An operational model for design processes

INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN
ICED 01 GLASGOW, AUGUST 21-23, 2001

AN OPERATIONAL MODEL FOR DESIGN PROCESSES

Maxim Ivashkov and Kees van Overveld

Keywords: Knowledge representation, process modelling, hierarchy, ontologies, semantics, architectural design, early design phases.

1 Introduction

A design process can be viewed as a sequence of actions, which comprise the transformation from an initial state, comprising the design goals, to a final state detailing a new product or service and how it should be created [1]. Commonly, a design process is seen as a knowledge-based exploratory and evolutionary process. Many models of a design process recognize phases of analysis and conceptual design [2], [3]. There is a variety of sub-activities in a design process and in this paper we concentrate on the conceptual construction of an architecture of the artefact to be designed (ATBD). The architecture forms the skeleton for all subsequent design stages; design support systems therefore require a representation of this architecture [4]. The fact that design problems are under-defined and open-ended complicates architectural design [5]. Even at the end of the architecture phase many alternatives are left open. It is often necessary to consider several of these alternatives and then to compare their suitability.

The problem we want to study is how the designer can be assisted in maintaining a representation of the ATBD during the early design phases, where he[1] has to generate alternatives and to make conceptual architectural decisions. Nowadays, design support systems (DSS) are used in this phase. DSSs aim to help designers to maintain complex structures of requirements, context, alternatives of an ATBD architecture, etc. According to [6], DSSs have to provide support for exploration, evolution, cooperation, integration, and automation throughout the design process. In order to perform these tasks, many DSSs contain domain-specific models for design knowledge representation and administration during a design process. These models, however, may cause a bias towards certain design styles. This may sometimes interfere with the intentions of the designer, or restrict his creative freedom.

In this paper we will therefore consider a model for design knowledge representation that aims to be as *empty* as possible, whereas it should still be useful in taking the administrative burden off of the designer's shoulders. Also, our model tries to help designers integrate context, requirements, and alternatives for the ATBD architecture at hand. Furthermore, as our model invites the designer to think in terms of well-defined relations between the various concepts he uses, we think that our model also stimulates cleaner thinking about the design problem. Using the model it is easy to express ideas, but also questions, doubts and alternatives. The model allows documentation in a uniform way by means of well-defined semantics.

[1] Everywhere in this paper, 'he/his' is short for 'she or he/her or his'

2 Related Work

Several authors address the issue of (formally) representing design knowledge in multi-disciplinary contexts. Our work is similar to, and partially inspired by, Fujii et.al., [15] in the sense that a state transition model of a design process is used, and properties of the ATDB are expressed in predicates; transitions are defined by their effect on these properties. The underlying modal logic in [15] is based on the work of Brazier [16] et. al. Our model features less advanced automated reasoning, since we allow attributes (functions) of concepts that are not restricted to truth-values; hence we cannot apply formal logic manipulations to the same extent as [15] and [16]. (See 3.1). Some knowledge, however, that we *do* represent and manipulate explicitly comprises various forms of *hierarchy*. Our model caters for the various types of *hierarchical relations* that designers use in all stages of design. (See 3.3) Several authors explain the role of hierarchies in designing: according to Simon [10], hierarchies increase evolution speed and allow the decomposition of the ATBD. Douglas describes how a design problem can be reduced to a hierarchy of decisions or alternatives [5]. Therefore the machinery behind our model uses hierarchical relations from Object Orientation (OO) [7], extended with some operators to express intentional relations, together with notions from spreadsheets [8] and symbolic manipulation [9]. (see3.1) Numerous models of design knowledge representation are based on hierarchical approaches. Such models can be divided into *descriptive* and *prescriptive* models.

2.1 Descriptive models

Descriptive models of design process are used for *describing* different kinds of products, design problems, specifications, and solutions. An example is the STEP library. STEP is a set of ISO [11], [12], [13] standards, which provides the exchange of engineering product data between databases and CAD systems. STEP is built on an information modelling language that can formally describe the structure and correctness conditions of any engineering information. The library consists of several parts, each consisting of a hierarchy of instances of entities in a data model. Examples of parts are classes of physical objects, classes of aspects of physical objects, etc. Classes of objects have a textual definition but also an intentional definition via a class of properties. Each class has to be associated to another class by types of associations (is-a, used-in, part-of, connected-with etc.). STEP does not support the design *process* of the ATBD. Alternatives to an ATBD architecture can only be partially expressed using optional associations with other classes. There is no computational engine to propagate decisions throughout an ATBD description.

2.2 Prescriptive models

Prescriptive models concentrate on the development and application of strategies, methods, techniques, and tools that are used for designing. An example is the KBDS system.

KBDS [14] is a Knowledge Based Design System, which allows a team of designers to maintain a representation of the design process not only as a historical record of the development of a design artefact, but as an "active" repository of information. Alternatives are represented in the form of the space of design alternatives (SDA). There is also the space of design objectives (SDO) related to SDA. The SDO is used for both generation and evaluation of alternatives in SDA. The model allows the designer to keep track of several design alternatives at a time; evaluating an alternative design against a set of design objectives; and transparently accessing external applications. The ideas that we propose in this paper are in no

way as mature as either of the above models: for instance, a computer implementation is still under development. Still, we think that it is worthwhile to explore a model that has both descriptive *and* prescriptive sides.

3 An operational model for design processes

We propose a model for design processes that should allow denoting design processes for the sake of exercising, researching and teaching cross-disciplinary design skills. Here, 'cross-disciplinary' refers to design situations where several designers with substantially different backgrounds have to co-operate. We assume that therefore no single discipline-related design methodology is available, and both technological and non-technological issues should be taken into account. Our model typically relates to early design stages, when major architectural decisions are taken without the ability to rely on the detailed knowledge that is only available in later stages. We want the same model to govern requirements, context and architectural issues. The model should cope with administrating (the consequences of) alternatives for design decisions, and it must be possible to postpone decisions or revoke earlier decisions. If our model proves to be adequate (which is still to be assessed), it should be possible to give automated support, e.g. to ensure, as far as possible, consistency among various decisions. It should, to a large extent, take the administrational burden from the designers without enforcing a strict format upon the creative design process.

3.1 Basics of the model

To this aim, our model contains the following ingredients:

- concepts (classes or instances)
- attributes, which are predicates over these concepts
- few pre-defined attributes, such as 'has-a', 'is-a', needs-a', that occur in virtually all kinds of design situations
- the notion of a spreadsheet, as this is a familiar device for incremental 'what-if'-analysis in many disciplines
- the notion of partial and symbolic evaluation (see below), as this allows decisions to be postponed or revoked. Notice: traditional spreadsheets don't allow cells to be non-evaluated: they cannot propagate expressions, but only the (numerical, logical or alpha-numerical) *values* of expressions. This is clearly insufficient for postponing decisions.

With this model, a design process is a sequence of *tables*. A table contains 0 or more rows; every row represents one concept. A table contains 10 or more columns; every column represents an *attribute*. We have 10 pre-defined attributes with pre-defined meanings. Also, the propagation of these meanings to other cells is well defined. This means that, if the designer defines or changes the value of one attribute, the values for related pre-defined attributes can be updated automatically. The pre-defined attributes come in two sets of 5 (=4+1). The two sets are each other's inverses, in the sense that 'has-a' is the inverse of 'part-of'. One pair of inverses is the 'classifies-

	intended	observed
ontologic	has-a / part-of	is-a / specializes-as
causal	needs-a / allows-a	causes-a / has-cause

as' and 'instantiates-as' pair that connects classes and instances. The other 4 pairs of pre-defined attributes are defined for classes only. They are depicted in the above diagram. Some examples illustrate these pre-defined attributes: is-a(table) → furniture; specializes-as(furniture) ⇒ table; has-a(table) → legs; part-of(legs) ⇒ table; instantiates-as(my-favourite-chair) → chair; classifies-as(my-favourite-chair) ⇒ chair; needs-a(lamp) → electricity; causes-a(lamp) → light; et cetera. The expression after the arrows indicates the contents of a *cell* in the table. The user enters expressions after single arrows (→); the expressions after double arrows (⇒) follow automatically from the meaning of one of the pre-defined attributes. A cell is characterized by a *row* (concept, say C) and a *column* (attribute, say A). The cell represents the expression A(C). If A(C) can be evaluated; it also represents *the value* of this expression, similar as is done in a spreadsheet. In all previous examples, evaluation was fully possible. In some cases, for instances with conditional expressions (see 3.2), evaluation is only partially possible. Indeed, the contents of cells can be more involved as we will explain below. With our tables, consisting of rows, columns and cells, we represent the design process as a state transition machine. The initial state is a table with 10 columns and 0 rows. A state transition can be one of: adding a concept (=adding a row); adding an attribute (=adding a column), or *modifying* the contents of a cell. It is assumed that the modification of a cell immediately invokes updating other cells if the pre-defined semantics of the attributes allows this. For instance, if the cell has-a(table) is filled with the word 'legs', and 'legs' is the name of an already existing concept, then the cell part-of(legs) is filled with the word 'table'. More complicated examples of updates will be explained later.

3.2 Expressions in cells

Attributes will often have multiple values. In order to express this, the expressions in cells may contain operators. Operators include: BOTH, ALT (abbreviation of 'alternative'), OPT (abbreviation of 'optional'), IF, ITE (abbreviation of if-then-else), IN, AND, OR, and other obvious relational operators for Boolean and numerical types. We illustrate some of these operators with examples:

- has-a(chair)→BOTH(seat, backrest, legs) gives rise to part-of(seat) ⇒ chair; part-of(backrest) ⇒ chair; part-of(legs) ⇒ chair.

- needs-a(electric-lamp)→ALT(battery,220V). Notice that in this case no automatic deduction of allows-a(battery) or allows-a(220V) can take place. Also notice that the entries 'battery', '220V', and all other constants are left unevaluated unless the designer considers it worthwhile to identify *concepts* with these terms for further elaboration. Our system does not contain any ontological information about the domain; it only administrates the ontology as it builds up in the designer's mind. At a future stage, however, the designer may want to elaborate on, say, the battery: then he may introduce a concept 'battery' with attributes in its own right.

- has-a(chair)→BOTH(seat, legs, backrest, OPT(armrests)) expresses that a chair has a seat, legs and a backrest; armrests are optional.

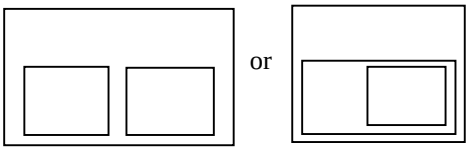
- is-a(chair)→ITE(made-by-artist(chair),piece-of-art, furniture), where the first argument of ITE is a condition. ITE is an abbreviation for If-Then-Else. If this condition evaluates to 'TRUE', the second argument is returned; otherwise the third argument is returned. So in this case the value of the cell is-a(chair) is only defined if the attribute 'made-by-artist' returns a definite value for the concept 'chair'. But even if 'made-by-artist(chair)' does not evaluate to TRUE or FALSE (for instance, the designer hasn't made up his mind yet,

so the cell made-by-artist(chair) is still empty), we can use the contents of the cell is-a(chair) in other expressions (which may or may not be evaluated in turn). The assignment of a definite value to any attribute may take place at any time; its effect will immediately propagate to all cells that depend on it. Similar, a definite value may be withdrawn at any time, and *values* that depend on it will loose their validity; the associated *expressions*, however, will continue to hold.

- If times-used(room)→BOTH(night, day), then has-a(room)→IF(IN(night, times-used(room)),lamp) evaluates to 'lamp'. So the operator IN(x, y) is used to check if x occurs in (the expression) y. If y in turn is a conditional expression, depending on condition K, the condition K is also accounted as a condition in the result of 'IN'.

- part of the pre-defined semantics of 'is-a' is the following: suppose, for concept x, attribute A(x)→E(x) where E is some expression in x. If is-a(y)→x, then we can automatically deduce A(y)⇒E(y). This is consistent with the concept of inheritance from Object Orientation.

3.3 Transitivity, hierarchies and architectures



The 10 pre-defined attributes, has-a … is-caused-by, are transitive relations between classes. Transitive relations, such as (multiple) inheritance, are at the heart of hierarchies as they occur commonly in all sorts of design processes. For modelling design the exploitation of transitivity in concert with our operators has some attractive consequences. For instance, suppose we want to postpone the decision between the two architectures depicted here. Then we can simply write part-of (S1)→S; part-of(S2)→ ALT(S,S1). This allows further argumentation where both options are left open for a while. In a similar way, transitivity can be used to deduce about fulfilment of requirements or side effects. For instance, from causes-a(p)→r and needs-a(p)→s and causes-a(q)→v and needs-a(q)→r we may automatically deduce that needs-a(v) ⇒ s and causes-a(s) ⇒ v. Obviously, these argumentation patterns occur frequently throughout most design processes. Indeed, design is about the fulfilment of requirements, and it is crucial to assess if at a certain stage all requirements are fulfilled. Our model provides a natural mechanism to integrate these argumentation patterns (that involve stakeholders and their attributes and other terms from the context) with argumentation about technological or architectural aspects of the design.

In the next section, we present a 'hand-made' example of a small design process that makes use of our model. For more realistic examples, a software implementation is indispensable. At the end of section 4 we comment on the current status of our prototype.

4 Example

As an example of a system to design we chose a prototype of a *dynamic board* (d-board.) The main idea of a d-board is to enhance the functionality of a normal lecture room blackboard (board.) A conventional blackboard serves in teaching processes, where a teacher writes or draws on the board synchronously with his oral presentation. Therefore, a snapshot of a board with the writing on it only captures a small portion of the meaning. The dynamical process, which carries a lot of the didactics, is lost. We aim to enhance the functionality of a board in

such a way that speech and drawn information are recorded in the chronological order for later usage, thus enhancing the meaning.

4.1 Dynamic board prototype

The starting point in the design process was made when we decided to enhance an existing board instead of building a completely new system. The first concepts introduced in the table are related to our `Lecture room` and a class of `Boards` that will fit the `Lecture room`. (See steps 1-6.) We can also introduce new attributes such as `Area` (steps 3-4). Our doubts about a board size can be expressed as alternatives. For instance, we restrict the board area to medium (`m`) and large (`l`), because these are commonly used in lecturing (step 7). The auxiliary devices, namely the `Marker` and the `Eraser`, are used as tools. Using the `IS-A` attribute they are generalized into the more generic concept `Tool` (steps 8-9). Any later modification of the `Tool` will cause automatic modification of both the `Marker` and the `Eraser`. For instance, adding an attached `Wire` to the `Tool` (step 10) causes automatic propagation of the `Wire` to the `Marker` and the `Eraser`. Notice that all the concepts we have introduced so far are either specific instances (`My Lecture room`) or generic classes (`Lecture room, Board`.) They are used to describe a context or system parts. In step 11 we introduce the concept of `Understandability`, which is of a different kind. It does not have a specific instance and furthermore it describes a requirement. Despite these big differences we treat all concepts in a similar way. The `Understandability` will increase if both `video` and `audio` components are present (step 12). After thinking about different alternatives we came up with the `Microphone (Mic)` to provide `audio` and the two alternatives, namely `Camera` and `Digitizer,` to provide `video` (steps 13-18). In steps 19-20 we decide to use a `Camera` for medium-sized boards and to use a `Digitizer` for large boards. For our purposes a `Digitizer` needs a special `Interface (Int-ce)` and a built-in `Microphone` (step 21). The `Interface` of the `Digitizer` can be either wired or wireless (step 22). However, we know that a wireless interface is more expensive then a wired one. To express this knowledge, we add a new attribute `Price` and assign a value to it that corresponds to the `Digitizer` (steps 23-24).

In a similar way we, can continue until the design is finished or no open alternatives are left. For instance, if we decide to use a large board then the table automatically maintains consistency and computes that a `Board` has a `Digitizer` with a `Wireless Interface;` the total `Price` will then exceed $400.

4.2 Software: prototype and future extensions

The table at the end of this section was hand-generated. However, we are developing a system to automate the administration of such tables. The core of this system is a parser, an expression evaluator, and an engine to propagate expressions between cells based on the pre-defined attributes. The first two components are now operational. In order to propagate expressions, thereby taking the semantics of pre-defined attributes and conditions into account, we propose to use, in every cell, an internal normal form. Such a normal form is a list of all possible return values of a cell (constants or concepts), where for every entry in the list we administrate the condition for which this value results. This normal form facilitates merging expressions, which is necessary, among other things for multiple inheritance. Finally, the core will be interfaced to a standard spreadsheet front-end, so that familiar interaction techniques automatically apply.

0) CONCEPTS	0) IS-A	0) SPECIALIZES-A	0) HAS-A	0) PART-OF	0) CAUSES-A	0) NEEDS-A	0) CLASSIFIES-A	3) AREA	23) PRICE
1) My Lecture room							2) Lecture room	4) 30	
2') Lecture room			5) Board						
5') Board			6) BOTH(Marker, Eraser) 19'-20') OPT(Camera,Digitizer)	5')Lecture room				7) ALT(m, l)	
6') Marker	8) Tool		10')OPT(wire)	6')Board					
6') Eraser	9) Tool		10')OPT(wire)	6')Board					
8') Tool		8') BTH(Marker, Eraser)	10) OPT(wire)						
11) Understan-dability						12) BOTH(video, audio)			
13) Camera			15)Mic	19)IF(IN(Size(Board),m), Board))	14) video				
15') Mic				15')Camera 21')BOTH(Camera, Digitizer)	16) audio				
17) Digitizer			21)BOTH(Int-ce,Mic)	20)IF(IN(Size(Board),l),Board))	18) video				24)ITE(IN(IS-A(Int-ce), wireless), >$400, <$400)
21') Int-ce	22) ALT(wire-less, wired)			21')Digitizer					

In this table we represent the dynamics of the design process. To that aim, all entries are prefixed with a rank number; these numbers count the subsequent actions (decision) of the design. If one cell contains two entries, with different numbers, this indicates that in the course of the process, the content of this cell was changed from the first to the second entry. Numbers with prime (e.g. 6') result from automatic propagation; in this case step 6 gives rise to the automatic update in the entry labelled 6'. Label 0 is given prior to the start of the design process. Abbreviations: 7) m-medium, l-large 14) Mic -Microphone 23) Int-ce –Interface

Conclusions and Future Work

We study the problem of assisting designers in early design phases of interdisciplinary design situations where no *dedicated* design theory or methodology is available. Our approach is characterised in that is it generic but at the same time aims to give support in administrating,

postponing and revoking design decisions. We are in the process of implementing our ideas and using them in test cases with practitioners in order to show their practical usefulness.

References

[1] Banares-Alcantara, R. (1991). "Representing the engineering design process: two hypotheses." Computer-Aided Design 23(9): 595-603.

[2] Hubka, V. "Design for quality and design methodology." J. Engang Des., 1992, 3(1), 5-15.

[3] French, M, J. "Conceptual design for engineers", 1st edition, The Design Council, London, 1971.

[4] Smithers T. "AI-based design versus geometry-based design, or why design cannot be supported by geometry alone." Computer-Aided Design 21, 1989, 141-150.

[5] Douglas, J.M. "Conceptual design of Chemical Processes", McGraw-Hill chemical engineering series, 1988, 5-20

[6] Bañares-Alcántara R. "Design support systems for process engineering. I. Requirements and proposed solutions for a design process representation." Computers & Chemical Engineering 19, 1995, 267-277.

[7] Booch, Grady. "Object-Oriented Design with applications." Benjamin/Cummings, 1991

[8] Microsoft Corporation, "a handbook for Excel", Edition 5, Ireland, 1994

[9] Stephen Wolfram, "the Mathematica Book", third edition, Cambridge University Press, 1996

[10] Simon, H.A. "The Science of the Artificial", The MIT Press Cambridge, Massachusetts London, England, second edition, 1981.

[11] ISO 10303-41, PART 41: "Integrated Resources: Fundamentals of Product Description and Support", ISO, 1994.

[12] ISO 10303-44, PART 44: "Integrated Resources: Product Structure Configuration", ISO, 1994.

[13] ISO 10303-203, PART 203: "Application Protocol: Configuration Controlled Design", ISO, 1994.

[14] Bañares-Alcántara R. and H. M. S. Lababidi. "Design support systems for process engineering. II. KBDS: an experimental prototype." *Computers & Chemical Engineering* **19**, 1995, 279-301.

[15] Fujii, H. and Nakai, S.: "On Formal Representations of Multi-Disciplinary Design Process." Third International IFIP WG 5.2 Workshop on Formal Design Methods for CAD, 1997

[16] Brazier, F., van Langen, P, and Treur, J.: "A Logical Theory of Design." In J.S. Gero (ed.), Advances in Formal Design Methods for CAD, Chapman & Hall, 1995, 243-266.

M.Y. Ivashkov, M.Sc., C.W.A.M. van Overveld, Ph.D.
Stan Ackermans Institute, 5600 MB, Eindhoven, The Netherlands, Tel: +31 40 2474772, e-mail: m.ivashkov@tue.nl / k.van.Overveld@wxs.nl, http://www.sai.tue.nl/research/