design process.

In this paper a generic task model of design (see Brazier, Langen, Ruttkay and Treur, 1994*b*), is applied to the VT task. This model is based on a logical theory of design (see Brazier, Langen and Treur, 1995*a*) and task models of design tasks in different fields of application (for example, Brumsen, Pannekeet & Treur, 1992; Geelen & Kowalczyk, 1992; Geelen, Ruttkay & Treur, 1992). The framework within which the task has been modelled, specified and operationalized, is the DESIRE framework (DEsign and Specifiation of Interactive REasoning components), presented in Langevelde, Philipsen and Treur (1992) and Brazier, Treur, Wijngaards and Willems (1995*c*); introduced below in Section 2. In Section 3, the role of the

1071-5819/96/3/40469 + 52$18.00/0

generic task model of design in initial problem analysis is discussed. The specialization of the generic task model of design to the VT task is presented in Section 4. In Section 5 the specification of an ontology for the elevator domain is described. In Section 6 parts of a sample trace are presented and in Section 7 implementation aspects are discussed. The approach as a whole is discussed in Section 8, together with an indication of areas for future research.

2. Knowledge modelling approach: DESIRE

Within DESIRE the primary focus has been on tasks and interactions between tasks, with slightly less focus on structures for knowledge representation. Both aspects are, however, equally important for knowledge modelling. The framework supports task analysis, formal specification, and operationalization in the form of a prototype system, each of which is discussed below.

2.1. TASK ANALYSIS

To model expert knowledge, the parties involved in knowledge acquisition must reach a common understanding of the problem, possible solutions, and the strategies entailed. Mediating models, discussed by Ford, Bradshaw, Adams-Webber and Agnew (1993), play an important role in this respect. An explicit type of mediating model distinguished within the DESIRE approach to knowledge acquisition is the *shared task model* as presented by Brazier, Treur and Wijngaards (1994*a*). Shared task models are conceptualizations of tasks for which a representation accepted by knowledge engineer(s) and expert(s) is devised during knowledge acquisition. Existing (generic) task models are re-used to guide the initial acquisition process of a shared task model. Which task models are used during knowledge acquisition depends on the initial description of a task or parts of a task: in interaction with experts existing models are examined, discussed, rejected, modified and/or refined.

Within the DESIRE framework a number of (generic) task models exist for this purpose. These models have been defined on the basis of experience and logical analysis. The concept of a generic task, introduced by Chandrasekaran (1986) and Brown and Chandrasekaran (1989), is comparable to the notion of *generic task model* in that they are both generic with respect to domains. Generic task models within the DESIRE framework, however, are generic with respect to both tasks and domain: generic task models can be refined with respect to the task by *specialization* (e.g. further decomposition of a sub-task) and refined with respect to the domain by *instantiation* (e.g. addition of domain-specific knowledge). Moreover, the way a generic task model is specified in DESIRE is more declarative (with semantics based on temporal logic) than the way generic tasks are described in Chandrasekaran (1986) and Brown and Chandrasekaran (1989).

Different levels of abstraction and composition play an important role during knowledge acquisition: more specific knowledge is acquired of the task and task (de)composition, domain-specific knowledge content, interaction between tasks and sub-tasks within the decomposition, interaction between participating agents, etc. As a result, the shared task model becomes more explicit, more refined (i.e. specialized and instantiated). Very specific levels of detail are, however, most frequently

omitted: such details are often neither applicable nor relevant for the understanding of the task by the different parties involved (e.g. knowledge engineer(s) and expert(s)).

Domain-specific knowledge is modelled in knowledge structures, and is included in task models by references to such structures. Which techniques are used for knowledge elicitation is not predefined. Techniques vary in their applicability, depending on the situation, the resources, the task, the type of knowledge on which the knowledge engineer wishes to focus, etc.

The different types of knowledge included in a task model can be represented in a number of ways: informal and formal representations, but also conceptual and specific representations are possible. A complete specification is, however, always formal. The types of knowledge included in a task model are as follows.

- The task decomposition.
- Information exchange between subtasks.
- Sequencing of subtasks.
- Knowledge structures.
- Task delegation.

Each of these types of knowledge is discussed below in more detail.

2.1.1. Task (de)composition

To model and specify (de)compositions of tasks, knowledge is required of:

- a task hierarchy;
- information a task requires as input and information a task produces as output;
- meta-object relations between (sub)tasks (i.e. which (sub)tasks reason about which other (sub)tasks).

Within a *task hierarchy,* composed and primitive tasks are distinguished: in contrast to primitive tasks, composed tasks are tasks for which sub-tasks are identified. Sub-tasks, in turn, can be either composed or primitive. A task hierarchy defines which sub-tasks are distinguished and the task–sub-task relations between them. This entails a one-to-many relation.

The types of information required as *input* for a (sub)task or generated as *output* as a result of (sub)task performance are specified explicitly in the interface of the (sub)task. The actual contents of these specifications is given through reference to the knowledge structures described below.

Reflective reasoning is an essential element in most complex reasoning processes. Tasks which include reasoning about other tasks (for example about the results of a task or lack thereof, about the goals to be pursued, about assumptions, defaults, preferences, etc.) are modelled as *meta-level tasks* with respect to *object-level tasks.* Often more than two levels of reasoning are involved in a complex task, resulting in meta-meta- . . . reasoning tasks.

2.1.2. Information exchange

Knowledge of *information exchange* between (sub)tasks defines the types of information transferred between (sub)tasks, explicitly specified by relations between tasks. Also the grounds are defined upon which the ‘‘decision’’ to transfer this

information is based. Explicit evaluation criteria may be specified for this purpose, for example that a specific (sub)task has succeeded in deriving specific information.

2.1.3. Sequencing of tasks

Knowledge of *task sequencing* defines temporal relations between (sub)tasks: which tasks must (directly) precede other tasks and which may be activated in parallel. Task sequencing knowledge specifies under which conditions which tasks (directly) precede which other tasks. These conditions, preconditions for task activation, may be based on *evaluation criteria* expressed in terms of the evaluation of the results (success or failure) of one or more of the preceding tasks. The evaluation criteria, the result and the name of the next task(s) to be activated are specified explicitly.

Task control is limited to evaluation and activation of a task's immediate sub-tasks and is independent of the content of underlying (sub)tasks and knowledge.

2.1.4. Knowledge structures

During knowledge acquisition appropriate *structures* for domain *knowledge* must be devised. These structures may be referenced within a task decomposition to specify input and output information types, and knowledge bases. The meaning of the concepts used to describe a domain and the relations between concepts and groups of concepts, must be determined. Concepts are required to identify objects distinguished in a domain, but also to express the methods and strategies employed to perform a task. Often concepts and relations between concepts are defined in knowledge structures such as hierarchies and rules, but alternative knowledge structures are possible.

2.1.5. Delegation of tasks

In complex situations often a number of autonomous systems and/or users are involved. Knowledge of task delegation refers to the division of tasks amongst these participants. In a minimally interactive task, tasks can be divided between an automated system and an end-user. In more complex situations often more participants are involved. Essentially *task delegation* is defined by a set of participants (i.e. *agents*) and a relation between tasks and agents.

2.2. FORMAL SPECIFICATION

Within the declarative DESIRE framework, conceptual task models are acquired and mapped onto compositional architectures for which formal specifications are devised. The five types of knowledge distinguished above are formally specified.

2.2.1. Task (de)composition

Tasks and sub-tasks correspond to *components* and *sub-components*: composed tasks to *composed components,* primitive tasks to *primitive components.* Each component has a *kernel.* The kernel of a primitive component may be specified by either a *knowledge base* or another type of specification tuned to the technique used (e.g. neural network, database, calculation module, OR algorithm). The kernel knowledge of composed components contains specifications of sub-components.

For each of the information types required/produced by a (sub)task, *signatures*

are referenced for the *input* and *output* interfaces of the (sub)task. Signatures are specified by knowledge structures that define units of information: (ground; i.e. instantiated) *atoms* and their *truth values* (`true, false, unknown`).

Meta-object relations between tasks are specified by means of levelled signatures. Within input and output signatures different meta-object levels are distinguished and between signatures meta-object level links are defined.

2.2.2. Information links

Interaction between and within tasks and sub-tasks is defined by activation of *information links* between components and sub-components. Links may transfer not only information generated in one component to be used as input by another component, but also meta-information about the reasoning process itself, such as the goals, assumptions, and epistemic information. Each information link relates output of one component to input of another by specifying which truth value of a specific output atom is linked with which truth value of a specific input atom. This allows for *renaming* of atoms: each component may have its own lexicon, independent of other components.

Within composed components the following types of links are distinguished.

- Links between the input interface and a sub-component, and between a sub-component and the output interface (*mediating links*).
- Links between sub-components (*private kernel links*).

The conditions for activation of information links are explicitly specified as (part of) task control knowledge.

2.2.3. Task control knowledge

Specification of knowledge of task sequencing is distributed over the component hierarchy. Within a component, knowledge of task sequencing is explicitly modelled as *task control knowledge.* It includes not only knowledge of which sub-tasks should be activated when and how, but also knowledge of the goals associated with task activation and the amount of effort which can be afforded to achieve a goal to a given extent. These aspects are specified as sub-component and link activation together with sets of *targets* (specifying evaluation criteria) and *requests, extent* and *effort* to define the sub-component's goals. Sub-components are, in principle, black boxes to the task control of an encompassing component: task control is based purely on information about the success and/or failure of component activation. Activation of a component is considered to have been successful, for example, with respect to one of its target sets if it has reached the goals specified by this target set (and specifications of the number of goals to be reached—e.g. any or every—and the effort to be afforded).

2.2.4. Knowledge structures

Within DESIRE *knowledge structures* such as hierarchies and rules are specified by *signatures* and *knowledge bases* expressed in order-sorted predicate logic. Knowledge structures may include references to other knowledge structures.

The advantage of reference is not only that existing specifications may be easily

adopted (existing ontologies), but also that during the knowledge modelling process non-instantiated specifications may be referenced (to be specified at a later date).

2.2.5. Task delegation

Relations between tasks and *agents* are explicitly specified. Assignment of tasks can be determined in advance, but may also be dynamic (determined during task execution, for example).

2.3. FORMAL SEMANTICS AND OPERATIONALIZATION

The formal specification of compositional architectures defines the formal semantics of a system's behaviour in terms of temporal logic (Brazier *et al.,* 1995*c*; Engelfriet and Treur, 1994). Validation and verification can be based on such specifications, but in addition an operational (prototype) system can be automatically generated.

Formal specifications in DESIRE can be translated directly into operational code, for testing and demonstration purposes. The DESIRE framework has tools for this purpose, including implementation generators and interpreters for a number of environments (such as ADS and PROLOG) and platforms. Syntax-directed and graphical editors support different phases of specification.

3. Initial problem analysis

As discussed above, knowledge acquisition within the DESIRE approach is most often based on direct interaction with domain experts aimed at deriving a shared task model of the task at hand. Generic task models provide support during the initial phase of modelling. For the VT task, direct interaction with experts was not possible; the domain description provided by Yost (1994) was the only material available for analysis. This description included compiled (expert) knowledge of the design task; more detailed knowledge was often not included in the task description. As the DESIRE approach to modelling a design task ''from scratch'' would have been to structure the knowledge acquisition process on the basis of the generic task model of design, this approach has been simulated for the VT task. Before discussing the initial problem analysis in relation to the generic task model of design, the generic task model itself is briefly described.

3.1. GENERIC TASK MODEL OF DESIGN

Design is a dynamic complex task in which requirements and (partial) design object descriptions are (continually) manipulated until a satisfactory solution has been found. Conflicting interests, requirements, design possibilities, and design strategies are inherent to design tasks, as is the coordination of (parallel) partial design processes. The generic task model of design can be used for two types of processes: (1) a single designer's design process and (2) the coordination between designers' design processes, itself a design process. The VT task describes a process of the first type.

The generic task model of design described by Brazier *et al.* (1994*b*) assumes the existence of a problem statement and more specific knowledge of (initial) *requirements* and *requirement qualifications,* in addition to knowledge of the *design objects* and knowledge of *design strategies.* Requirements, the necessary and desired properties of the design object (within a given context), are not all equally ''important.'' Preferences often exist between requirements, and/or sets of requirements. These preferences, modelled as requirement qualifications, together with the requirements themselves, are often modified on the basis of the evaluation of (partial) design object descriptions, frequently in interaction with the customer. To determine which set of requirements to consider at a particular point in time (on its own or in parallel with other sets of requirements), often depends on what is known about the (partial) design object description (for example, to which extent requirements are fulfilled), about the requirements (for example, whether they are conflicting or not), and about the relations between views within the design process.

Strategies employed for the creation of the design artifact itself necessarily consider different types of knowledge. A description of a design object from one point of *view* will often differ from a description of the same object from another point of view. The design object description is most often partial: it is extended and modified during design, on the basis of additional knowledge and integration of sub-solutions.

Design process coordination, in particular *design process evaluation,* analyses the current state of the design process and determines which strategy to employ for exploration of the design space. This strategy influences the coordination of the manipulation of requirement qualification sets and design object descriptions. The result of the design task, a design object description, fulfils a set of requirements (developed during the design process) and complies with the knowledge of the domain.

Figure 1 shows a compositional architecture for the design task, in which (sub)components are arranged hierarchically, corresponding to the task decomposition. A short description of the model is given below in three sections, each describing a main component of the model: (1) `requirement-qualification-set-manipulation`, (2) `design-object-description-manipulation`, and (3) `design-process-coordinator`. In the sequel, requirement qualification set will often be abbreviated as RQS, design object description as DOD, and design process coordination as DPC.

3.1.1. Requirement qualification set manipulation

Requirements and their qualifications are acquired from the customer. Requirement qualification set manipulation guides the process of requirement qualification acquisition. Given a set of requirements and their qualifications, the determination of the most relevant subset of requirement qualifications entails a closer analysis of the qualifications (e.g. relevance, importance, strength) of the individual requirements and their relations. 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

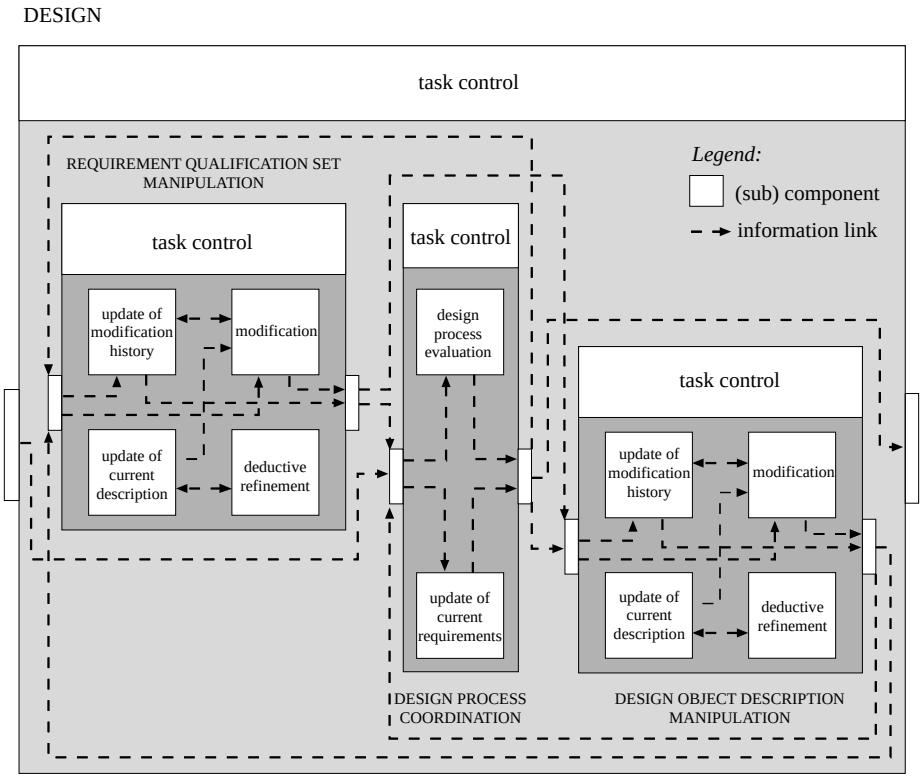

FIGURE 1. Generic task model of design: task decomposition and information links.

determination of the set of requirements to be considered, are based on knowledge of preferences between requirements. Within the `requirement-qualification-set-manipulation` component, the following sub-components are distinguished.

- The `RQS-modification` component determines the modification of the current set of requirement qualifications.
- The `RQS-deductive-refinement` component determines which requirements are implied by the resulting set of requirement qualifications.
- The `RQS-update-of-current-description` component keeps track of the most recent requirement qualification set.
- The `RQS-update-of-modification-history` component keeps track of the requirement sets considered during the design process.

To determine which requirements to consider first, which to ignore, and which to modify or add (e.g. by decomposing requirements into more specific requirements), the possible modifications need to be considered. Explicit ranking criteria between preferred sets of requirements are sometimes available, but most often other types of strategic knowledge are required. One global strategy for determining which modifications are most relevant can be based on a possible distinction between the sources of a requirement: requirements based on user preferences may be given higher priority than requirements formulated on the basis of default assumptions (similar to the approach described by Haroud, Boulanger, Gelle & Smith 1994).

3.1.2. Design object description manipulation

Creating a design object description on the basis of the requirements imposed, entails determining a strategy for design object description construction. This process is similar to the requirement qualification set manipulation process, although the knowledge differs considerably. Within the `design-object-description-manipulation` component, the following sub-components are distinguished.

- The `DOD-modification` component determines which parts of the current design object description should be modified.
- The `DOD-deductive-refinement` component determines, on the basis of its domain knowledge which information on the design object description is implied by the (modified) current design object description.
- The `DOD-update-of-current-description` component keeps track of the most recent design object description.
- The `DOD-update-of-modification-history` component keeps track of the design object descriptions considered during the design process.

The relationship between the `update-of-modification-history` components of the `RQS-manipulation` component and the `DOD-manipulation` component is explicitly defined.

3.1.3. Coordination of the design process

Coordination of the design process is dedicated to determining whether to continue the design process or not, and if so, how (according to which strategy). It consists of two sub-components namely `design-process-evaluation` and `update-of-current-requirements`.

The component `design-process-evaluation` is responsible for determining whether it makes sense (from a strategic point of view) to continue the design process by manipulating either the current requirement qualification set or the current design object description. For this purpose, it monitors the progress of the design process (by making use of information on the modification histories maintained by the `update-of-modification-history` components), decides on a coordination strategy and informs the manipulation components about its strategic decision.

The task of `update-of-current-requirements` maintains the set of requirements to which the design process is (temporarily) committed.

3.2. THE VT TASK

The VT task is clearly a design task: requirements exist and an object is designed on the basis of the requirements. The requirements given in the VT task description are the problem specification values: customer specifications and relevant building information. The object designed is an elevator configuration. The problem specification values (input values) may be changed, if necessary, although this is considered to be highly undesirable. Initially, all requirements are considered to be of equal importance. Modification knowledge is compiled into defaults and into fixes with different levels of desirability. The least desirable are the fixes that modify elements of the design object description which contradict requirements imposed by the customer. The most desirable are the fixes that prescribe an alternative value

for elements of the design object description. As most of this knowledge is ''hard-wired'' in the VT task description, further analysis was required to interpret the knowledge presented.

As mentioned in Section 2.1, the generic task model of design described in Section 3.1 was devised on the basis of a logical analysis of design, as well as analysis of and abstraction from more specific task models developed for a number of domains, such as design of measures for environmental policy, routes for international payment orders, and office assignments. The application of the generic task model to the VT task as described in this paper has provided validation of the genericity and usefulness of the generic task model in a new design domain.

4. Problem solving method

In many knowledge modelling approached (such as KADS $(ML)^2$, PROTÉGÉ, DIDS, KARL, VITAL, see articles in this issue) the concept of *problem solving method* plays an important role. In DESIRE, the concept of a non-instantiated (i.e. not instantiated with knowledge structures for the specific domain) task model is comparable to the concept of a problem solving method. Problem-solving methods are specified in the form of task models that are generic with respect to the specific domain. The strict relation between task decomposition and control decomposition (i.e. specification of the task control is distributed over the task hierarchy) as employed in DESIRE, however, is not included in, for example, KADS-based approaches such as KADS/$(ML)^2$ and KARL. In this section, the generic task model of design introduced in Section 3, is refined to a problem solving method for the VT task: a task model with references to knowledge structures that are instantiated only with knowledge independent of the elevator domain.

To develop a task model of the VT task, the VT task description provided by Yost (1994) has been thoroughly analysed. As direct interaction with an expert was not one of the options available, the task model of the VT task could only be validated by analysis of the test case in Section 9 of Yost (1994). This test case specifies parameter values before and after design modifications, all constraints violated before design modifications, and the design modifications employed to resolve these constraint violations.

The organization of this section is again based on the five types of knowledge distinguished in DESIRE: task decomposition, information links, task control control knowledge, knowledge structures, and task delegation. Only knowledge structures that are independent of the elevator domain are included in the task model described in this section; knowledge structures specific to the elevator domain remain non-instantiated. These non-instantiated domain-specific knowledge structures for the elevator domain are the subject of Section 5.

4.1. TASK DECOMPOSITION

The three elements in a task decomposition (the task hierarchy, input and output specification, and meta-object level distinctions) are described below for (parts of) the VT task. The task decomposition will be motivated by citations from Yost (1994). To illustrate the formalization of these concepts within DESIRE, examples of both graphical and textual formal specifications are presented.

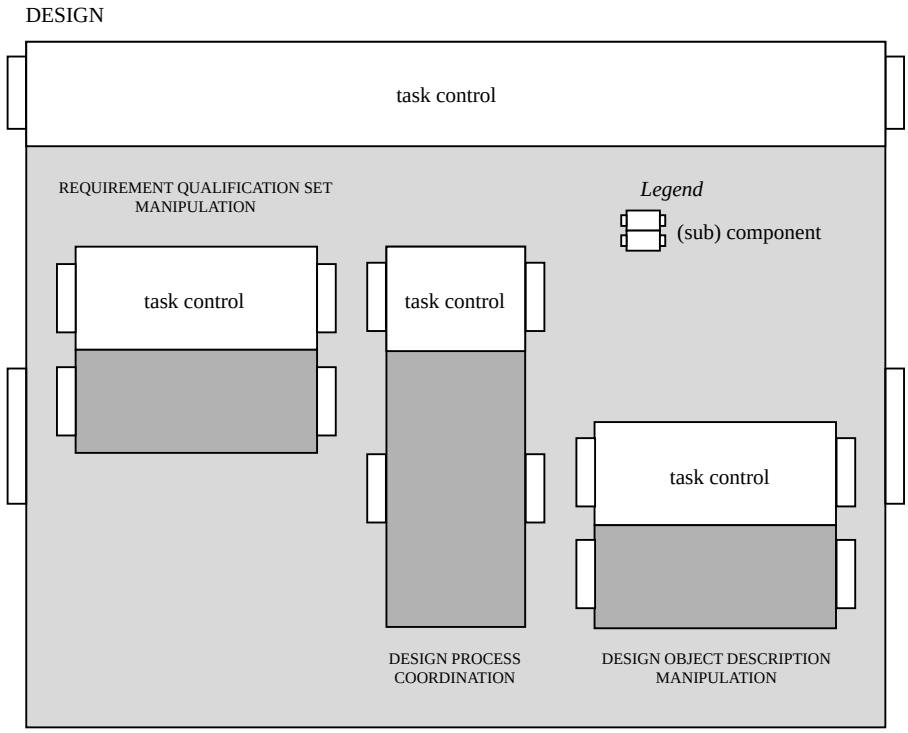

FIGURE 2. Block notation for the top level of the generic task model of design.

4.1.1. Task hierarchy

Task hierarchies can be represented graphically in many ways. In DESIRE, *tree* and *block notation* are two graphical representations often employed. For task hierarchies, these two notations are equivalent. In this section, to illustrate the representation, the block notation is used to specify the top level of the task hierarchy of the generic task model of design (see Figure 2). The hierarchies for the two manipulation sub-tasks will be represented by the tree notation (see Figures 3 and 4). The block notation will also be used to represent a lower level part of the design task hierarchy (see Figure 5).

Section 1.2 of Yost (1994) describes an elevator configuration system that is able to:

- accept customer specifications and relevant building information,
- derive preliminary values for parts and parameters,
- check for constraint violations,
- propose and implement configuration modifications until a complete configuration with no constraint violations is devised,
- print a description of the final configuration

As discussed in Section 3.2 of this paper, the VT elevator configuration task can be viewed as a design task. In the generic task model of design, the design task is decomposed into three sub-tasks, depicted in block notation in Figure 2. In a graphical block notation, composed components are depicted as nested blocks, in

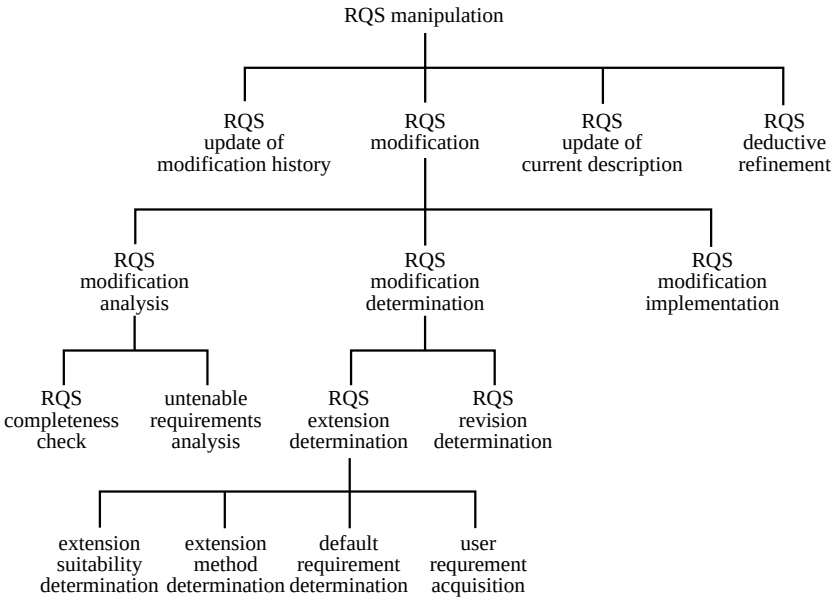

FIGURE 3. Complete task decomposition for `RQS-manipulation`.

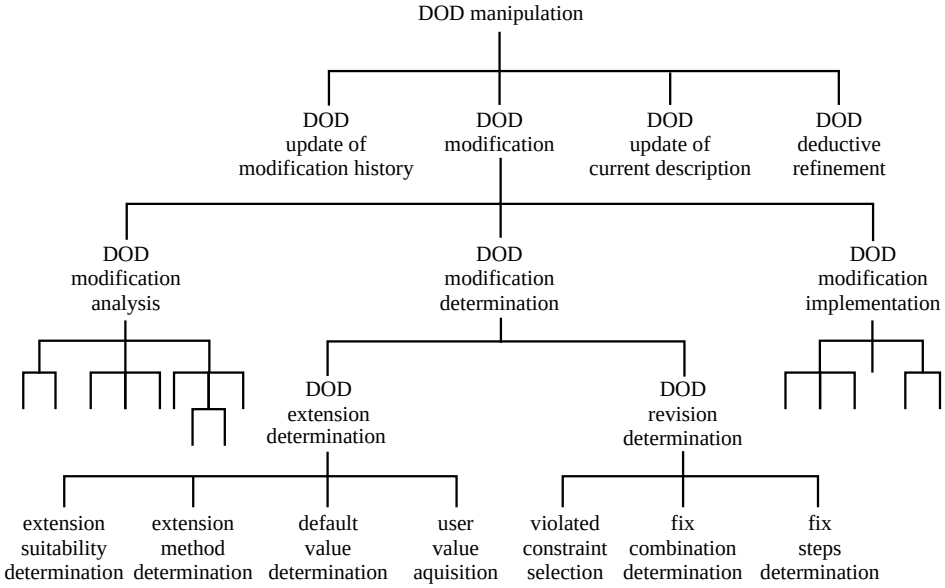

FIGURE 4. Partial task decomposition for `DOD-manipulation`.

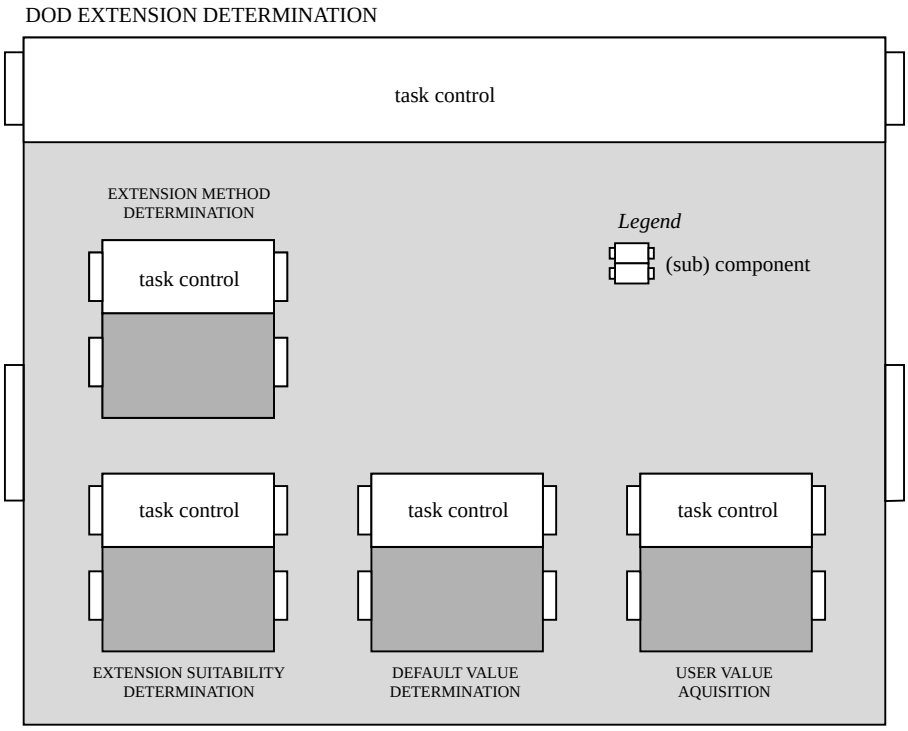

FIGURE 5. Block notation for a lower level task.

which not all abstraction levels are necessarily represented: the content of the kernel of a component at a lower level may be left unspecified. In Figure 2 a block representation of only the top level of the task model is depicted. The three main components of the task model are shown, together with the task control at each of the two levels of abstraction depicted.

The sub-task *requirement qualification set manipulation* acquires requirements of the elevator configuration in the form of customer (or contract) specifications and relevant building information. The sub-task *design object description manipulation* derives preliminary values for parts and parameters, checks for constraint violations, and modifies the configuration until it is complete and violates no constraints. The sub-task *design process coordination* determines and monitors the elevator configuration strategy suggested in Section 1.4 of Yost (1994), according to which subtasks related to the above five capabilities are invoked. The subsequent sub-task activations, determined by design process coordination, are in accordance with the procedure specified by Yost (1994).

As in Section 3, requirement qualification set will often be abbreviated as RQS, design object description as DOD, and design process coordination as DPC.

As explained in Section 3.1, the task of RQS manipulation is decomposed into four sub-tasks (see Figure 3 for a representation in tree notation). The sub-task *RQS modification* adds customer specifications and relevant building information to the current (and initially empty) set of requirements and deletes requirements that are

untenable in view of finding a solution by means of the given knowledge. As no essential distinctions in qualifications of requirements are made in the documentation of the VT task, the qualifications are not explicitly mentioned in the presentation of the VT task model. The RQS modification sub-task is decomposed in a separate sub-section below. The sub-task *RQS deductive refinement* deduces additional requirements from those present in the current requirement set. Unfortunately, Yost (1994) does not provide knowledge for this sub-task. The sub-task *RQS update of current description* keeps track of the contents of the current requirement set and the sub-task *RQS update of modification history* records the modifications to requirement sets made during the design process. These two sub-tasks are implicitly present in a number of text fragments in Yost (1994).

The task of DOD manipulation is similarly decomposed into four sub-tasks (see the tree representation in Figure 4). The sub-task *DOD modification* asserts initial values for parts and parameters, checks for constraint violations, and modifies the current configuration until it is complete and does not violate any constraints. This sub-task is further decomposed in a separate sub-section below. The sub-task *DOD deductive refinement* deduces additional parameter values from those present in the current configuration. Yost (1994) provides ample knowledge for this sub-task; for example, in Section 4.2 it is stated that ''[the SLING UNDERBEAM] *is equal to the CAR CAB HEIGHT · · · plus the SLING UNDERBEAM SPACE.*'' The sub-task *DOD update of current description* keeps track of the contents of the current configuration and the sub-task *DOD update of modification history* records the configuration modifications made during the design process. These two sub-tasks are explicitly indicated by Yost (1994) in Section 7: ''*Tentatively make the changes . . . ,*'' ''*. . . undo the tentative changes . . . ,*'' and ''*. . . the tentative changes should be made permanent.*''

The task of design process coordination is decomposed into two sub-tasks. The sub-task *design process evaluation* determines, on the basis of the results achieved at a given point in the design process, which behaviour is appropriate. Section 1.4 of Yost (1994) provides knowledge on this matter. The sub-task *update of current requirements* keeps track of the current requirements produced by RQS manipulation that need to be satisfied by any elevator configuration produced by DOD manipulation.

It can be observed from the above task descriptions that most of the functionality required for the elevator configuration system has to be provided by the modification subtasks of RQS manipulation and DOD manipulation. These two sub-tasks are further decomposed in the following two sub-sections.

Task hierarchy of RQS modification

As argued earlier, RQS modification is a complex task for which a characterization must be sought. Yost (1994) states in Section 1.4 the following: ''*First, get values for all of the input parameters . . . from the customer.*'' In other words, requirements are acquired from the user, possibly in subsequent steps, until for each input parameter known to be relevant for configuring elevators, a requirement exists that prescribes its value. RQS modification can be seen as a process control task: it is decomposed into three sub-tasks that (1) perform an analysis of the current state, (2) determine a next modification, and (3) implement this modification. This is a task decomposition

of a *process control task* that is adopted from the analyses of, and task model for, process control by Brazier, de Klerk, van Langen, and Treur (1993).

In Figure 3 the complete task decomposition of RQS modification is included. The sub-task *RQS modification analysis* examines the results of modifying the requirements. It determines whether the last modification of the requirement set has resulted in a complete requirement set. This first element is mentioned in Section 1.4 of Yost (1994): "*First, get values for all of the input parameters . . .*" This element also checks whether the last modification, or any earlier modifications, introduced untenable (sets of) requirements. The second element is implied in Section 7 of Yost (1994): "*. . .* [fixes] *may require changing building dimensions or contract specifications.*" The sub-task *RQS modification determination* proposes a modification to the current requirement set. The sub-task *RQS modification implementation* applies the proposed modification to the current requirement set. Any further consequences can be derived by RQS deductive refinement (although, as remarked earlier, Yost (1994) does not provide knowledge for this sub-task).

The task of RQS modification determination has been decomposed into two sub-tasks. The sub-task *RQS extension determination* proposes a requirement to be added to the current requirement set. This task has been decomposed into four sub-tasks. The sub-task *extension suitability determination* selects a type of requirement that needs to be added to the current requirement set and the sub-task *extension method determination* proposes a method to specify a requirement of the selected type. Yost (1994) states in Section 1.2 that there are two types of requirements: "*. . . customer specifications and relevant building dimensions.*" The sub-task *user requirement acquisition* collects requirements on input parameters from the customer. The sub-task *default requirement determination* specifies default requirements that are applicable to most configuration problems. Unfortunately, Yost (1994) does not provide knowledge for this sub-task. The second sub-task of RQS modification determination, *RQS revision determination,* proposes a set of changes to the current requirement set to remove untenable requirements.

Task hierarchy of DOD modification

DOD modification is a complex task which requires further analysis. Yost (1994) states in Section 1.2 that the configuration has to be modified "*until a complete configuration with no constraint violations is achieved.*" In Section 1.4, Yost again postulates to stop modifying the configuration "*when there are no more parameters or constraints to process . . . .*" This suggests that DOD modification can be regarded as process control task. Figure 4 includes a partial task decomposition of DOD modification in the form of a tree. (The task decompositions of DOD modification analysis and DOD modification implementation which are present in our VT task model are not shown in this paper.)

As a process control task, the task of DOD modification can be decomposed into three sub-tasks. The sub-task *DOD modification analysis* investigates the results of the last modification. It determines whether the last modification resulted in a complete configuration, whether it produced a configuration without any violated constraints, or whether it fixed a particular constraint violation without introducing any new violations. The first two elements are both mentioned in Section 1.2 and the third in Section 7 of Yost (1994). The sub-task *DOD modification determination*

proposes a modification to the current configuration that has not yet been tried. Yost (1994) apparently assumes that a modification as such, if needed, can be determined at any point in design, even when fixing constraint violations. Whenever a constraint violation has been detected, DOD modification determination proposes revisions of the current configuration to resolve this violation. This is explicitly stated in Yost (1994), Section 7: "*If the constraint is violated . . . you should immediately try to find design modifications that remedy the violation.*" The sub-task *DOD modification implementation* applies the proposed modification to the current configuration. DOD deductive refinement (deductively) derives additional information required.

The task of DOD modification determination has been decomposed into two sub-tasks. The sub-task *DOD extension determination* proposes a value for a parameter that does not have a value in the current configuration. The sub-task *DOD revision determination* proposes a set of changes to existing parameter values to resolve a particular constraint violation in the current configuration. These complex sub-tasks are decomposed further below.

The task of DOD extension determination has been decomposed into four sub-tasks. The sub-task *extension suitability determination* selects a parameter that does not yet have a value in the current configuration and the sub-task *extension method determination* proposes a method to assign a value to the selected parameter. Yost (1994) suggests in Section 1.4, the following procedure for determining a parameter and a method: "*First, get values for all of the input parameters . . . from the customer. Then, derive values for all of the other parameters . . . . Values for parameters can be derived in any order, once values have been derived for any parameters on which they depend.*" The sub-task *user value acquisition* extracts values for input parameters from the requirements provided by the customer. The sub-task *default value determination* assigns initial values to parameters, provided that knowledge about initial (i.e. default) values for these parameters is available. For example, Section 4.2 of Yost (1994) states that "[the SLING UNDERBEAM SPACE] *is initially 21 inches, but may be changed to fix constraint violations . . . .*" In addition, DOD deductive refinement can be used to deduce values for parameters (by computation) from the values of parameters already assigned. This has been explained above as part of the decomposition of DOD manipulation.

The task of DOD revision determination has been decomposed into three sub-tasks. The sub-task *violated constraint selection* selects a violated constraint to be resolved next. Yost (1994) states in Section 7 that if "*more than one constraint can be processed at the same time, pick one arbitrarily,*" although "*. . . if both MACHINE GROOVE PRESSURE and HOIST CABLE TRACTION RATIO constraints are violated at the same time, try to fix the MACHINE GROOVE PRESSURE violation first.*" The sub-task *fix combination determination* proposes a set of fixes that could resolve the selected constraint violation. Section 7 of Yost (1994) describes an algorithm to compute fix combinations, the order of which is determined by the desirability of the fixes involved. The sub-task *fix steps determination* computes for the current fix combination which steps for which of the fixes should be tried. Yost (1994), Section 7 states: "*Some fixes specify that a value should be stepped along some dimension . . . , all possible combinations of*

steps . . . should be tried before moving on to the next basic fix combination.'' An example of a fix for which this holds is given for the CAR BUFFER BLOCKING HEIGHT constraint: ''*. . . try increasing the HOISTWAY PIT DEPTH . . . by one-inch steps . . .*''

4.1.2. Input and output specifications

The information required by a sub-task modelled and specified as input for the corresponding sub-component, is specified for each sub-task within the hierarchies presented above. This also holds for the information produced by a sub-task, modelled and specified as output of the corresponding component. At the most abstract level depicted above in Figure 2 for the VT task, the initial values for specific parameters given by the customer or representing relevant building information, are input for the design task (and used for the formulation of requirements). The input interface is depicted by the long narrow rectangle attached to the left edge of the outer block. Other values for these parameters may be determined during the design process, but values for other parameters are not expected as input (and thus not explicitly modelled). The final output of the VT task is a list of values for the parameters specified by Yost (1994). In addition, the list of requirements on which the final design has been based, can be seen as a result, and thus as part of the output of the design task. The output interface is depicted in Figure 2 by the long narrow rectangle attached to the right edge of the outer block. During the design process, questions can be generated for the customer, for instance the question whether a particular requirement may be changed (in order to apply a highly undesirable fix). These questions are modelled as intermediate output.

For a more detailed example, in Figure 5 a block representation of a lower level hierarchy for the task of determining an extension for the design object description, is depicted, including the four lower level tasks distinguished. The hierarchies depicted in Figure 2 and 5 will be used below to illustrate the remaining types of knowledge modelled and specified for the VT task model in more detail.

One of the sub-components of `DOD-extension-determination` depicted in Figure 5 is the sub-component `extension-suitability-determination`. This sub-component determines the next parameters for which a value must be obtained. The output of this component, the parameters suitable for extension, is input for the components `defalut-value-determination`, and `user-value-acquisition`. The input and output information types are specified by named signatures. For instance, the (domain-independent) output signature of the component `extension-suitability-determination` is `Suitable-parameter-sig`, specified below.

The signature definitions are part of the specification of the knowledge structures. A signature in DESIRE is a (partial) declaration of sorts, sub-sorts, objects and functions designating elements of these sorts, and relations defined over these sorts. Also references to other signatures can be used to build up a new signature.

```
signature Parameter-sort-sig
  sorts
    PARAMETER
end signature
```

```
signature Suitable-parameter-sig
  signatures
     Parameter-sort-sig, Parameter-object-sig;
  relations
     suitable-for-extension: PARAMETER;
end signature
```

These signatures together give rise to atoms of the form `suitable-for-extension(P)` with `P` a parameter as defined in `Parameter-sort-sig`. This is task-oriented information that is part of a problem-solving method. The actual parameters, the possible instantiations (such as FLOOR HEIGHT and HOISTWAY DEPTH), are assumed to be given in the domain-specific signature `Parameter-object-sig` to which reference is made. This signature will be discussed in Section 5.

An example of an input signature is `Extension-focus-sig`, the input signature of the component `default-value-determination`, that defines a relation indicating on which parameters to focus:

```
signature Extension-focus-sign
  signatures
     Parameter-sort-sig, Parameter-object-sig;
  relations
     in-focus-of-extension: PARAMETER;
end signature
```

4.1.3. Object-meta distinctions

The VT task model is a task model for a complex reasoning task, namely design. Design entails a considerable amount of reflection.

- Reasoning about requirements (which to consider first, which to adapt given conflicts, etc.) is meta-level reasoning with respect to the requirements.
- Reasoning about a design object description (which part of the design artifact to consider first, which inconsistencies to accept during design, etc.) is meta-level reasoning with respect to the design object description.
- Reasoning about which global design strategy to employ is meta-level reasoning with respect to design process coordination.

Reflection of this nature, inherent to design, is explicitly modelled and specified for the VT task. Within the component `DOD-extension-determination`, for example, the component `extension-method-determination` is a meta-level reasoning component with respect to the components `extension-suitability-determination`, `default-value-determination`, and `user-value-acquisition`. It reasons (by means of domain-specific strategic knowledge) about the most appropriate way to determine the value of the selected parameter. As another example, the `modification` components of both `requirement-qualification-set-manipulation` and `design-object-description-manipulation` are meta-level components with respect to their `deductive-refinement` components: they reason about the results of the reasoning within

deductive-refinement (e.g. the fact that the value of a particular parameter has not been derived yet by DOD-deductive-refinement).

4.2. INFORMATION LINKS

As described in Section 2, links are used to model and specify exchange of information between components. Mediating links are used to specify interaction between the input and output interfaces of a component and sub-components: (1) to transfer information provided as input to a component to a sub-component, and (2) to transfer information produced as output of a sub-component to the output interface of the component. This holds, for example, for the transfer of customer requirements and building specifications, provided as input to the VT task, to the input interface of the component requirement-qualification-set-manipulation. Private links are used to transfer information between sub-components. The component RQS-modification within RQS-manipulation, for example, transfers the initial list of requirements and building specifications to the history component to be stored for possible future reference.

The information links for the top levels of the VT task model are depicted graphically in Figure 1 in Section 3. The information links for the component DOD-extension-determination are depicted in Figure 6.

Which information links are used within a component is specified as part of its

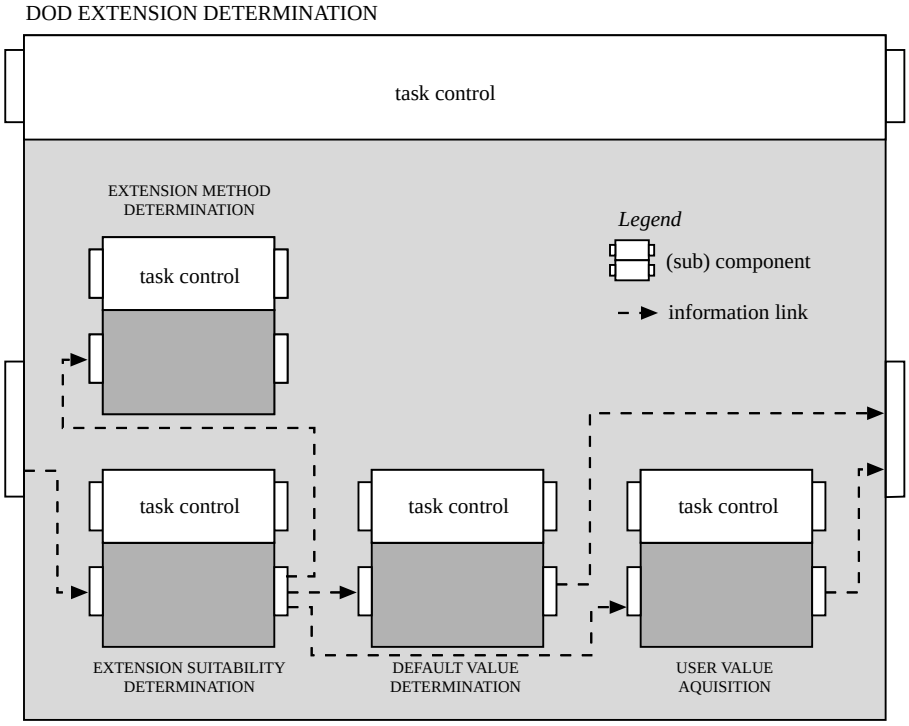

FIGURE 6. Information links in the component DOD-extension-determination.

task information; the partial specification of DOD-extension-determination's task information is as follows:

```
task information DOD-extension-determination-info
  sub-components      extension-suitability-determination,
                      extension-method-determination,
                      default-value-determination, user-value-
                      acquisition;
  informationlinks    configuration-parameters-from-input,
                      epistemic-info-on-focus-determination,
                      focus-for-default-value-determination,
                      default-value-to-output,
                      focus-for-user-value-determination, user-
                      value-to-output;
  (task control omitted here)
end task information DOD-extension-determination-info
```

An example of an *information link* within DOD-extension-determination is the link between the component extension-suitability-determination and default-value-determination, with which the parameters focused on are transferred. The specification of this link is shown below:

```
private link focus-for-default-value-determination: object-object
  domain              extension-suitability-determination
    output
      level           object-output
      signature       Suitable-parameter-sig;
  co-domain           default-value-determination
    input
      level           object-input
      signature       Extension-focus-sig;
  sort links          identity
  object links        identity
  term links          identity
  atom links          (suitable-for-extension(P: PARAMETER), in-focus-
                      of-extension(P: PARAMETER)): ⟨⟨true, true⟩,
                      ⟨false, false⟩, ⟨unknown, unknown⟩⟩.
end link
```

This link relates output of extension-suitability-determination to input of default-value-determination. If this link is activated (this depends on task control knowledge, see Section 4.3), the truth value of the atom suitable-for-extension(P: PARAMETER) is transferred from extension-suitability-determination to default-value-determination and the atom is renamed into in-focus-of-extension(P: PARAMETER).

An example of an *object-meta link* is the link which transfers information about the parameters (whether they have been determined suitable or not) to the meta-level component extension-method-determination to reason about an

appropriate method. This link between extension-suitability-determination and extension-method-determination uses the meta-level signature Epistemic-output-of-extension-suitability-determination declaring the standard unary relations true, false, and known on terms corresponding to atoms at the object level of extension-suitability-determination. The epistemic meta-predicates true, false, and known are two-valued. In the link, only the meta-level relation true is used. This relation is true for an atom a on the object level if and only if a has truth value true (i.e. the relation is false if a has truth value false or unknown). The signature Extension-focus-determination-results-sig declares the unary relations determined-to-be-in-focus-of-extension and left-outside-focus-of-extension on PARAMETER. The link is specified as follows:

```
private link     epistemic-info-on-focus-determination:
                 epistemic-object
  domain         extension-suitability-determination
   output
   level         epistemic-output
   signature     Epistemic-output-of-extension-suitability-
                 determination;
  co-domain      extension-method-determination
   input
     level       object-input
     signature   Extension-focus-determination-results-sig;
  sort links     identity
  object links   identity
  term links     identity
  atom links     (true(suitable-for-extension(P: PARAMETER)),
                 determined-to-be-in-focus-of-extension(P:
                 PARAMETER)):
                 ⟨⟨true, true⟩, ⟨false, false⟩⟩;
                 (true(suitable-for-extension(P: PARAMETER)),
                 left-outside-focus-of-extension(P: PARAMETER)):
                 ⟨⟨false, true⟩, ⟨true, false⟩⟩;
end link
```

This link relates output of the component extension-suitability-determination to input of the component extension-method-determination, by which the truth value true(or false) of an atom of the form true(suitable-for-extension(P: PARAMETER)) is translated into the truth value true(or false) of an atom of the form determined-to-be-in-focus-of-extension(P: PARAMETER) and the truth value false(or true) of the atom true(suitable-for-extension(P: PARAMETER)) is translated into the truth value true(or false) of an atom of the form left-outside-focus-of-extension(P: PARAMETER). Note that also this link renames atoms between components.

Information link names are used in the task control of DOD-extension-determination to specify under which conditions to transfer the up-to-date information.

Information link names are used in the task control of DOD-extension-determination to specify under which conditions to transfer the up-to-date information.

4.3. TASK CONTROL KNOWLEDGE

Knowledge related to sequencing of tasks is modelled and specified as task control knowledge, as discussed in Section 2. Task control knowledge does not specify a fixed sequence of component activation but defines the global conditions for component and link activation. Parallel activation of components is therefore possible, although not applied in the VT task. Each composed component has its own task control knowledge, as shown in Figure 2 for the VT task. Task control specifies under which conditions and how (e.g. with what evaluation criteria, extent of reasoning, and effort to be afforded) components and the related links are to be activated. Top-level task control knowledge within the VT model, for example, specifies the conditions for activation of the three top-level sub-components and the information links between the three sub-components. Each of these sub-components has its own task control knowledge to specify when and how its sub-components and links are to be activated, etc.: task control knowledge is distributed over the component hierarchy.

To illustrate the specification of task control knowledge, activation of the sub-tasks in DOD-extension-determination will be used. An example in which the success of one component is required before a next component can be activated (with the necessary information) is the following rule:

```
if      evaluation(extension-suitability-determination, parameter-
        suitability, succeeded)
  and   previous-component-state(extension-suitability-determination,
        active)
then    next-component-state(extension-method-determination, active)
  and   next-target-set(extension-method-determination, method-
        suitability)
  and   next-link-state(epistemic-info-on-focus-determination, up-to-
        date);
```

This (temporal) task knowledge rule states that:

if the component extension-suitability-determination has just succeeded in accomplishing the targets defined by its target set parameter-suitability according to its effort and extent settings (i.e. it has determined some suitable parameters),

then the component extension-method-determination is assigned a new set of targets method-suitability to accomplish, and it is to be activated (with the aim of determining methods by which values for the parameters in focus should be found) with information that the link epistemic-info-on-focus-determination has updated after activation of extension-suitability-determination and before activation of extension-method-determination.

Activation of extension-method-determination results in the determination of methods with which values for the parameters focused on can be determined. The success of activation of extension-method-determination is evaluated by establishing whether (and which) one of the target sets suitability-of-default-value-determination or suitability-of-user-value-determination has been successfully achieved. The result of evaluation is used to determine which component is to be activated next, specified in the following task control rules of the component DOD-extension-determination:

```
if      evaluation(extension-method-determination, suitability-of-
        default-value-determination, succeeded)
 and    previous-component-state(extension-method-determination, active)
then    next-component-state(default-value-determination, active)
 and    next-link-state(focus-for-default-value-determination, up-to-
        date);
if      evaluation(extension-method-determination, suitability-of-user-
        value-determination, succeeded)
 and    previous-component-state(extension-method-determination, active)
then    next-component-state(user-value-determination, active)
 and    next-link-state(focus-for-user-value-determination, up-to-date);
```

4.4. KNOWLEDGE STRUCTURES

As discussed in Section 2, specifications of task models include references to specific, applicable knowledge structures such as *signatures* and *knowledge bases.* Signatures define the conceptualization of the domain: a terminological structure in terms of which information and knowledge can be expressed. The specification of signatures was addressed in Section 4.1. The task-specific signature Suitable-parameter-sig was introduced to illustrate the use of signatures. This signature, defined for the output interface of extension-suitability-determination, specifies a relation that indicates which parameters are suitable for extension of the design object description. The link focus-for-default-value-determination interprets the set of suitable parameters as a focus for extension, that is input information for the component default-value-determination.

To determine suitable parameters on which to focus, extension-suitability-determination needs information on which parameters have a value in the current configuration, as well as information on derivational dependencies between parameters. The component extension-suitability-determination receives these types of information through its input interface. The input signature of extension-suitability-determination refers to the signatures Configuration-parameter-sig and Parameter-dependency-sig presented below.

```
signature Configuration-parameter-sig
  signatures
     Parameter-sort-sig, Parameter-object-sig;
```

```
  relations
     parameter-of-configuration: PARAMETER;
end signature

signature Parameter-dependency-sig
  signatures
     Parameter-sort-sig, Parameter-object-sig;
  relations
     dependent-on: PARAMETER*PARAMETER;
end signature
```

Note that Yost (1994) does not impose dependencies between parameters, but he suggests dependencies by expressing domain knowledge on the relations between parameters as equalities of the form $y = f(\bar{x})$ (or the equivalent in words), sometimes accompanied by conditions stating when such equalities are appropriate. The VT task model assumes that all domain knowledge that can be represented in this form. During the configuration process, this knowledge is inspected to determine dependencies between parameters: another form of reflective reasoning within the VT task model described.

The component `extension-suitability-determination` uses the above mentioned types of information to determine the most appropriate focus. This is achieved by reasoning about which parameters are candidates for the focus. This type of information is specified by the signature `Extension-candidate-sig`:

```
signature Extension-candidate-sig
  signatures
     Parameter-sort-sig, Parameter-object-sig;
  relations
     candidate-for-extension: PARAMETER;
end signature
```

The following two knowledge base rules are used to determine which parameters are *not* candidates:

```
if          parameter-of-configuration(P: PARAMETER)
then        not candidate-for-extension(P: PARAMETER);

if          not parameter-of-configuration(P1: PARAMETER)
  and       dependent-on(P1: PARAMETER, P2: PARAMETER)
  and       not parameter-of-configuration(P2: PARAMETER)
then        not candidate-for-extension(P1: PARAMETER);
```

Meta-level reasoning is needed to make the assumption that a parameter *can* be a candidate, on the basis of the information that this parameter has *not* been derived to be a non-candidate. This is a form of closed world assumption and is specified by the following meta-rule:

```
if      not false(candidate-for-extension(P: PARAMETER))
then    to-assume(candidate-for-extension(P:PARAMETER),positive);
```

Using reflective reasoning based on this meta-rule within `extension-suitability-determination` positive facts of the form `candidate-for-`

`extension(P: PARAMETER)` are postulated. For the VT task, it is not necessary to make a further distinction among the candidates thus established: any candidate is suitable. This is expressed in the following rule:

```
if      candidate-for-extension(P: PARAMETER)
then    suitable-for-extension(P: PARAMETER);
```

The knowledge bases specified above are knowledge structures referenced within the specification of `extension-suitability-determination` in the VT task model. The component `extension-suitability-determination` is a composed component. It contains three sub-components, that determine, respectively, non-candidate parameters, assumptions on candidate parameters, and parameters suitable for extension. In each of these sub-components, references are specified to the related knowledge structures.

Note that these signatures and knowledge bases abstract from the elevator domain. They describe task-specific terms and knowledge that can be used for the task in different domains. As such they can be viewed as part of the problem solving method.

To be of use in a specific domain such as the elevator domain, however, additional domain knowledge is required: domain-specific signatures and knowledge bases need to be defined. They are discussed in more detail in Section 5.

4.5. TASK DELEGATION

In Yost's document not much attention is paid to the distinction between tasks for the user and tasks for the system. In our task model, not only does the user provide initial input, namely customer requirements, but the user may also be consulted on other possible values for a requirement, if the given requirement can not be fulfilled. The same holds for the design object description modification—consulting the user for an appropriate value is an option which may be considered during design.

5. Domain ontology

A domain ontology is a definition of the terms and the relations used in the specification of domain knowledge. An existing generic ontology may be used to guide the knowledge acquisition process, as a structure for new knowledge and facts. Such an ontology is generic in the sense that it is expressed in terms independent of the particular application domain (e.g. independent of the elevator domain). The VT ontology for design, written in ONTOLINGUA by Gruber and Runkel (1993), distinguishes the *generic terms* parameter, value, formula, and constraint. For the domain at hand, the elevator domain, these generic terms are instantiated with *elevator domain specific terms*: specific names of parameters, constraints, etc. These generic terms and elevator domain specific terms are called *domain-oriented.*

In addition to domain-oriented structures, structures for (problem solving) process-oriented or *task-oriented* notions distinguished by Yost (1994) and by Brazier *et al.* (1994*b*) in the generic task model of design are specified; e.g. violated constraints, or parameters in focus (for extension or modification), fixes.

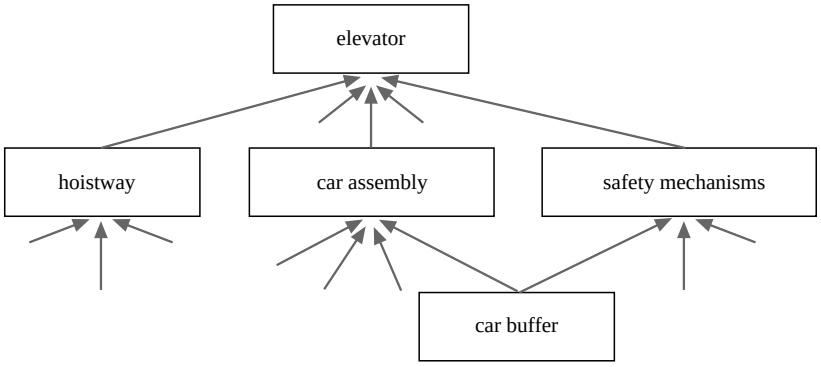

FIGURE 7. Part of the part-of hierarchy for an elevator (arrows indicate "part-of" relations).

5.1. ONTOLOGIES

Ontologies are often expressed in terms of concept hierarchies and concept-attribute-value structures. In the elevator domain, both types of structures have been distinguished.

The main organizational structure used by Yost (1994) is a *concept hierarchy* in terms of components of an elevator (not to be confused with components of a task model): `hoistway`, `car assembly`, `counterweight assembly`, `suspension`, `safety mechanisms`, and `cables`. Each of these components is descriebd in terms of its sub-components. For example, the `car assembly` consists of a `passenger cab`, a `supporting structure` (with sub-components `platform` and `sling`), and `safety mechanisms`. Part of the resulting concept hierarchy is depicted in Figure 7.

In this conceptual structure components are not necessarily disjoint (i.e. the concept hierarchy is not a tree). Yost (1994) states in Section 2 that the `safety mechanisms` component, for instance, is a sub-component of the `elevator`. Yost also mentions: "*The car assembly consists of the* passenger cab, *its supporting structure, and safety mechanisms*" together with "*There is at least one buffer under each of the* car *and* counterweight." This has been interpreted as stating that the `car buffer` is part of the `safety mechanisms` as well as part of the `car assembly`.

The second type of structure often used to specify ontologies is the *concept-attribute-value* structure. The concept `car assembly`, for example, has a number of attributes, each of which can be assigned a value. This is shown in Figure 8.

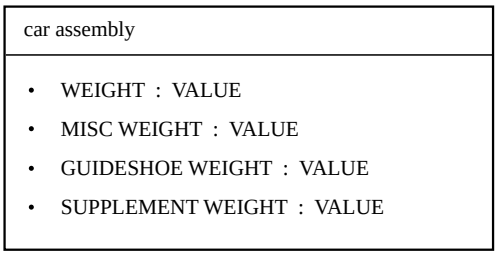

FIGURE 8. The concept `car assembly` and four of its attributes.

The component structure components-with-constraints of Gruber and Runkel (1993) is based on the DIDS knowledge base for VT, which in turn is tuned to the problem-solving method used: configuration-design (Runkel & Birmingham, 1994). In the ONTOLINGUA ontology, all sub-components of an elevator are part of the elevator component; sub-components themselves do not have parts. This structure does not correspond to the component structure partially depicted in Figure 7, which is based on Yost (1994).

The component structure components-with-constraints seems to be based on the additional assumption that a constraint related to a component only refers to parameters of that same component. This would explain, for instance, that almost every parameter is related to the elevator component (which contains 119 parameters), including for example CAR MISC WEIGHT (see Section 5.6 of Yost (1994) on the car assembly). In other words, in the ONTOLINGUA ontology, parameters are organised in an object-oriented fashion as slots of components.

In the procedure for the VT task described by Yost (1994), knowledge of concepts and their relations is not used in the process of generating an elevator configuration. All domain knowledge and constraints are expressed in terms of values of concept attributes. Therefore, the concept hierarchy need not be explicitly modelled and specified for the VT task, but may be useful for other tasks for which the same domain knowledge could be applied (reuse of ontologies).

Gruber and Runkel (1993) represent all knowledge about the domain as constraints, whether used for derivation of attribute values or for imposing restrictions on attribute values. Both in the generic task model of design and in the VT task model presented here, a distinction is made between object-level domain knowledge that does hold for any design (and can be used by object-level reasoning to derive further properties of the design object description) and meta-level domain knowledge that expresses what should hold for any design (and can be used by meta-level reasoning to analyse the current design object description). The first type of knowledge is used in the component DOD-deductive-refinement, the second type in the component DOD-modification (both within DOD-manipulation).

5.2. SPECIFICATION OF ONTOLOGIES IN DESIRE

Ontologies are specified in DESIRE as knowledge structures, defined in terms of signatures for order-sorted predicate logic. In this section, example specifications are presented of domain-oriented ontologies (in Section 5.2.1) and task-oriented ontologies (in Section 5.2.2).

5.2.1. Domain-oriented ontology

A feature of DESIRE, shown in Section 4.1.2, is that signatures may be constructed through reference. For example, generic signatures may refer to signatures specifying domain-specific instances, and vice versa. This enables the separation of generic knowledge structures from domain-specific knowledge structures. In the following signature specifications, a generic concept hierarchy is specified by means of the signature Concept-hierarchy-sig, attributes of a concept by Concept-attribute-sig, and values of concept attributes by Concept-attribute-value-sig. Each of the generic signatures below refers to other generic signatures:

Concept-sort-sig, Attribute-sort-sig, and Value-sig (the latter is specified later).

```
signature Concept-sort-sig
  sorts
    CONCEPT;
end signature

signature Concept-hierarchy-sig
  signatures
    Concept-sort-sig, Concept-object-sig;
  relations
    part-of: CONCEPT*CONCEPT;
end signature

signature Attribute-sort-sig
  sorts
    ATTRIBUTE:
end signature

signature Concept-attribute-sig
  signatures
    Concept-sort-sig, Attribute-sort-sig, Concept-object-sig,
    Attribute-object-sig;
  relations
    concept-attribute: CONCEPT*ATTRIBUTE;
end signature

signature Concept-attribute-value-sig
  signatures
    Concept-sort-sig, Attribute-sort-sig, Value-sig, Concept-object-sig,
    Attribute-object-sig;
  relations
    concept-attribute-value: CONCEPT*ATTRIBUTE*VALUE;
end signature
```

Within the generic signature specifications above also references are made to the domain specific signatures `Concept-object-sig` and `Attribute-object-sig`. For the elevator domain these are specified by:

```
signature Concept-object-sig
  signatures
    Concept-sort-sig;
  objects
    elevator, hoistway, car-assembly, safety-mechanisms, car-buffer, . . .:
    CONCEPT;
end signature

signature Attribute-object-sig
  signatures
    Attribute-sort-sig;
  objects
    model, weight, height, length, thickness, . . .: ATTRIBUTE;
end signature
```

This illustrates how in DESIRE a separation can be made between generic ontologies and (elevator) specific instances of them.

A concept and an attribute together uniquely identify (and determine the name of) a parameter of the elevator configuration. As argued in Section 5.1, concept hierarchies need not be explicitly modelled and specified for the VT task described by Yost (1994). In the VT task model specified in DESIRE, only parameters and values are used to describe elevator configurations. This choice is supported by the fact that Yost (1994) also names parameters on the basis of the concepts and attributes to which they refer.

In the domain-oriented part of the ontology in DESIRE, a sort `PARAMETER` (see the generic signature `Parameter-sort-sig` in Section 4.1.2) is defined for the purpose of modelling and specifying the parameters distinguished by Yost (1994). In the example below, a reference is made in the domain-specific signature `Parameter-object-sig` to the generic signature `Parameter-sort-sig`. The signature `Parameter-object-sig` specifically instantiates the sort `PARAMETER` by elevator-domain specific instances of parameters.

```
signature Parameter-object-sig
  signatures
     Parameter-sort-sig;
  objects
    car-weight,     door-speed,     hoistway-depth,     sling-model, . . .;
    PARAMETER;
end signature
```

The four types of value that can be assigned (see Figure 9) to parameters are as follows (with reference to Yost, 1994).

- `INTEGER` for values of parameters such as OPENING COUNT (Section 3: "*The number of floors the elevator will stop on.*").
- `REAL` for values of parameters such as PLATFORM RUNNING CLEARANCE (Section 4.4: "[The PLATFORM RUNNING CLEARANCE] *is 1.25 inches.*").
- `BOOLEAN` for values of parameters such as CAR LANTERN (Section 3: "*Whether or not the car should be equipped with a lantern* (*yes or no*).").
- `STRING` for values of parameters such as DOOR MODEL (Section 5.1: "*For side-opening doors, the* [*DOOR MODEL*] *consists of the DOOR MODEL CODE followed by a code identifying the DOOR OPENING STRIKE SIDE . . . , with the two codes separated by a dash.*").

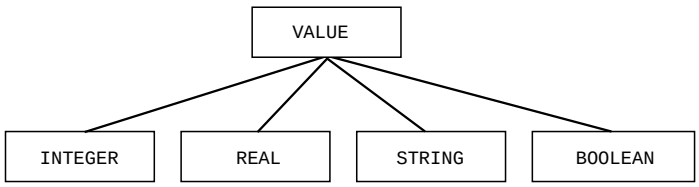

FIGURE 9. Value type hierarchy (supertype-sub-type links are from top to bottom).

In the signature `Value-sig` for which the specifications are depicted below, sub-sorts are used to model and specify the relation between `VALUE` and the four more specific value types. The signature `Parameter-value-sig` defines a relation expressing which value is assigned to which parameter.

```
signature Value-sig
  signatures
    Integer-sig, Real-sig, String-sig, Boolean-sig;
  sorts
    VALUE;
  subsorts
    INTEGER, REAL, STRING, BOOLEAN < VALUE;
end signature
```

```
signature Parameter-value-sig
  signatures
    Parameter-sort-sig, Parameter-object-sig, Value-sig;
  relations
    value-of: PARAMETER*VALUE;
end signature
```

For the specification of the sorts `INTEGER` and `REAL`, the objects `0` and `1`, arithmetic comparison relations such as $=$ and $\leq$ and arithmetic functions such as $+$, $-$, and $*$ are assumed as usual.

Part of the VT domain knowledge can be used to deduce additional parameter values from those already known. For example, Yost (1994) states in Section 4.3 that "[the COUNTER-WEIGHT STACK HEIGHT] *is equal to the number of counterweight plates* (*COUNTERWEIGHT PLATE QUANTITY*) . . . *times the individual plate thickness* . . . (*COUNTERWEIGHT PLATE THICKNESS*)." This can be expressed in DESIRE by means of the following knowledge base rule (note that the choices of this rule implies that the parameter in the conclusion depends on the parameters in the condition):

```
if value-of(counterweight-plate-quantity, PQ: VALUE)
  and value-of(counterweight-plate-thickness, PT; VALUE)
  and SH: VALUE=PQ: VALUE*PT: VALUE
then value-of(counterweight-stack-height, SH: VALUE);
```

The constraints described in Section 7 of Yost (1994) also need to be modelled and specified. A typcial constraint is constraint C-22: "*The COUNTERWEIGHT STACK HEIGHT . . . can be at most the COUNTERWEIGHT FRAME HEIGHT . . . minus the COUNTERWEIGHT FRAME THICKNESS . . .*; *if it is not, three fixes are possible*: . . ." According to Section 7 of Yost (1994), each constrain focuses on a specific parameter: this parameter is explicitly mentioned in the introduction of the constraint. A test on the value of this parameter is expressed as a `WFF` (see Figure 10). Three kinds of constraints can be distinguished: *minimum, maximum,* and *compatibility constraints.* A minimum constraint sets a lower limit on

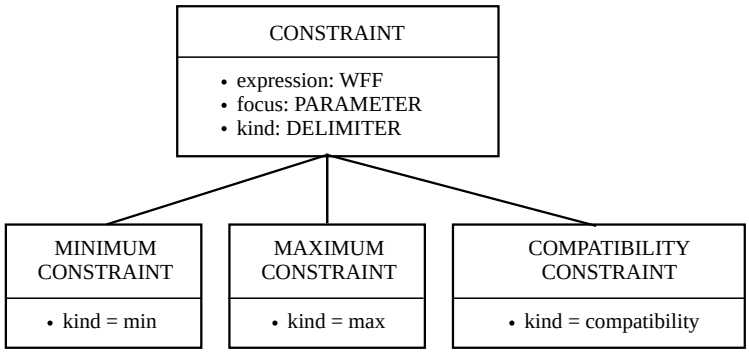

FIGURE 10. Constraint type hierarchy.

the value of a specific parameter, a maximum constraint sets an upper limit. For example, constraint C-22 cited above is a maximum constraint. A compatibility constraint limits the possible values of a parameter to a finite, enumerated set of values. For example, compatibility constraint C-34 states about MACHINE MODEL 18 that "*MACHINE MODEL 18 is compatible with MOTOR MODELS 10HP And* 15*HP.*" These three types of constraints can be modelled and specified in DESIRE in a sort hierarchy (sorts and sub-sorts) and/or by relations expressing the specific types of constraint. Figure 10 depicts both options.

In the VT task model, relations have been used to model and specify the three different types of constraints. Relations provide a more constraint. Figure 10 depicts both options.

In the VT task model, relations have been used to model and specify the three different types of constraints. Relations provide a more flexible representation: the type of a constraint can be derived dynamically. An example of a specification in which the type of constraint is specified is the signature `Constraint-attribute-sig` shown below.

```
signature Constraint-sort-sig
  sorts
    CONSTRAINT;
end signature

signature Delimiter-sig
  sorts
    DELIMITER;
  objects
    min, max, compatibility: DELIMITER;
end signature

signature Constraint-attribute-sig
  signatures
    Constraint-sort-sig, Parameter-sort-sig, Constraint-object-sig,
    Parameter-object-sig,
```

```
    Delimiter-sig, Wff-sig;
  relations
    kind: CONSTRAINT*DELIMITER;
    focus: CONSTRAINT*PARAMETER;
    expression: CONSTRAINT*WFF;
end signature

signature Parameter-value-fact-sig
  signatures
    Parameter-sort-sig, Parameter-object-sig, Value-sig;
  sorts
    PARAMETER-VALUE-FACT;
  functions
    value-of: PARAMETER*VALUE → PARAMETER-VALUE-FACT;
end signature
```

The meta-level function `value-of` is used to represent at the meta-level, the object-level relation `value-of` with the same arity. The arithmetic comparison relations at the object-level are represented at the meta-level in a similar way. The following signature defines the elevator-specific instances:

```
signature Constraint-object-sig
  signatures
    Constraint-sort-sig;
  objects
    eligible-motor-model,      max-machine-groove-pressure,      max-hoist-
    cable-traction-ratio,
    max-car-guiderail-vertical-force,              min-hoist-cable-safety-
    factor, . . .: CONSTRAINT;
end signature
```

Using the above signatures, knowledge about constraint C-22 can be expressed as follows:

```
kind(max-counterweight-stack-height, max)
focus(max-counterweight-stack-height, counterweight-stack-height)
expression(max-counterweight-stack-height,
  value-of(counterweight-stack-height, SH)
  and value-of(counterweight-frame-height, FH)
  and value-of(counterweight-frame-thickness, FT)
  implies SH ≤ FH − FT)
```

where `and` and `implies` are both functions from `WFF×WFF` to `WFF` (written in-fix), denoting logical conjunction and logical implication, respectively.

5.2.2. Task-oriented ontology

The representation of concepts by means of which the VT domain is described in the previous section is influenced by the task to be performed. The task determines not only which information about the domain is to be represented but also the terminology to be employed. In the VT task, for example, colour is irrelevant and therefore not included in the domain oriented ontology. Characteristics and

dimensions of elevator components are described in terms of parameters, which is motivated by the view of an elevator as a configuration. These observations agree with the argument put forward by Vanwelkenhuysen and Mizoguchi (1995) that domain knowledge cannot be adequately represented independent of the class of tasks for which it has been designed.

The domain-oriented ontologies presented in Section 5.2.1 to a certain extent reflect the task, but their semantics are based on the domain. However, besides domain-oriented ontologies, also concepts with process aspects of the task as their semantics are required: task-oriented ontologies. Notions like fixes are task-oriented and are part of the knowledge structures that are used to instantiate a problem-solving method.

Task-oriented information includes meta-level relations that express dynamic properties of the design task such as violations of constraints, applied fixes, and tentative parameter values. Task-oriented relations are represented in generic signatures, such as `Suitable-parameter-sig` introduced in Section 4.1.2 (see also Sections 4.2 and 4.4). The following signatures illustrate the use of meta-level relations to denote the contents of the current and the tentative configurations.

```
signature Configuration-content-sig
  signatures
    Parameter-value-fact-sig:
  relations
    in-current-configuration, in-tentative-configuration: PARAMETER-VALUE-
    FACT;
end signature
```

Meta-relations are also used to specify which parameter values are required by the user (Section 3 of Yost, 1994): customer specifications and relevant building information. These relations represent the requirements for the VT task. In addition, initial values for parameters are encoded by meta-relations, representing heuristics about plausible designs.

The following knowledge about fixes is modelled and specified (see Figure 11).

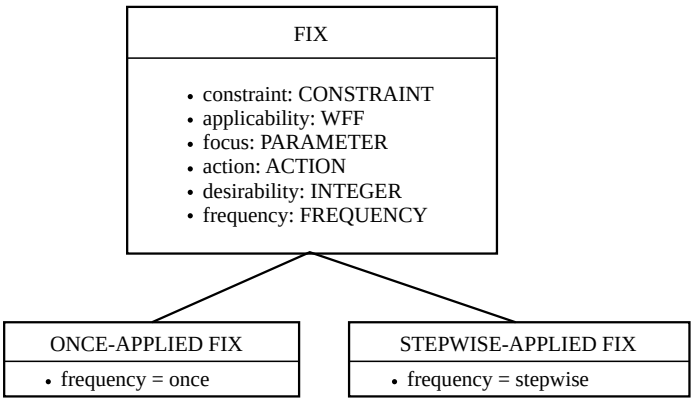

FIGURE 11. Fix type hierarchy.

- The constraint in relation to which the fix can be considered (in order to resolve the constraint's violation).
- A condition stating when the fix is applicable, given a violation of its related constraint.
- The parameter on which the fix focuses (i.e. of which the value is changed by the fix).
- The action to perform on (i.e. the change in value of) the parameter in focus when the fix is applied.
- The desirability of the fix.
- An indication of whether the fix may change the value of the parameter in focus once or repeatedly.

The applicability condition of a fix is expressed in terms of parameters and their current values. For example, in Section 7, Yost (1994) mentions a fix to constraint C-34 with the following applicability condition: "*the MACHINE MODEL is 18, or the MACHINE MODEL is 28 and the MOTOR MODEL is 25HP, 30HP, or 40HP.*"

Actions mentioned in Section 7 of Yost (1994) can be divided into *upgrade actions* ("*. . . try upgrading the SAFETY BEAM MODEL . . .*"), *increase actions* and *decrease actions* ("*. . . either increase the CAR CAB HEIGHT by the amount of the constraint violation, or decrease the OPENING HEIGHT by the amount of the constraint violation.*"). These different types are shown in Figure 12. Changing values of parameters consists of one of these actions.

The desirability of a fix, at constant encoded as an integer, guides the construction of fix combinations as described in Section 7 of Yost (1994). Although not stated explicitly, the desirability of a fix seems to depend on the type of parameter that is to be changed. Fixes to parameters of which the value is prescribed by requirements (customer specifications and relevant building information) have a desirability of `D6`, `D9` or `D10`. Whenever a fix tentatively changes a value prescribed by a requirement, the customer may be consulted about the change. The trace shown in

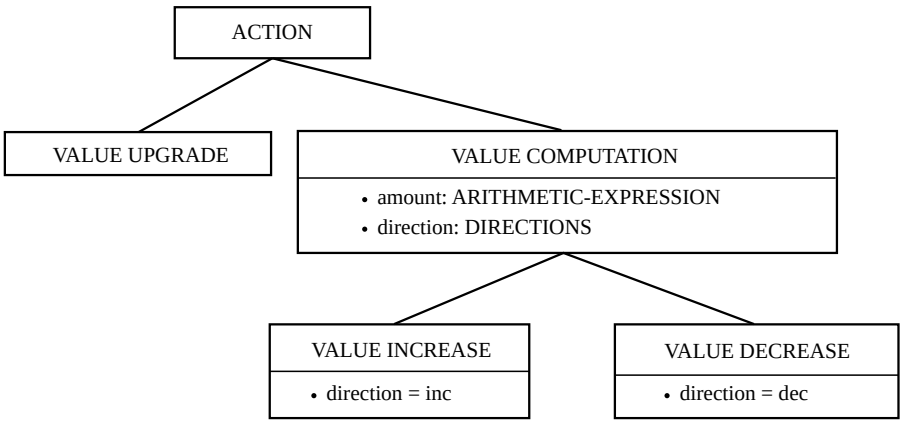

FIGURE 12. Action type hierarchy.

Table 2 (Section 6 of this paper) presents an example of consulting the customer in order to resolve a constraint violation.

Similar signatures as for constraints are provided to model and specify fixes. For example, one of the fixes to the violation of constraint C-22 is to increase the COUNTERWEIGHT PLATE DEPTH by half-inch steps. This can be expressed as follows:

```
constraint(fix-22-1, max-counteweight-stack-height)
applicability(fix-22-1, true)
focus(fix-22-1, counterweight-plate-depth)
action(fix-22-1, increase(0.5))
desirability(fix-22-1, 3)
frequency(fix-22-1, stepwise)
```

(where `true` is a tautology; i.e. a trivial or empty applicability condition).

Closely associated with these fixes are fix combinations. The current combination is encoded in a unary relation `in-combination` on `FIX`: it is expressed as a set of atoms `in-combination(F)` for all fixes `F` in the combination. Some of these fixes may be stepwise-applied fixes. The number of times a fix step has been applied is encoded in the binary relation `number-of-fix-steps-tried: FIX*INTEGER`. From these two relations, the actual modification can be formulated: for each fix in the current combination, the number of fix steps applied determines the next fix step to be applied, represented by the binary relation `current-design-modification: INTEGER*ACTION`.

Other types of objects distinguished are the dependencies between parameters (i.e. which parameters are needed to compute a specific parameter), between parameters and constraints (i.e. which parameters are involved in a constraint), and between parameters and fixes (i.e. which parameters are involved in a fix). These dependencies are represented by the relations `dependent-on: PARAMETER*PARAMETER`, `involved-in-constraint: PARAMETER*CONSTRAINT`, and `involved-in-fix: PARAMETER*FIX`. The first two relations correspond to the `USED-IN` attribute in Gruber and Runkel (1993). The third relation is task-specific knowledge that is not considered in Gruber and Runkel (1993). All three relations are needed to be able to derive just enough parameters to evaluate the results of applying fixes: ''*Recompute just enough values to find out if* [the applied fix is acceptable or not],'' as stated in Section 7 of Yost (1994).

6. Sample trace

In this section, excerpts from a sample trace are presented that have been produced by a prototype system automatically generated from the full DESIRE specification of the VT task model. Parts of this specification have been presented in Sections 4 and 5 of this paper. The test case described in Section 9 of Yost (1994) has been reproduced with the prototype system: for the given sample customer specifications and relevant building information, the same constraint violations were detected, the same design modifications were applied, and the same final configuration resulted.

Note that our aim has been to preserve the expert's knowledge described by Yost (1994) as much as possible, rather than to impose a particular problem-solving

method onto the task and neglect the expert's knowledge. The refinement of the generic task model of design leading to the VT task model has been motivated by statements from Yost (1994). Furthermore, to our opinion all of the expert's knowledge has been modelled and specified in the VT task model.

Table 1 shows the violated constraints that were detected during the design process, together with the fixes (design modifications) that were applied to remove these violations, for the test case provided in Section 9 of Yost (1994). The left column shows the constraints violated, including the one selected for fixing (in italics), the middle column shows the design modification tried, and the right column indicates whether or not the design modification was accepted.

Rather than showing the activations of all the components of the VT task model, two detailed examples are presented.

- Fixing of the `min-platform-to-hoistway-left` constraint violation, which illustrates the cooperation between `DOD-manipulation` and `RQS-manipulation`.
- Fixing of the `max-traction-ratio` constraint violation, which illustrates various activations of components within `DOD-manipulation`.

For each example a table is presented, containing a sequence of activations of components, together with the results of activations (using an abbreviated notation). In the left column, components are separated from subcomponents by means of colons.

In the first example, the constraint `min-platform-to-hoistway-left` is violated and a fix to resolve this violation is proposed. However, this fix is not immediately accepted, as it changes a value protected by a requirement. Instead, `RQS-manipulation` is activated and the customer is asked whether or not he or she can accept the proposed change. The customer agrees and the requirement is changed accordingly, after which `DOD-manipulation` is allowed to continue.

The second example focuses on resolving the violation of the constraint `max-traction-ratio`. Several fix combinations have to be tried to resolve this constraint violation. For the test case descirbed in Section 9 of Yost (1994), solving this particular constraint violation is the most extensive part of the design process. To resolve this constraint violation, a combination of three fixes is needed. As the complete component activation sequence is quite extensive, only a small fragment will be shown.

7. Implementation aspects

As discussed above, the DESIRE framework includes tools which can be used to generate a prototype implementation from a formal specification. A number of these tools have been used in the course of the VT project.

The implementation generator *impl* was used to generate executable prototype code for a UNIX/Prolog environment. One consequence of modelling the VT task is that efficiency within the DESIRE framework could (and had to) be greatly improved, in particular for the implementation generator and executor.

The general manager *gm* for this environment was used to run the prototype code. This tool also provides facilities for the developer to examine the results of the reasoning process per component. Through communication with *aid,* a graphical

TABLE 1
Violated constraints and design modifications in the design process

Violated constraints (selections in italics)	Design modification			
	Focus parameter	Step	Action	Accepted
min-platform-to-hoistway-left	opening-to-hoistway-left	1	increase by (8− platform-to-hoistway-left)	yes
eligible-motor-model	machine-model	1	upgrade	yes
max-vertical-rail-force	car-railunit-weight	1	upgrade	yes
min-hoist-cable-safety-factor	hoist-cable-quantity	1	increase by (5− hoist-cable-quantity)	yes
max-traction-ratio	cwt-to-platform-rear	1	decrease by 0.5	no
max-traction-ratio	cwt-to-platform-rear	2	decrease by 0.5	no
...	...	...	...	...
max-traction-ratio	cwt-to-platform-rear	7	decrease by 0.5	no
max-traction-ratio min-cwt-to-platform-rear	car-supplement-weight	1	increase by 100	no
...	...	...	...	...
max-traction-ratio	car-supplement-weight	6	increase by 100	no
max-traction-ratio max-car-supplement-weight	car-supplement-weight cwt-to-platform-rear	1 1	increase by 100 decrease by 0.5	no
...	...	...	...	...
max-traction-ratio	car-supplement-weight cwt-to-platform-rear	1 7	increase by 100 decrease by 0.5	no
max-traction-ratio min-cwt-to-platform-rear	car-supplement-weight cwt-to-platform-rear	2 1	increase by 100 decrease by 0.5	no
...	...	...	...	...
max-traction-ratio	car-supplement-weight cwt-to-platform-rear	5 7	increase by 100 decrease by 0.5	no
max-traction-ratio min-cwt-to-platform-rear	car-supplement-weight cwt-to-platform-rear	6 1	increase by 100 decrease by 0.5	no
max-traction-ratio max-car-supplement-weight	comp-cable-model	1	upgrade	no
max-traction-ratio	comp-cable-model cwt-to-platform-rear	1 1	upgrade decrease by 0.5	no
...	...	...	...	...
max-traction-ratio	comp-cable-model cwt-to-platform-rear	1 8	upgrade decrease by 0.5	no
max-traction-ratio min-cwt-to-platform-rear	comp-cable-model car-supplement-weight	1 1	upgrade increase by 100	no
...	...	...	...	...
max-traction-ratio	comp-cable-model car-supplement-weight	1 6	upgrade increase by 100	no
max-traction-ratio max-car-supplement-weight	comp-cable-model car-supplement-weight	1 1	upgrade increase by 100	no
	cwt-to-platform-rear	1	decrease by 0.5	
...	...	...	...	...
max-traction-ratio	comp-cable-model car-supplement-weight cwt-to-platform-rear	1 5 7	upgrade increase by 100 decrease by 0.5	yes
min-machine-beam-selection-modulus	machine-beam-model	1	upgrade	yes

editor for DESIRE specifications of which a prototype has been developed, this examination can be done graphically. A first optimization of both *impl* and *gm* decreased the time required for prototype generation and execution considerably.

8. Discussion

In this discussion first the design model presented in Section 3 is compared to other design models, then the formal specification language DESIRE is compared to other languages with formal specifications and finally the results presented in this paper are discussed.

8.1. COMPARISON WITH OTHER DESIGN MODELS AND THEORIES

Researchers from various disciplines (such as architecture, mechanical engineering, industrial design, and artificial intelligence) have developed many different models and theories of design. These models and theories describe characteristic features of design such as the following.

- Design problem statements (function, need, desire, goal, objective, required behaviour, requirement, constraint).
- Design objects (structure, attribute, property, behaviour).
- Design paradigms (decomposition, case-based design).
- Design knowledge (domain knowledge, strategic knowledge).
- Decomposition of the design task into sub-tasks.
- Input for and output of sub-tasks.
- Kinds of knowledge (and possibly also suitable inference mechanisms) for sub-tasks.
- Control of sub-tasks.

In the following, some of the well-known and/or more recent models and theories will be compared with the generic task model of design described in Section 3, discussing two of the five types of knowledge distinguished in Section 2: task (de)composition and task control knowledge. For the sake of brevity, the generic task model of design will be referred to as GTMD.

8.1.1. Task (de)composition

Design models often include a decomposition of the design task into sub-tasks. These sub-tasks may be further decomposed, but most design models contain a one-level decomposition only.

Archer (1970) decomposes design into 30 sub-tasks (in his terminology ''steps''), of which 12 describe the generation of requirements and 17 the generation of a design object description. One sub-task describes the evaluation of the design process, as do a few others partially. Examples of sub-tasks in Archer's model are as follows.

- Selection of the next sub-problem to handle.
- Identification of goals (i.e. requirements of properties of the design object) within the selected sub-problem.
- Analysis of the relationships between the states of properties of the design object and the fulfilment of the identified goals.

TABLE 2
Trace of component activations and results for fixing the `min-platform-to-hoistway-left` ***constraint***

Activation of component	Results
Within *`DOD-manipulation: DOD-modification:`*	
`DOD-modification-analysis: constraints-analysis`	Violated constraints: `min-platform-to-hoistway-left.`
`DOD-modification-determination: DOD-revision-determination: violated-constraint-selection`	Selected constraint: `min-platform-to-hoistway-left.`
`DOD-modification-determination: DOD-revision-determination: fix-combination-determination`	Determined fix combination: `increase opening-to-hoistway-left by (8 − platform-to-hoistway-left)`
`DOD-modification-determination: DOD-revision-determination: fix-steps-determination`	Determined next fix steps: `increase once opening-to-hoistway-left by` (8 − platform-to-hoistway-left).
`DOD-modification-implementation: implementation-focus-determination`	Parameters to recompute, among which: `opening-to-hoistway-left.`
`DOD-modification-implementation: fix-steps-application`	Tentative parameter value: `value-of(opening-to-hoistway-left, 33).`
`DOD-modification-analysis: fix-execution-analysis: configuration-status-determination`	The selected constraint is now satisfied, and no new constraint violations were introduced.
`DOD-modification-analysis: fix-execution-analysis: fix-steps-acceptability-determination`	The requirement on `opening-to-hoistway-left` is not met by the tentative configuration, so a dead end is reached.
Within *`DPC:`*	
`design-process-coordination`	Because no further progress can be made in devising a DOD, the current set of requirements has to be modified.
Within *`RQS-manipulation: RQS-modification:`*	
`RQS-modification-analysis untenable-requirements-analysis`	Untenable requirement: `value-of(opening-to-hoistway-left, 32).`
`RQS-modification-determination: RQS-revision-determination`	Requirement dropped: `value-of(opening-to-hoistway-left, 32).`
`RQS-modification-determination: RQS-extension-determination: extension-method-determination`	Selected extension determination method: `consult-customer.`
`RQS-modification-determination: RQS-extension-determination: user-requirement-acquisition`	New requirement (based on the last design modification tried): `value(opening-to-hoistway-left, 33).`
Within *`DPC:`*	
`design-process-coordination`	Because the current set of requirements has been modified, further attempts must be made to devise a design object description.
Within *`DOD-manipulation: DOD-modification:`*	
`DOD-modification-analysis: fix-execution-analysis: fix-steps-acceptability-determination`	The last design modification is acceptable.
`DOD-modification-implementation: configuration-update`	The values of all dependent parameters which have not yet been recomputed are now removed from the current configuration.
Within *`DOD-manipulation:`*	
`DOD-deductive-refinement`	The values of all dependent parameters are deduced.

- Identification of decision variables on which the states of properties of the design object depend.
- Assignment of states to the identified decision variables.
- Reiteration of the problem-solving process until the overall problem is resolved.

The GTMD can be further refined to specify Archer's model. For instance, of the sub-tasks above, the first two can be modelled as sub-components of `RQS-modification`, the third one as `RQS-deductive-refinement`, the fourth and fifth one as sub-components of `DOD-modification`, and the last one as `design-process-evaluation`.

Akin (1978) distinguishes the following sub-tasks (in his terminology ''mechanisms'') of architectural designs.

- Information acquisition, which collects external information about aspects of the design problem.
- Information interpretation, which expands the implications of the incoming information to the various aspects of the design problem at hand.
- Information storage, which stores past actions in design problem-solving to aid future actions.
- Partial-solution generation, which produces a solution to one or a few aspects of the total set of design requirements.
- Solution evaluation, which checks each new partial solution against the criteria used in generating all previous partial solutions.
- Solution integration, which combines all partial solutions into one overall solution, provided that the partial solutions do not conflict with the criteria.
- Input–output mechanisms, which aid in the presentation of input for and output from the design process, as part of both problem formulation and solution generation.

The GTMD can be extended to model information acquisition by the `modification` components of both `RQS-manipulation` and `DOD-manipulation`, information interpretation by the two respective `deductive-refinement` components, and information storage by the respective `update-of-modification-history` components. Partial-solution generation, solution evaluation, and solution integration can be modelled by sub-components of `DOD-modification`. Input–output mechanisms are not considered explicitly in the GTMD, but could be modelled as sub-components of the `modification` components of both `RQS-manipulation` and `DOD-manipulation`.

Coyne (1988) identifies the following sub-tasks of design.

- Interpretation of designs (by deduction).
- Derivation of interpretative knowledge (by induction, producing logical rules).
- Delimitation of a space of designs conforming to a set of interpretations (by abduction).

In the GTMD, interpretation is modelled by `DOD-deductive-refinement`. Delimitation of a space of designs is specified in the GTMD as deductive reasoning on the meta-level with respect to the design object description, resulting in

TABLE 3
Trace of component activations and results for fixing the `max-traction-ratio` *constraint*

Activation of component	Results
Within `DOD-manipulation: DOD-modification:`	
`DOD-modification-analysis: constraints-analysis`	Violated constraints: `max-traction-ratio.`
`DOD-modification-determination: DOD-revision-determination: violated-constraint-selection`	Selected constraint: `max-traction-ratio.`
`DOD-modification-determination: DOD-revision-determination: fix-combination-determination`	Determined fix combination: `decrease cwt-to-platform-rear by 0.5.`
`DOD-modification-determination: DOD-revision-determination: fix-steps-determination`	Determined next fix steps: `decrease cwt-to-platform-rear 1 time by 0.5.`
`DOD-modification-implementation: implementation-focus-determination`	Parameters to recompute, among which: `cwt-to-platform-rear.`
`DOD-modification-implementation: fix-steps-application`	A tentative configuration, including: `value-of(cwt-to-platform-rear, 4.75).`
`DOD-modification-analysis: fix-execution-analysis: configuration-status-determination`	Violated constraints: `max-traction-ratio.`
`DOD-modification-analysis: fix-execution-analysis: fix-steps-acceptability-determination`	The fix step is not acceptable.
`DOD-modification-analysis: fix-execution-analysis: fix-failure-determination`	Further fix steps for the given fix combination are possible.
`DOD-modification-determination: DOD-revision-determination: fix-steps-determination`	Determined next fix steps: `decrease cwt-to-platform-rear 2 times by 0.5.`
The same scenario develops, until:	
`DOD-modification-determination: DOD-revision-determination: fix-steps-determination`	Determined next fix steps: `decrease cwt-to-platform-rear 7 times by 0.5.`
`DOD-modification-implementation: implementation-focus-determination`	Parameters to be recomputed, among which: cwt-to-platform-rear.
`DOD-modification-implementation: fix-steps-application`	A tentative configuration, including: `value of(cwt-to-platform-rear, 1.75).`
`DOD-modification-analysis: fix-execution-analysis: configuration-status-determination`	Violated constraints: `max-traction-ratio, min-cwt-to-platform-rear.`
`DOD-modification-analysis: fix-execution-analysis: fix-steps-acceptability-determination`	The fix step is not acceptable.
`DOD-modification-analysis: fix-execution-analysis: fix-failure-determination`	Further stepping will never resolve the violation of `min-cwt-to-platform-rear.`
`DOD-modification-determination: DOD-revision-determination: fix-steps-determination`	No more fix steps are possible for the given fix combination.
`DOD-modification-determination: DOD-revision-determination: fix-combination-determination`	Determined fix combination: `increase car-supplement-weight by 100.`
And so forth.	

assumptions on (parts of) the design object. This type of reasoning is modelled by the `DOD-modification` component. Derivation of interpretative knowledge is not addressed by the GTMD.

Brown and Chandrasekaran (1989) distinguish the following design sub-tasks, as part of a family of methods called propose-verify-redesign.

- Design problem decomposition.
- Design plan generation (i.e. precompiled partial solutions to design (sub)goals).
- Design proposal by critiquing and modifying almost correct designs.
- Design proposal by constraint satisfaction.
- Goal/constraint propagation to sub-problems.
- Recomposition (i.e. to glue partial solutions of the sub-problems back into a solution of the original problem).
- Design verification.
- Design criticism.

Most of these sub-tasks are further discussed by Chandrasekaran (1990); he categorizes them as sub-tasks of the propose-critique-modify family of methods. Another common name for this family is propose-and-revise.

In the GTMD,

• Testing whether the proposed solution necessitates additional structural modifications.

In the GTMD, both the diagnosis sub-task and the repair sub-task can be specified by decompositions of `DOD-modification`. In fact, part of the decompositions given in Section 4 can be used: an alternative decomposition of `DOD-modification-analysis` can be used for the diagnosis sub-task and `DOD-modification-determination` can be tuned to the repair sub-task.

Gero (1990) defines a model of the design process in which the following sub-tasks (in his terminology ''activities'') are included.

- Formulation, or specification, which transforms the functions to be achieved to expected behaviours of the design object.
- Synthesis, which generates a structure of the design object on the basis of the object's expected behaviours.
- Analysis, which derives specific behaviours of the design object from its structure.
- Evaluation, which compares the predicted behaviours of the design object's structure with the expected behaviours in order to determine whether the structure is capable of producing the functions to be achieved.
- Reformulation, which changes the expected behaviours of the design object in response to the (successful or failed) synthesis of structures and the analysis of their behaviours.
- Production of the design description, which transforms the design object's structure into a design description (e.g. a collection of drawings and notes).

In the GTMD, formulation and reformulation are modelled by `RQS-modification`, and analysis by `DOD-deductive-refinement`. Synthesis, evaluation, and design description production can be modelled by a decomposition of `DOD-modification`. In fact, the decomposition given in Section 4 can be used in part for this purpose: `DOD-modification-analysis` is meant for evaluation and `DOD-modification-determination` for synthesis.

Maher (1990) distinguishes three main design sub-tasks (''sub-processes'' in her terminology).

- Formulation, which identifies the requirements of the design problem.
- Synthesis, which includes the identification of one or more design descriptions that are consistent with the requirements defined during formulation and additional requirements identified during synthesis.
- Evaluation, which involves interpreting a (partially or completely) specified design description for conformance with the requirements.

Maher further distinguishes three distinct models of design synthesis.

- Decomposition, which divides the design problem into smaller, less complex design sub-problems and recomposes sub-solutions into a solution for the original problem.
- Case-based reasoning, which uses analogical reasoning to select and transform specific solutions to previous design problems to be appropriate as solutions for a specific new design problem.
- Transformation, which uses rules to transform the initial set of design requirements into a design solution.

In the GTMD, the formulation of initial and additional requirements is handled by `RQS-manipulation`. Similar to Gero's (1990) model, synthesis and evaluation can both be modelled as sub-components of `DOD-modification`. Brazier, van Langen, Treur and Wijngaards (1995*b*) show how synthesis by case-based reasoning can be modelled as a decomposition of the GTMD.

Takeda, Veerkamp, Tomiyama and Yoshikawa (1990) have developed a cognitive design model, constructed from unit design cycles of which each consists of five sub-tasks.

- Awareness of the problem by comparing the design object under consideration with the functional specifications.
- Suggestion of key concepts needed to solve the problem.
- Development of candidates for the problem from the key concepts using various types of design knowledge.
- Evaluation of the candidates in various ways.
- Conclusion on which candidate to adopt (and the corresponding modification of the descriptions of the design object).

Note that Takeda *et al.*'s (1990) evaluation sub-task judges design proposals *during* synthesis, whereas in the models of Gero (1990) and Maher (1990) a judgement is made *after* synthesis.

Takeda *et al.* (1990) distinguish two levels in the design process. One is the object level, where the designer thinks about design objects themselves, involving the sub-tasks suggestion of key concepts and development of candidates. The other is the action level, where the designer thinks about how to proceed with the design. This level is linked to the object level by the sub-tasks awareness of the problem, evaluation of candidates and conclusion on the candidate to adopt.

In the GTMD, the above sub-tasks can be modelled by a decomposition of `DOD-modification`. In particular, the decomposition into sub-components for the generation of candidate assumptions on (parts of) the design object, the comparison of candidates, and the selection of candidates is a common way to model the sub-tasks of development, evaluation, and conclusion.

Runkel, Balkany and Birmingham (1994) do not assume a general or generic model of design. Rather, they adapt existing models of specific design tasks for new design tasks in other domains of application. To make a comparison, the mechanisms mentioned in their VT problem-solving metohd (VT-PSM) will be taken as sub-tasks.

- Checking whether there are required functions (in the case of VT, customer specifications and building dimensions) that are not yet realised by the design description.
- Selection of one function that has not yet been realised.
- Generation of a part description that can be used to provide the selected function.
- Addition of the part description to the overall design description, including the fixing of constraint violations that might be introduced as a result of the addition.
- Checking whether all required functions are realised by the design object description.
- Chronological backtracking in case addition of the part description fails (because none of the available fixes could resolve the constraint violations introduced).
- Display of the solution.

The VT task model described in Section 4 differs from Runkel *et al.*'s (1994) VT-PSM. First, our VT task model is more differentiated with respect to the fixing of constraint violations. The sub-tasks involved in fixing constraint violations have been explicitly modelled in our VT task model, primarily by the decomposition of DOD-modification, whereas in the VT-PSM fixing is not made explicit in terms of sub-tasks of the addition of the generated part description.

Second, in our VT task model, checking of the requirements is done in another way. Within DOD-extension-determination, the component user-value-acquisition proposes assignments of values to parameters in accordance with the requirements given by the user. By checking whether all output parameters have been assigned a value (within DOD-modification-analysis), it is then also made sure that all requirements have been met. On the other hand, checking of constraints is not made explicit in the VT-PSM, but an implicit sub-task of the addition of the generated part description. In our VT task model, there are sub-components that specify constraint checking within DOD-modification-analysis.

Third, in our VT task model, there is no specific sub-task for chronological backtracking. Instead, this has been modelled by an interplay of DOD-modification (involving DOD-revision-determination and DOD-modification-implementation) and DOD-update-of-modification-history (for retrieval of previous configurations).

Smithers, Corne and Ross (1994) do not focus on a task decomposition of design, but they mention the following sub-tasks.

- Description of requirements (on the basis of the client's or customer's needs/desires).
- Problem construction (i.e. generation of a well-structured problem statement from an ill-structured requirements description).
- Problem solving (i.e. generation of a satisfactory design object description).
- Requirements revision and modification (in response to the results of problem solving).

The sub-tasks of requirements description, problem construction, and requirements revision and modification are modelled in the GTMD by RQS-manipulation. For problem construction, a decomposition of RQS-modification will be necessary. Furthermore, problem solving is modelled by DOD-manipulation.

Wielinga, Akkermans and Schreiber (1995) also do not focus on a task decomposition of design, but they present a very global model of design problem solving, in which needs and desires (from a client) and informal constraints are analysed (by the designer) to a formal set of requirements and a formal set of constraints. These results of analysis are then used (by the designer) in a synthesis process to develop a structure (the design) consisting of a number of elements with specified properties and relations between them. In the GTMD, analysis is modelled by RQS-manipulation and synthesis by DOD-manipulation.

8.1.2. Task control knowledge

Design models usually incorporate task control aspects. Task control is often organized in terms of a statement of steps that have to be undertaken more or less

sequentially. In only a few cases, the rationale behind the sequence of these steps is made explicit. The papers discussed below all provide explicit means to express task control knowledge.

Archer (1970) describes task control knowledge within a model comprising 30 sub-tasks (mentioned earlier in Section 8.1.1). For example, in the description of the sub-task of evaluating the design description with respect to the requirements, task control knowledge is brought to bear which states what to do in case the design description does not establish a solution of the selected sub-problem. Archer does not really specify task control knowledge separately, with the exception of the reiteration sub-task.

Akin (1978) uses design plans to express task control knowledge. His design plans consist of statements of the form condition-action-intent, which can be read as: if ⟨condition⟩ holds, then take ⟨action⟩ in order to achieve ⟨intent⟩. The action instantiated by the control structure is a direct consequence of the state of the process at the moment of initiation.

Brown and Chandrasekaran's (1989) generic tasks incorporate task control knowledge in their inference strategies. If a problem matches the function of a generic task, then the generic task provides a knowledge representation and an inference strategy that can be used to solve the problem. Thus, generic tasks provide a method for accomplishing each of the sub-tasks into which a task such as design can be decomposed. For example, in the decomposition sub-task, the default inference strategy is to attack design problems top-down: the larger problems are analysed before the smaller ones. (This does not imply that these problems are solved in a top-down order.)

Also Chandrasekaran (1990) pays explicit attention to control issues in design problem decomposition. In his view, there are two types of control issues: one deals with which sets of problem decompositions to choose and the other with the order in which the sub-problems within a given decomposition ought to be attacked. He provides examples similar to Brown and Chandrasekaran (1989) and states that the appropriateness of a given control strategy relies on the dependencies between the sub-problems.

Takeda *et al.* (1990) organize task control knowledge in design scenarios consisting of procedures and rules. These scenarios drive a metamodel mechanism of stepwise refinement of information about the design object. A step in the design process is performed by executing a design scenario, to be selected by the designer. Execution of the scenario transfers the meta-model (comprising all information about the design object regarding functional specification, structure, and actual behaviour) from its current state to the next. If the scenario produces satisfactory results, another scenario is selected to further refine the meta-model. Otherwise, an alternative scenario is selected for the original state of the meta-model.

Runkel *et al.* (1994) use a propose-and-revise method for the VT task, which is expressed as a program with WHILE and IF statements in which design sub-tasks are invoked. These sub-tasks implement operators that describe how to move from state to state in the problem space and that determine if a state is a goal state.

Smithers *et al.* (1994) assume in their theory of design as exploration the availability of a control strategy that uses the history of the design process and the available design knowledge to decide on whether to produce a new well-structured

problem statement or to generate a new (revised) requirements description. Which parts of the design knowledge are relevant for this purpose is not indicated.

In conclusion, the following remarks can be made. First, Archer (1970) and Takeda *et al.* (1990) do not separate task control knowledge clearly from the other types of task knowledge, whereas in the GTMD and in the other approaches discussed above, task control knowledge is specified separately. Second, the GTMD task control structure resembles that of design plans by Akin (1978), inference strategies by Brown and Chandrasekaran (1989) and problem solving methods by Runkel *et al.* (1994). In contrast to DIDS, by means of which the VT-PSM has been constructed by Runkel *et al.* (1994), DESIRE also supports the specification of task control knowledge within composed components that themselves have sub-components. Within our VT task model the control involved in fixing constraint violations has been specified explicitly within a sub-component.

8.2. COMPARISON OF LANGUAGES WITH FORMAL SPECIFICATIONS

There are many commonalities, but also differences, between DESIRE and other formal specification languages. The scope of this section is not to present a detailed comparison, but to highlight some important differences.

In KADS-based specification languages such as $(ML)^2$ and KARL the underlying model is the model of expertise that consists of the domain layer, the inference layer, and the task layer. Other models of KADS (e.g. the communication model) are not included in the model of expertise. DESIRE specifications include specifications of (reasoning about) interaction. Communication between agents (e.g. a system and a user) is explicitly modelled and specified, not only at the level of (object) information exchange but more importantly at the level of strategic (meta) information exchange. The process of interaction may also be subject to strategic reasoning.

A main difference between DESIRE and other approaches is that meta-level reasoning is explicitly modelled and specified. Meta-information can be obtained from tasks at a lower level, reasoned about at a meta-level and meta-information can be reflected downwards to tasks at a lower level. Such information may include, for example, epistemic information about the information state of a task. There is no restriction on the number of meta-levels incorporated in a model. Not only is reasoning about reasoning possibly, but also reasoning about knowledge structures. In the VT-task, for instance, by reasoning about the structure of the domain knowledge parameter dependencies are recognized.

The control of the reasoning process is an aspect in which approaches differ. KARL generates all solutions, but restricts the logical language to make the inference decidable. In $(ML)^2$ the logical language in an inference action is too powerful to be decidable, but limited control is possible during evaluation: any or any new solution may be derived. In DESIRE the extent to which reasoning is afforded to determine the result of a task is explicitly specified: four types of exhaustiveness may be specified indicating any, any-new, all-possible, or every result.

For every DESIRE specification of a system (with finite sorts) it is possible to automatically generate an executable (prototype). This is similar to KARL, but in contrast to $(ML)^2$.

8.3. DISCUSSION OF RESULTS

The main objective of modelling the VT task was to compare different approaches to knowledge modelling. To obtain a clear understanding of the differences and similarities between approaches, underlying assumptions behind knowledge modelling need to be made explicit.

In this paper the DESIRE approach to knowledge modelling has been illustrated for the VT task: the conceptualization, formalization and (prototype) operationalization of the VT design task within a compositional framework have been presented. The philosophy behind DESIRE is that one framework should incorporate these three aspects of knowledge modelling, supporting prototyping of partial task models during the development of a system for a complex reasoning task. Not only are prototype implementations always part of the development process, so are formal specifications. Formal specifications are namely the necessary condition for prototyping. Although other formal specification languages exist (see for a comparison Treur & Wetter, 1993; Fensel & van Harmelen, 1994) few other environments have been developed in which formal specifications play an important role during knowledge modelling and prototyping (MIKE is one of the exceptions, see Landes, Fensel & Angele, 1993).

The conceptualization of a system for complex reasoning tasks is based on a shared task model, acquired in interaction with one or more experts. This model is refined during system design. Due to the reconstruction of the knowledge modelling process which was necessary for the VT task, it is unclear at which level of conceptualization the task model for VT should be seen as a shared task model. The model presented in Section 4 most likely includes too much detail, but this cannot be supported due to lack of actual interaction with experts.

The advantages of a (formal) compositional approach to system design may not, at first, be apparent. Compositional architectures in DESIRE, though, provide support for *reuse,* for the design of *transparent* architectures involving *multiple agents,* for *verification* and *validation* based on formal semantics, and *multiple structures* for knowledge representation. These contributions are discussed below. For reuse the advantages have been demonstrated at two levels: the level of structures and the level of instantiation.

Reuse of existing components such as the generic task model of design, is an example of *reuse of structures.* As discussed in Section 3 the generic task model of design with which the VT task has been modelled, is the result of (1) logical analysis of design and (2) abstraction of existing task models of design tasks in different domains of application. One of the domains of application on which the generic task model of design is based, is the office assignment task, the previous Sisyphus task. The common generic structure of the task model for the VT task and the task model for office assignment is present in the generic task model of design. Specialization (further decomposition) and instantiation (addition of specific knowledge structures) of the generic task model of design for VT follows the task description provided by Yost (1994) closely, both with respect to task and domain knowledge. The generic task model of design provided a basis for knowledge acquisition and formalization, although some re-engineering was needed to extract knowledge regarding the manipulation of requirements from the VT task description. For instance, fixes with

desirability `D9` and `D10` operate on parameters of which the values are dictated by requirements (customer specifications and relevant building information). This also holds for modelling interaction with the user, an essential element in the design of design support systems, for which the DESIRE framework is equipped. In the task model of the VT task presented in this paper a customer can influence requirement and design object description modifications, when necessary.

Reuse of instantiated components is an example of reuse at the level of instantiation. The (composed) component `constraints-analysis` is a component which can be (re)used (unaltered) in task models for applications within other domains. This component has been added to the (as yet unstructured) DESIRE library of pre-specified components.

For *transparency,* the integration of task, control, and knowledge (de)composition at different levels of abstraction within a task model provides a means to combine conceptualization and formalization within one framework. Explicit representation of control for (de)composed tasks provides a means to specify strategic reasoning at an applicable level of decomposition. Reasoning about reasoning processes, meta-level reasoning, is explicitly modelled in compositional architectures with multiple (meta-)levels.

The strong relationship between hierarchical (de)composition of control and task (de)compositions, together with the distinction between meta-level and object-level components provides flexibility with respect to reflective reasoning which is not available within KADS-oriented approaches to task modelling.

For the determination of the formal semantics of a system's behaviour, compositional architectures provide a well-defined structure. The temporal semantics of compositional architectures have been presented in Brazier *et al.* (1995*c*), providing a basis for *validation* and *verification,* as demonstrated in preliminary research in this area by Treur and Willems (1994, 1995).

The integration of *multiple knowledge structures* within one task model is supported by DESIRE. By referencing parts of knowledge structures, parts of ontologies can be imported into task models when required. Different knowledge representation structures can be used for different types of knowledge, when preferred. The generic ontology for design, in ONTOLINGUA, provided a basis for structuring the knowledge required for the VT task. However, the VT-specific ontology given by Gruber and Runkel (1993), in which components do not map onto the ''natural'' components of an elevator but seem to be influenced by the problem-solving method envisioned, was not used entirely as a basis for the DESIRE knowledge structures. The parts of the VT specification which were in line with Yost (1994) were imported.

The DESIRE specification of the VT task model was devised without considering efficiency. Efficiency is not a criterion which is considered during knowledge acquisition and specification with DESIRE; efficiency is a criterion for tool design. One of the consequences of modelling, specifying and implementing the VT task was the recognition of a weakness of the DESIRE environment with respect to efficiency. A first optimization of the current implementation generator, for example, decreased the time required for the prototype generation and execution considerably. The need for improved graphical editors has also been recognized, together with the need for more advanced knowledge acquisition tools that can

import existing ontologies. Semi-automatic retrieval of pre-specified components from the DESIRE library could improve efficiency of the modelling process. The development of more advanced tracing and debugging facilities (combined with the graphical editor) can further improve efficiency of the development process.

This research has been (partially) supported by the Dutch Foundation for Knowledge-based Systems (SKBS), within the A3 project ''An environment for modular knowledge-based systems (based on meta-knowledge) for design tasks'' and NWO-SION within project 612-322-316, ''Evolutionary design in knowledge-based systems.''

References

Akin, Ö. (1978). How do architects design? In J.-C. Latombe, Ed. *Artificial Intelligence and Pattern Recognition in Computer Aided Design,* pp. 65–119. Amsterdam: North-Holland.

Archer, L. B. (1970). An overview of the structure of the design process. In G. T. Moore, Ed. *Emerging Methods in Environmental Design and Planning,* pp. 285–307. Cambridge: MIT Press.

Brazier, F. M. T. & Treur, J. (1994). User-centred knowledge-based system design: a formal modelling approach. In L. Steels, A. Th. Schreiber & W. Van de Velde, Eds. *A Future For Knowledge Acquisition,* Lecture Notes in AI, pp. 283–302. Berlin: Springer-Verlag.

Brazier, F. M. T., Treur, J. & Wijngaards, N. J. E. (1994*a*). *Modelling interaction with experts*: *the role of a shared task model.* Technical report, AI Group, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam.

Brazier, F. M. T., Langen, P. H. G. van & Treur, J. (1995*a*). A logical theory of design. In J. S. Gero & F. Sudweeks, Eds. *Preprints IFIP WG5.2 Workshop on Formal Methods for CAD,* pp. 247–271, Mexico City. Also Technical Report IR-374, AI Group, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam.

Brazier, F. M. T., Klerk, D. A. de, Langen, P. H. G. van & Treur, J. (1993). *A generic architecture for knowledge-based control of processes in dynamic environments.* Technical Report IR-347, AI Group, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam.

Brazier, F. M. T., Langen, P. H. G. van, Ruttkay, Zs. & Treur, J. (1994*b*). On formal specification of design tasks. In J. S. Gero & F. Sudweeks, Eds. *Artificial Intelligence in Design '94, Proceedings of AID '94,* pp. 535–552. Dordrecht: Kluwer Academic Publishers.

Brazier, F. M. T., Langen, P. H. G. van, Treur, J. & Wijngaards, N. J. E. (1995*b*). Redesign and reuse in compositional knowledge-based systems. In P. M. Wognum & I. F. C. Smith, Eds. *Knowledge Based Systems,* Special Issue on Models and Techniques for Reuse of Designs (to appear).

Brazier, F. M. T., Treur, J., Wijngaards, N. J. E. & Willems, M. (1995*c*). Formal specification of hierarchically (de)composed tasks. In B. R. Gaines & M. A. Musen, Eds. *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW '95,* pp. 25/1–25/20. Calgary: SRDG Publications, Department of Computer Science, University of Calgary.

Brown, D. C. & Chandrasekaran, B. (1989). *Design problem solving*: *knowledge structures and control strategies.* Research Notes in Artificial Intelligence. London: Pitman.

Brumsen, H. A., Pannekeet, J. H. M. & Treur, J. (1992). A compositional knowledge-based architecture modelling process aspects of design tasks. *Proceedings of the 12th International Conference on Artificial Intelligence, Expert systems and Natural Language, Avignon-92,* pp. 283–294. Nanterre: EC2.

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert,* **1,** 23–30.

Chandrasekaran, B. (1990). Design problem solving: a task analysis. *AI Magazine,* **11,** 59–71.

COYNE, R. D. (1988). *Logic Models of Design.* London: Pitman.

ENGELFRIET, J. & TREUR, J. (1994). Temporal theories of reasoning. In C. MACNISH, D. PEARCE & L. M. PEREIRA, Eds. *Logics in Artificial Intelligence. Proceedings of the 4th European Workshop on Logics in Artificial Intelligence, JELIA '94,* Lecture Notes in AI, pp. 279–299. Berlin: Springer-Verlag. Extended version (1995) in *Journal of Applied Non-Classical Logics* (to appear).

FENSEL, D. & VAN HARMELEN, F. (1994). A comparison of languages which operationalize and formalize KADS models of expertise. *The Knowledge Engineering Review,* **9,** 105–146.

FORD, K. M., BRADSHAW, J. M., ADAMS-WEBBER, J. R. & AGNEW, N. M. (1993). Knowledge acquisition as a constructive modelling activity. In K. M. FORD & J. M. BRADSHAW, Eds. *International Journal of Intelligence Systems,* Special Issue on Knowledge Acquisition as Modelling, **8,** 9–23.

GEELEN, P. A. & KOWALCZYK, W. (1992). A knowledge-based system for routing of international blank payment orders. *Proceedings of the 12th International Conference on AI, Expert Systems and Natural Language, Avignon-92,* pp. 669–677. Nanterre: EC2.

GEELEN, P. A., RUTTKAY, ZS. & TREUR, J. (1992). On the (non-)brittleness of a DESIRE model. In M. LINSTER, Ed. *Sisyphus '92*: *Models of Problem Solving, Sisyphus yearbook,* GMD Working Papers 630.

GERO, J. S. (1990). Design prototypes: a knowledge representation schema for design. *AI Magazine,* **11,** 26–36.

GOEL, A. & CHANDRASEKARAN, B. (1989). Functional representation of design and redesign problem solving. In N. S. SRIDHARAN, Ed. *Proceedings of the 11th International Joint Conference on Artificial Intelligence, IJCAI '89,* pp. 1388–1394. Los Altos: Morgan Kaufmann.

GRUBER, T. R. & RUNKEL, J. (1993). *Ontolingua files for the elevator-configuration task* [Machine readable data files]. Knowledge Systems Laboratory, Stanford University, Stanford.

HAROUD, D., BOULANGER, S., GELLE, E. & SMITH, I. F. C. (1994). Strategies for conflict management in preliminary engineering design. In I. F. C. SMITH, Ed. *Proceedings of the AID '94 Workshop on Conflict Management in Design,* Lausanne.

LANDES, D., FENSEL, D. & ANGELE, J. (1993). Formalising and operationalizing a design task with KARL. In J. TREUR & TH. WETTER, Eds. *Formal Specification of Complex Reasoning Systems,* pp. 105–141. Chichester: Ellis Horwood.

LANGEVELDE, I. A. VAN, PHILIPSEN, A. W. & TREUR, J. (1992). Formal specification of compositional architectures. In B. NEUMANN, Ed. *Proceedings of the 10th European Conference on Artificial Intelligence, ECAI '92,* pp. 272–276. Chichester: John Wiley and Sons. Extended version (1991): Technical Report IR-282, AI Group, Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam.

MAHER, M. L. (1990). Process models for design synthesis. *AI Magazine,* **11,** 49–58.

MARCUS, S. & MCDERMOTT, J. (1989). SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence,* **39,** 1–37.

MARCUS, S., STOUT, J. & MCDERMOTT, J. (1988). VT: an expert elevator designer that uses knowledge-directed backtracking. *AI Magazine,* **9,** 95–112.

RUNKEL, J. & BIRMINGHAM, W. P. (1994). Solving VT by reuse. In A. TH. SCHREIBER & W. P. BIRMINGHAM, Eds. *Proceedings of the 8th Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW '94,* pp. 42/1–42/28. Calgary: SRDG Publications, Department of Computer Science, University of Calgary.

RUNKEL, J. T., BALKANY, A. & BIRMINGHAM, W. P. (1994). Generating non-brittle configuration-design tools. In J. S. GERO & F. SUDWEEKS, Eds. *Artificial Intelligence in Design '94,* pp. 183–200. Dordrecht: Kluwer Academic Publishers.

SMITHERS, T., CORNE, D. & ROSS, P. (1994). On computing exploration and solving design problems. In J. S. GERO & E. TYUGU, Eds. *Proceedings of the IFIP WG5.2 International Workshop on Formal Methods for CAD.* Amsterdam: North-Holland.

TAKEDA, H., VEERKAMP, P. J., TOMIYAMA, T. & YOSHIKAWA, H. (1990). Modelling design processes. *AI Magazine,* **11,** 37–48.

TOMIYAMA, T. & YOSHIKAWA, H. (1987). Extended general design theory. In H. YOSHIKAWA & E. A. WARMAN, Eds. *Proceedings of the IFIP WG5.2 Working Conference on Design Theory for CAD,* pp. 95–124. Amsterdam: North-Holland.

TREUR, J. & WETTER, TH., Eds. (1993). *Formal Specification of Complex Reasoning Systems.* Chichester: Ellis Horwood.

TREUR, J. & WILLEMS, M. (1994). A logical foundation for verification. In A. G. COHN, Ed. *Proceedings of the 11th European Conference on Artificial Intelligence, ECAI '94,* pp. 745–749. Chichester: John Wiley & Sons.

TREUR, J. & WILLEMS, M. (1995). Formal notions for verification of dynamcis of knowledge-based systems. In M. AYEL & M. C. ROUSSET, Eds. *Proceedings of the European Symposium on the Validation and Verification of Knowledge-Based Systems, EUROVAV '95,* pp. 189–199. Chambéry: ADERIAS-LIA.

YOST, G. R. (1994). *Configuring elevator systems.* Technical Report. Stanford Version 1.3, February 1994, edited and changed by T. E. ROTENFLUH, Medical Computer Science Group, Knowledge Systems Laboratory, Stanford University, Stanford.

VANWELKENHUYSEN, J. & MIZOGUCHI, R. (1995). Workplace-adapted behaviours: lessons learned for knowledge reuse. In N. J. I. MARS, Ed. *Towards Very Large Knowledge Bases. Proceedings of the 2nd International Conference on Building and Sharing Very Large-Scale Knowledge Bases,* pp. 270–280. Amsterdam: IOS Press.

WIELINGA, B. J., AKKERMANS, J. M. & SCHREIBER, A. TH. (1995). A formal analysis of parametric design. In B. R. GAINES & M. A. MUSEN, Eds. *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW '95,* pp. 37/1–37/15. Calgary: University of Calgary, Department of Computer Science, SRDG Publications.