Ulam Quarterly — Volume 1, Number 4, 1992

Knowledge Based Specification of the Design Process Using Many-Sorted Logic

Mihaly Lenart *
Department of Architecture
University of Kassel
Haenschelstr 2, Germany

Ana Pasztor
School of Computer Science
Florida International University
Miami, FL 33199

Abstract

Due to the unprecedented pace of technology development and the increasing flexibility of manufacturing, there is a growing demand for design automation. Since design often requires creativity or even ingenuity, it is one of the most complex tasks humans perform. Because of its complexity, design automation has been limited to routine and detail design. Recently, however, attention has been shifted towards conceptual and innovative design [Navichandra, 1990]. A number of techniques and methods has been proposed, such as [Mostow, 1985], [Goel, 1989] or [Navichandra, 1991a]. Most of these approaches, however, emphasize implementational issues. Design aids or specification methods still lack conceptual frameworks. The research described in this paper is concerned with the formal specification of the entire design process. The proposed many-sorted first order logic approach provides several advantages, such as modular decomposition, encapsulation, direct implementability and reusability. Its main advantage, however, is the faithful formal representation of the design process.

First order many-sorted logic is an extension of classical first order logic. In classical first order logic, the world is represented by a class of models which contain homogeneous sets. A mathematical model in first order logic is, therefore, a homogeneous set with a set of (total) functions and relations. Our design environments, however, are made of different types of objects and for each type we have different sets of relations and operations. Therefore, an adequate representation of the world should contain heterogeneous sets. A heterogeneous or many-sorted model is based on a set of elements divided into subsets or sorts. Each element belongs

---

On leave, current address: School of Design, Florida International University, Miami, FL 33199, USA

to exactly one sort. Functions are relations are defined only on elements belonging to a specific sort.

1. Introduction

McCarthy and Hayes write in [McCarthy, 1969]: "... intelligence has two parts, which we shall call the epistemological and the heuristic. The epistemological part is the representation of the world in such a form that a solution of problems follows from the facts expressed in the representation. The heuristic part is the mechanism that on the basis of the information solves the problem and decides what to do." Most of the work that has been done in design related AI research is of heuristical nature. Our aim is to provide an epistemological approach in order to gain a better understanding of the entire design process. Better understanding, on the other hand, will provide a better representation of design problems. A better representation leads to efficient solution strategies (heuristics) for solving particular design problems.

By the design process we understand the development of a plan how to make a transition from a given state to a goal state. The goal state is usually a model of a given design object. The design object can be almost anything: a building, a ship, computer program, book pages, machine parts, etc. However, we would like to distinguish between physical design objects, such as furniture or computer chips, and abstract ones, such as computer programs or chemical (drug) compositions. In the following we restrict ourselves to physical (2 or 3 dimensional) design objects.

It seems that there is a general consensus about the main components of the design process and in which order these components are executed. The design process comprises the following three components, which we will explained in detail later:

1. Description of the design parameters, constraints and goals,
2. generation of a model for the design objects, and
3. evaluation of the model.

After the evaluation of the model, constraints might be relaxed, goals redefined and the above sequence repeated, until a satisfactory solution emerges. A solution, i.e. a model of the design object, is considered to be satisfactory if we achieved the goals without violating the constraints. However, there is no general rule or method for measuring the degree of goal achievement or constraint satisfaction. (Since, instead, unspecific measurements are commonly used, we also investigate modal extensions of our specification language.) If this degree is quantifiable, then we talk about optimization problems. Most of the design problems are not optimization problems. Moreover, even if we are confronted with optimization problems, it doesn't mean that we can describe or solve them by well known mathematical methods. The reason for this is twofold:

a. Even the simplest optimization problems often lead to combinatorial explosions, and

b. goals are often interdependent, i.e. the satisfaction of one goal has an impact on the satisfiability of another goal.

If we can measure the degree of goal satisfaction, but at the same time have interacting goals, then we talk about relative or Pareto optimal solutions. Pareto optimal solutiions are those whose improvement regarding any of the goals set would reduce the degree of satisfaction of another goal [Pareto, 1896].

Since optimization is a specific and often intractable technique for tackling design problems, it cannot be applied widely. Current representtion or problem solving methods, such as production systems, object oriented or logic programming paradigms, on the other hand, are geared toward the solution of specific design problems. Formal specification becomes a necessity once we move toward more complex and conceptual tasks. This has been widely recognized in other areas, such as planning (see e.g. [Allen, 1984]) or database management (see e.g. [Gallaire et al., 1984]).

Classical first order logic has been established as a commonly used declarative method for specification. In the following, we propose the appliction of an extension of first order logic for specification. This extension is first order many-sorted logic, which – as will be shown in this paper – is more appropriate for design specification. In classical first order logic, the world is represented by a class of models which contain homogeneous sets. A mathematical model in first order logic is, therefore, a homogeneous set with a set of (total) functions and relations. Our design environments, however, are made of different types of objects and for each type we have different sets of relations and operations. Therefore, an adequate representation of the world should contain heterogeneous sets. A heterogeneous or many-sorted model is based on a set of elements divided into subsets or sorts. Each element belongs to exactly one sort. Functions and relations are defined only on elements belonging to a specific sort. (This concept is similar to type declarations in programming languages.) Our specification method is based on the language of many-sorted first order logic. It allows a faithful representation and fine tuned modeling of design tasks.

The present paper is structured as follows. First, we discuss relevant concepts. Then we introduce the specification language. The concepts and the use of the specification language are illuminated by an example. Due to space limitations, the example has been kept simple and the specification partial. The proof theoretical part is the subject of ongoing research and will be published separately. A temporal version of many-sorted logic has been successfully applied to knowledge based user interfact specification, see [Pasztor, 1991], [Pasztor et al., 1991a] and [Pasztor et al., 1991b]. An application of many-sorted first order logic for modeling building design was shown in [Markusz, 1981], [Markusz, 1983] and [Markusz, 1989]. This

application has also shown how the formal specification in many-sorted first order logic can be implemented in Prolog.

2. Making Design Tasks Discrete

We distinguish between continuous and discrete design. *Continuous* design means infinitely many alternatives at certain stages of the design process. An example could be the design of a wooden chair, where we can choose almost any shape we want for the legs, the back or the arms of the chair. Even though the design is limited by a number of constraints (manufacturing, structural, functional, etc.), the number of solutions is also unlimited. This, however, is not the case with the design of computer boards containing a number of chips, transistors, resistors, switches, etc. This, like many other design problems, such as allocating machines or designing buildings made of prefabricated elements, has a finite number of solutions at any given stage of the design process. The latter are often referred to as *configuration problems* and have a finite solution space. There are, however, design prolems that are not configuration problems, but still have finite solution space. An example is typographical design. In this case we have constraints, such as letter styles or sizes defined by professional standards, limiting the number of choices. As opposed to continuous design, we call all design tasks with finite solutioin space *discrete*.

Since humans, as well as computers, can handle only a finite number of alternatives, continuous design tasks will often be transformed into discrete ones. This transformation, as well as further concepts, can be best explained by way of an example. For this purpose let us look at the following simple task of designing the arrangement of adjustable shelf units for accommodating books and a stereo system. The units are made of uprights connected by horizontal shelves. The uprights are identical and their height is 6'. The connection between the shelves and the uprights is solved by holes or notches in the uprights at 2" distances. Appropriate connecting elements are attached to the holes or notches in the uprights and to the shelves, but the details of the attachement or joints are irrelevant for our example. Instead of connections at regular distances, we might have a continuous rail with sliding connections allowing arbitrary vertical adjustments of the shelves. This choice turns the task into a continuous one. However, for all practical purposes, it is still a discrete one, since it is sufficient to consider only a finite number of adjustments based on the size of the objects we want to put onto the shelves. In our example, let us assume that we have paperback books whose average height needs a vertical shelf distance of 8", hard covers requesting 10", stereo basic components 4" and speakers 16". Since the greatest common divisor is 2", we really need to consider only shelf distances which are multiples of 2" (if we start stacking the shelves from the bottom). In other words, the designer can ignore the rails and imagine holes in the uprights at 2" distances without loosing any

interesting solution. In this example, transforming a continuous task into a discrete one is easily justifiable. If, however, there is no apparent reason for this transformation, designers still apply self-imposed constraints, such as grids or standard sizes, making continuous tasks discrete.

3. Semantic Layers

Design is carried out in a top-down fashion at different abstraction levels. Similar to hierarchical planning [Sacerdoti, 1973], design tasks can recursively be broken down into subtasks of manageable size. Accordingly, the specification should allow abstraction at different levels. In our example we have three components:

1. a shelf system,
2. objects to be arranged on the shelves, and
3. the room or environment we will furnish with the shelves.

The shelf system is made of units. A unit is a stack of shelves with one upright on the left and with either another unit or an upright on the right. The units need to be stabilized by crossbars. A crossbar attached to the back of two neighboring uprights stabilizes at most three connected units. The objects are books and stereo components. The books are either paperback or hard cover. The stereo components are basic components, such as amplifier, tuner and a CD-player and two speakers. For keeping the example simple, we consider only geometrical (size) constraints; moreover, only two dimensional ones. This means that we don't consider e.g. weight or esthetical constraints and we will ignore the depth of both shelves and objects. The place we want to furnish is an empty wall.

Preliminary design is concerned with the calcualtion of storage space and determining how many units we need. After obtaining this number, we would like to arrange the units along the wall. Once we have arranged the units, we shift around the objects in order to find an acceptable or satisfactory arrangement of objects on the shelves.

Detail design is concerned with the placement of the shelves and the arrangement of particular objects on any given shelf. The arrangement of the units, the shelves and the objects on the shelves also provide assembling information for subsequent planning tasks.

4. Design Process

The first step in the design process is to describe parametrs, constraints and goals. In our exmaple *design parameters* are the size and number of the parts of the shelf system, the size and number of the objects to be placed on the shelves and the size of the wall.

Constraints can be classified as functional, aesthetic, structural, etc. Functional constraints restrict e.g. the location of the speakers or the placement of the units in relation to doors and windows. Aesthetic constraints might say something about dense or loose arrangements of objects, color distribution or symmetrical vs. asymmetrical configurations. A structural constaint in our example is the placement of the stabilizing crossbar. Another structural constraint would be prohibiting to put all heavy objects on top shelves. (Most of these constraints cannot be applied to our example since we excluded all but size parameters.)

A *goal* of the design process is a description or specification of the design object. In our case a goal would be a shelf configuration accommodating all specified objects. Goals may include constraints of high priority, i.e. constraints which have to be satisfied as opposed to other constraints that under circumstances may be violated. Such a high priority constraint would be to arrange all objects using the least number of shelf parts or having all objects on the shelves without any empty space left.

5. The Specification Logic

A specification logic SL consists of three components, namely

- the language of SL,
- a calculus for SL, and
- a set Ax of formulas of the language of SL, called the axioms of the specification. The axioms consist of goals, parameters and constraints.

The axioms form the actual specification, while the first two items (i.e. the language with the calculus) represent the *logic* of the specification. The axioms of Ax are formulated in the language of SL, while a calculus for SL is an algorithm to derive all consequences of the specification Ax.

The *language* of SL is a triple $\langle Fm, Md, \models \rangle$. Fm is the set of formulas of SL and represents the *syntax* of the language. The pair $\langle Md, \models \rangle$ is the *semantics* of SL, Md, being the class of all possible models for the language, and $\models$ being a relation between the formulas and the possible models called the *satisfaction* relation.

To illustrate the above concepts by way of an example, suppose we want to talk formally about the natural numbers. We will use a language with the symbols + (as a binary operation symbol), 0 (as a constant symbol) a.s.o. We will be able to write formulas like e.g. $x+y=0$, $x+y=y+x$ etc. A possible model for this language is the set of all natural numbers with the usual meaning of + and 0 being addition and zero, respectively. However, the set of all natural numbers with the operation of taking the maximum of two numbers as the meaning of + and the number 10 as the meaning of 0, is also a possible model. The same way the set of all sequences of some fixed

set of symbols with the concatenation of two sequences as the meaning of + and the empty sequence as the meaning of 0, is also a possible model. (Among all these possible models, we will be interested only in those which actually satisfy the axioms of our specifiction Ax. For example, in this case, if $x+y=y+x$ were our axiom, we would really be interested only in those possible models in which the meaning of + is a commutative operation.)

First we will concentrate only on the *language* part of our logic.

The axioms or the actual specification is a set of formulas in Fm representing a formal description of the subject of specification, e.g. a shelf system in our concrete case. The specification is said to be *correct*, if every possible model $\mathcal{M} \in Md$ which satisfies the specification (and some additional constraints, like e.g. initiality in [ADJ, 1976]) is an *intended* model (i.e. is acceptable to the end user). The specifiction is *complete* if every property of the class of all intended models can be derived from the axioms with the calculus.

After experimenting with various logics for our specification, we came to the conclusion that first order many-sorted logic is the most suitable for our specification. The reason for this is that our main concern is the faithful representation of the complexity of design tasks within the framework of classical first order logic. In the following we give a brief definition of first order many-sorted logic.

First we define the *syntax* of our language, i.e. we specify the symbols to be used and then define those strings of symbols (called *formulas*), which we accept as "legal".

Notation. We will use ω to denote the set of all natural numbers.

Definition 5.1 (the nonlogical symbols of the langulage of L_S)**.** The following are the "ingredients" of the language of L_S:

1. A nonempty set S whose elements are called sorts.
2. For every $s \in S$, a set $X_s = \{x_i^s : i \in \omega\}$ of variables of sort s.
3. (S^* denotes the set of all finite length strings of elements in S. S^+ denotes $S^* - \{\lambda\}$, where λ denotes the string of length 0.) For every string $\sigma \in S^+$, the set Rel_σ of relation symbols of arity σ. (In our shelf system design example, we will have relation symbols which are going to be interpreted as properties of objects. So their arities are strings of length one, e.g. $PB \subset Rel_{ob}$, where ob is the sort of the objects to be placed on the shelves and PB stands for the property of being a paperback book.)
4. For every $\sigma \in S^*$ and $s \in S$, the set $Op_{\sigma;s}$ of operation symbols of arity σ; s, (i.e. of rank σ and sort s). (In our shelf system design example, we have operation symbols whose meaning is the projection of units of shelves – viewed as sequences of shelves – on their component shelves. So $p_1, \ldots, p_{36} \in Op_{u;sh}$.)

Definition 5.2 (the set Tm_s of all terms of sort s)**.** Let $s \in S$. Then

1. For every $x \in X_s$, $x \in Tm_s$;
2. Let $f \in Op_{\sigma;s}$. Assume $\sigma = s_1, \ldots s_n$ for some $n \in \omega$ and $s_i \in S$ $(1 \leq i \leq n)$. Let $\tau_1 \ldots \tau_n$ be terms of sort $s_1, \ldots, s_n$, respectively, i.e. $\tau_1 \in Tm_{s_1}, \ldots, \tau_n \in Tm_{s_n}$. Then $f(\tau_1, \ldots, \tau_n) \in Tm_s$.

Definition 5.3 (The set Fm_s of formulas of L_S)**.**

1. Let $s \in S$ and $\tau_1, \tau_2 \in Tm_s$. Then $\tau_1 = \tau_2$ is an atomatic formula of L_S. Let $P \in Rel_\sigma$ and $\sigma = s_1 \ldots s_n \in S^+$. Suppose $\tau_1 \in Tm_{s_1}, \ldots, \tau_n \in Tm_{s_n}$. Then $P(\tau_1, \ldots, \tau_n)$ is also an atomic formula of L_S. Each atomic formula is a formula.
2. Let $\varphi, \psi \in Fm_S$, $x \in X_s$, $s \in S$. Then
   $\neg\varphi \in Fm_S$,
   $(\varphi \wedge \psi) \in Fm_S$ and
   $\exists\, \varphi \in Fm_S$.

We use the abbreviation $\forall x \varphi$ for $\neg\exists x \neg\varphi$, $(\varphi \rightarrow \psi)$ for $(\neg\varphi \vee \psi)$, and $(\varphi \leftrightarrow \psi)$ for $((\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi))$. with this we have defined the syntax of L_S. We are now going to define the *semantics* of L_S. In other words, we are going to define the *meaning* of formulas in possible models.

Definition 5.4 (Possible models for L_S)**.** A possible model $\mathcal{M}$ for L_S consists of the following.

1. For each $s \in S$, a nonempty set D_s, called domain or universe of sort s,
2. For each relatioin symbol $P \in Rel_\sigma$ with $\sigma = s_1 \ldots s_n \in S^+$, a relation $P^{\mathcal{M}} \subseteq D_{s_1} \times \ldots \times D_{s_n}$, and
3. For each function symbol $f \in Op_{\sigma;s}$ with $\sigma = s_1 \ldots s_n$, an operation $f^{\mathcal{M}} \colon D_{s_1} \times \ldots \times D_{s_n} \rightarrow D_s$.

We denote by Md_S the class of all possible models for L_S.

Definition 5.5 (The satisfaction relation $\models \subseteq Fm_S \times Md_S$)**.** Given a possible model $\mathcal{M} \in Md_S$, an evaluation (of the variables to $\mathcal{M}$) is a family $e = (e_s \colon s \in S)$ of maps, where $e_s \colon X_s \rightarrow D_s$ for each $s \in S$.

Let $\mathcal{M} \in Md_S$, $\tau \in Tm_s$, $s \in S$ and let e be an evaluation. We first define the value $e(\tau)$ of τ in $\mathcal{M}$ under evaluation e.

1. If $\tau = x \in X_s$, then $e(\tau) = e_s(\tau)$.
2. Suppose $f \in Op_{\sigma;s}$ with $\sigma = s_1 \ldots s_n$ and $\tau_1 \in Tm_{s_1}, \ldots, \tau_n \in Tm_{s_n}$. Then $e(f(\tau_1, \ldots, \tau_n)) = f^{\mathcal{M}}(e(\tau_1), \ldots, e(\tau_n))$.
3. Now let $\varphi \in Fm_S$. We define $\mathcal{M} \models \varphi[e]$, i.e. "$\mathcal{M}$ satisfies φ under the evaluation e (of the variables)".
   (a) If $\varphi \equiv (\tau_1 = \tau_2)$ for some $\tau_1, \tau_2 \in Tm_s$ and $s \in S$, then $\mathcal{M} \models \varphi[e]$ if $e(\tau_1) = e(\tau_2)$.
   (Note: the relation $\equiv$ stands for "syntactically equivalent" or "of the form".)

Suppose $\varphi \equiv P(\tau_1, \ldots, \tau_n)$ for some $P \in Rel_{s_1 \ldots s_n}$ and $\tau_1 \in Tm_{s_1}, \ldots, \tau_n \in Tm_{s_n}$. Then $\mathcal{M} \models \varphi[e]$ if $(e(\tau_1), \ldots, e(\tau_n)) \in P^{\mathcal{M}}$.

(b) Let $\varphi, \psi \in Fm_S$ and $x \in X_s$ for some $s \in S$. Then
$\mathcal{M} \models (\varphi \wedge \psi)[e]$ if $\mathcal{M} \models \varphi[e]$ and $\mathcal{M} \models \psi[e]$,
$\mathcal{M} \models (\neg\varphi)[e]$ if it is not the case that $\mathcal{M} \models \varphi[e]$,
$\mathcal{M} \models (\exists\, x\varphi)[e]$ if there is an element $d \in D_s$, such that
$\mathcal{M} \models \varphi[e(x/d)]$, where for each $y \in \cup_{s \in S} X_s$

$$e(x/d)(y) = \begin{cases} e(y) & \text{if } y \neq x \\ d & \text{if } y = x. \end{cases}$$

We say $\mathcal{M} \models \varphi$ ("$\mathcal{M}$ satisfies φ"), if $\mathcal{M} \models \varphi[e]$ for every evaluation e.

Given a set Ax of formulas (axioms), we say that "$\mathcal{M}$ satisfies Ax" or "$\mathcal{M}$ is a model of Ax", i.e. $\mathcal{M} \models Ax$, if $\mathcal{M} \models \varphi$ for every $\varphi \in Ax$.

Notice that a set Ax of axioms selects from all possible models of our language exactly those which *satisfy* Ax, i.e. are models of Ax (see Definition 5.5). For example, the goal formula says that we are looking for a shelf system whose length does not exceed the length of the wall which we are furnishing, and whose shelves are maximally filled with the objects, moreover, all objects are placed somewhere in the shelf system. This axiom has many models with different arrangements. We will need some kind of a "recipe" to filter out those models which exactly fit the intended design model. (An example of such a "recipe" is ADJ's "initial algebra model" – see e.g. [ADJ, 1976] and [Ehrig et al., 1985]).

This step completes the definition of the logic L_S. For more details on first order logic see e.g. [Monk, 1976].

6. The Specification of a Shelf System

6.1 The Sorts

We will use the following notation for sorts:

ob for the sort of objects to be placed on the shevles. We will use unary relations to distinguish between paperback and hard cover books, stereo elements and speakers.

nat for the natural numbers. The natural numbers come with the usual relations and operations, like $<$, $\leq$, $+$, min, max, ... We will use these same symbols in formulas and assume that their interpretation is the usual one.

sh for shelves. We model shelves by sets of objects. So there are two kinds of shelves we talk about: sets of objects to model shelves and real shelves (of a given fixed length $length\_of\_sh \in \omega$).

u for shelf units. Shelf units are sequences of an upright and 36 shelves. The empty set of objects is used as a shelf whenever for some reason a shelf-upright connection (i.e. a hole or notch) is not to be used.

sy for shelf systems. Shelf systems are sequences of 10 units. (We assume that 10 is the largest number of units that can be built into one system.)

6.2 The Relation and Operation Symbols

There are four relations on objects, i.e. PB, HC, ST, $SP \in Rel_{ob}$, to determine the set of paperback and hard cover books, stereo elements and speakers, respectively.

We also have constant symbols of sort ob to denote the actual objects: a^i_{pb} $(i = 1, \ldots, k_{pb})$, a^i_{hc} $(i = 1, \ldots, k_{hc})$, a^i_{st} $(i = 1, 2, 3)$, a^i_{sp} $(i = 1, 2)$, for the k_{pb} paperback books, the k_{hc} hard cover books, the three stereo elements, and the two speakers, respectively.

We will also need the constants $length\_of\_pb$ ($height\_of\_pb$), $length\_of\_hc$ ($height\_of\_hc$), $length\_of\_st$ ($height\_of_st$), and $length\_of\_sp$ ($height\_of\_sp$) of sort nat, to denote the length (height) of the paperback books, the hard cover books, the stereo elements, and the speakers, respectively. We assume that all paperback books are of the same dimensions. The same holds for the three other kind of objects.

For the uprights we use the constant symbols a^i_{up} $(i = 1, \ldots, k_{up}) \in Op_{\lambda;up}$. In addition, we use the constant symbol $nil_{up} \in Op_{\lambda;op}$, which is used to construct an empty unit, which, in turn, is used whenever the shelf system needs less then 10 units. So $\langle nil_{up}, \emptyset, \ldots, \emptyset \rangle$ is the empty unit, while a unit $\langle a^i_{up}, \emptyset, \ldots, \emptyset \rangle$ consists only of an upright and no shelves. Such a unit is used to end the shelf system.

For shelves we have a constant symbol $length\_of\_sh \in Op_{\lambda;nat}$, which denotes the length of the real shelves. We also have an operation $length_{sh} \in Op_{sh;nat}$ to provide the length of the "sets of objects" – shelves. We further have the "element of" relation (symbol) $\in$ in $Rel_{ob\,sh}$. We will use this symbol in infix notation. The relation symbol $full \in Rel_{sh}$ stands for the property of a shelf of being maximally filled with objects.

Since units are sequences of an upright and shelves, we have a tupling operation symbol $\langle\ \rangle_u \in Op_{up\,sh\ldots sh;u}$ and projections $p_0 \in Op_{u;up}$ and $p_1, \ldots, p_{36} \in Op_{u;sh}$. We agree to write $\langle x^{up}, x^{sh}_1, \ldots, x^{sh}_{36} \rangle_u$, rather than $\langle\ \rangle_u\ (x^{up}, x^{sh}_1, \ldots, x^{sh}_{36})$.

The function symbol $length_u \in Op_{u;nat}$ stands for the function which provides the lengths of a unit (which is 0 or $length\_of\_sh$). Notice that we assume uprights have no length.

For the shelf systems we also have a tupling operation symbol $\langle\ \rangle_{sy} \in Op_{u\ldots u;sy}$ and projections $q_1, \ldots, q_{10} \in Op_{sy;u}$. The notational agreement on the use of $\langle\ \rangle_u$ applies here, too. The length of a shelf system is given by the function represented by $length_{sy} \in Op_{sy;nat}$.

The length of the wall to be furnished with the shelf system is given by the constant symbol $length\_of\_wall \in Op_{\lambda;nat}$.

6.3 The specification: Goals, Parameters and Constraints

We agree that all the free variables in the formulas to follow henceforth are to be read as universally quantified.

- Our goal is to find a shelf system whose length does not exceed the length of the wall which we are furnishing, and whose shelves are maximally filled with the objects. Moreover, all objects are placed somewhere in the shelf system. Formally:

$$\begin{aligned}
&\exists\, x^{sy}(length_{sy}(x^{sy}) \leq length_of_wall \wedge\ \wedge_{j=1}^{10} \wedge_{i=1}^{36} full(p_i(q_j(x^{sy}))) \wedge \\
&\wedge_{k=1}^{k_{pb}} \vee_{j=1}^{10} \vee_{i=1}^{36} (a_{pb}^k \in p_i(q_j(x^{sy}))) \wedge \\
&\wedge_{k=1}^{k_{hc}} \vee_{j=1}^{10} \vee_{i=1}^{36} (a_{hc}^k \in p_i(q_j(x^{sy}))) \wedge \\
&\wedge_{k=1}^{3} \vee_{j=1}^{10} \vee_{i=1}^{36} (a_{st}^k \in p_i(q_j(x^{sy}))) \wedge \\
&\wedge_{k=1}^{2} \vee_{j=1}^{10} \vee_{i=1}^{36} (a_{sp}^k \in p_i(q_j(x^{sy})))).
\end{aligned}$$

- To make sure that shelves are indeed sets of objects, we need the following axiom (of extensionality):

$$((x^{ob} \in x_1^{sh}) \longleftrightarrow (x^{ob} \in x_2^{sh})) \longleftrightarrow (x_1^{sh} = x_2^{sh}),$$

  i.e. shelves are uniquely determined by their elements (i.e. their objects).

- The constants representing a certain type of objects (e.g. paperback books) are in the appropriate relation (e.g. PB):

$$\wedge_{i=1}^{k_{pb}} PB(a_{pb}^i) \wedge\ \wedge_{i=1}^{k_{hc}} HC(a_{hc}^i) \wedge\ \wedge_{i=1}^{3} ST(a_{st}^i) \wedge\ \wedge_{i=1}^{2} SP(a_{sp}^i).$$

- The sum of the lengths of the elements of a shelf equals the length of the shelf:

$$\begin{aligned}
&\exists\, x_1^{nat} \ldots \exists\, x_{k_{pb}+k_{hc}+5}(length_{sh}(x^{sh}) = \Bigg(\sum_{i=1}^{k_{pb}} x_i^{nat} \cdot length_of_pb \\
&+\sum_{i=1}^{k_{hc}} x_{k_{pb}+i}^{nat} \cdot length_of_hc + \sum_{i=1}^{3} x_{k_{pb}+k_{hc}+i}^{nat} \cdot length_of_st \\
&+\sum_{i=1}^{2} x_{k_{pb}+k_{hc}+3+i}^{nat} \cdot length_of_sp\Bigg) \wedge \wedge_{i=1}^{k_{hc}} (((a_{pb}^i \in x^{sh}) \rightarrow x_i^{nat} = 1) \wedge \\
&(\neg(a_{pb}^i \in x^{sh}) \rightarrow x_i^{nat} = 0) \wedge\ \wedge_{i=1}^{k_{hc}} (((a_{hc}^i \in x^{sh}) \rightarrow x_i^{nat} = 1) \wedge \\
&(\neg(a_{hc}^i \in x^{sh}) \rightarrow x_i^{nat} = 0)) \wedge\ \wedge_{i=1}^{3} (((a_{st}^i \in x^{sh}) \rightarrow x_i^{nat} = 1) \wedge \\
&(\neg(a_{st}^i \in x^{sh}) \rightarrow x_i^{nat} = 0)) \wedge\ \wedge_{i=1}^{2} (((a_{sp}^i \in x^{sh}) \rightarrow x_i^{nat} = 1)) \wedge \\
&(\neg(a_{sp}^i \in x^{sh}) \rightarrow x_i^{nat} = 0))).
\end{aligned}$$

- The length of the shelves as sets of objects is less than or equal to the length of the real shelves:

$$length_{sh}(x^{sh}) \leq length\_of\_sh\,.$$

- A shelf is full if no new object fits on it:

$$\begin{aligned} &full(x^{sh}) \longleftrightarrow \\ &length\_of\_pb > (length\_of\_sh - length_{sh}(x^{sh})) \wedge \\ &length\_of\_hc > (length\_of\_sh - length_{sh}(x^{sh})) \wedge \\ &length\_of\_st > (length\_of\_sh - length_{sh}(x^{sh})) \wedge \\ &length\_of\_sp > (length\_of\_sh - length_{sh}(x^{sh})) \end{aligned}$$

- No object can be placed on different shelves in any shelf system:

$$\Big\{\neg(x^{ob} \in p_i(q_j(x^{sy})) \wedge x^{ob} \in p_k(q_\ell(x^{sy}))):\ j,l = 1,\ldots,10;\ i,k = 1,\ldots,36;\ j \neq \ell \quad \text{or} \quad i \neq k\Big\}.$$

- The tupling operation $\langle\ \rangle_u$ and the projections p_i are inverses:

$$\begin{aligned} &p_0\left(\left\langle x^{up}, x_1^{sh}, \ldots, x_{36}^{sh}\right\rangle_u\right) = x^{up} \wedge \\ &\wedge_{i=1}^{36}\left(p_i\left(\left\langle x^{up}, x_1^{sh}, \ldots, x_{36}^{sh}\right\rangle_u\right) = x_i^{sh}\right) \wedge \\ &(x^u = \langle p_0(x^u), p_1(x^u), \ldots, p_{36}(x^u)\rangle_u)\,. \end{aligned}$$

- A unit is of length $length\_of\_sh$ if it has a non-empty shelf, and 0 otherwise:

$$length_u(x^u) = \max\{length_{sh}(p_i(x^u)):\ i = 1,\ldots,36\}\,.$$

- If a shelf in a unit contains an element which is taller than 2" (which is the shelf distance), then appropriately many shelf connections immediately above it are not to be used, i.e. are to be occupied with the empty set of objects:

$$\begin{aligned} &\Big\{(\vee_{i=1}^{k_{pb}}(a_{pb}^i \in p_j(u)) \rightarrow \\ &(p_{j+1}(u) = \emptyset \wedge \ldots \wedge p_{\min\{36,j+[length_{pb}/2]\}}(u) = \emptyset)) \wedge \\ &(\vee_{i=1}^{k_{hc}}(a_{hc}^i \in p_j(u)) \rightarrow \\ &(p_{j+1}(u) = \emptyset \wedge \ldots \wedge p_{\min\{36,j+[length_{hc}/2]\}}(u) = \emptyset)) \wedge \\ &(\vee_{i=1}^{3}(a_{st}^i \in p_j(u)) \rightarrow \\ &(p_{j+1}(u) = \emptyset \wedge \ldots \wedge p_{\min\{36,j+[length_{st}/2]\}}(u) = \emptyset)) \wedge \\ &(\vee_{i=1}^{2}(a_{sp}^i \in p_j(u)) \rightarrow \\ &(p_{j+1}(u) = \emptyset \wedge \ldots \wedge p_{\min\{36,j+[length_{sp}/2]\}}(u) = \emptyset)):\ j = 1,\ldots,36\Big\}. \end{aligned}$$

- The tupling operation $\langle\ \rangle_{sy}$ and the projections q_i are inverses:

$$\wedge_{i=1}^{10} \left(q_i \left(\langle x_1^u, \ldots, x_{10}^u \rangle_{sy} \right) = x_i^u \right) \\ \wedge \left(x^{sy} = \langle q_1(x^{sy}), \ldots, q_{10}(x^{sy}) \rangle_{sy} \right).$$

- The length of a shelf system is the sum of the lengths of its units:

$$length_{sy}(x^{sy}) = \sum_{i=1}^{10} length_u(q_i(x^{sy})).$$

- The first unit of each system has a non-empty shelf, while the last unit consists only of an upright:

$$\vee_{i=1}^{36} (p_i(q_1(x^{sy})) \neq \emptyset) \wedge \\ \exists\, x^{up}\, (q_{10}(x^{sy}) = \langle x^{up}, \emptyset, \ldots, \emptyset \rangle_u \wedge x^{up} \neq nil_{up}).$$

- Finally, we give an example for a constraint. The speakers have to be placed at ear level, which is between the 15-th and the 20-th shelf, as well as in different units:

$$\exists\, x_1^u \exists\, x_2^u\, (\vee_{i=15}^{20} (a_{sp}^1 \in p_i(x_1^u)) \wedge \vee_{i=15}^{20} (a_{sp}^2 \in (x_2^u)) \wedge x_1^u \neq x_2^u).$$

7. Conclusion

Our purpose in this paper is to develop a framework for the formal specification of the design process. For this we propose to use many-sorted extensions of classical first order logic. The specification we gave is far from being complete, but it shows the basic concepts.

The next step in our research is to provide a proof system. The proof system is based either on resolution or algebraic specification. The resolution derives the goal from the axioms. Algebraic specification generates our models from the constants.

Design solutions satisfy the predefined constraints. In some cases it is possible to attach values to the constraints and quantify the degree of satisfaction. By this we can tell which solutions satisfy which set of constraints to what degree. This allows us an ordering on the set of solutions. Also, solutions might satisfy new, so far not considered constraints. At the same time, differentiation between degrees of satisfaction may lead us to the use of modal many-sorted logics.

References

[Abadi, 1988] Abadi, M.: *The Power of Temporal Proofs*, preprint of Digital systems Research Center (1988).

[ADJ, 1976] Goguen, J. A., Thatcher, J. W., and Wagner, E. G.: *An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types,* IBM Research Report RC 6487, 1976, also in: Current Trends in Programming Methodology IV: Data Structuring, ed: R. Yeh, pp. 80–144, Prentice Hall, 1978.

[Allen, 1984] Allen, J. F.: *Towards a General Theory of Action and Time,* Artificial Intelligence 23 (1984) 123–154.

[Chaochen, 1987] Chaochen, Z.: *Specifying Communicting Systems with Temporal Logic,* in: Temporal Logic in Specification, eds.: Banieqbal, B., Barringer, H., and Pnueli, A.; Lecture Notes in Computer Science 398, pp. 304–323, Springer, 1987.

[Ehrig et al., 1985] Ehrig, H. and Mahr, B.: *Fundamentals of Algebraic* Specification, Vol. 1, Springer, 1985.

[Goel, 1989] Goel, A. and Chandrasekaran, B.: *Functional Representation of Design and Redesign Problem Solving,* Proceedings of the Eleventh (IJCAI) International Joint Conference on Artificial Intelligence, pp 1394–1399, 1989.

[Gallaire et al., 1984] Gallaire, H., Minker, J. and Nicolas, J.-M.: *Logic and Databases: A Deductive Approach,* Computing Surveys, Vol. 16, No. 2, pp 153–185, 1984.

[Jaffer et al., 1987] Jaffar, J. and Lassez, J.-L.: *Constraint Logic Programming,* in: Proceedings 15th ACM Symposium on Principles of Programming Languages, 1987.

[Markusz, 1981] Markusz, Zs.: *Knowledge Representation of Design in Many-Sorted Logic,* Proceedings of the 7th Int. Joint Conference on Artificial Intelligence, Vancouver, B.C. Canada, 1981.

[Markusz, 1983] Markusz, Zs.: *On Appliction of Many-Sorted Model Theoretical Operators in Knowledge Representtion,* Tanulmányok 192/ 1986, MTA SZTAKI 1986.

[Markusz, 1989] Markusz, Zs.: *Az Építészeti Tervezés Modellezése a Többfajtájú Logika Eszközeivel,* unpublished thesis, 1989.

[McCarthy, 1969] McCarthy, J. and Hayes, P. J.: *Some Philosophical Problems from the Standpoint of Artificial Intelligence,* in: B. Maltzer and D. Michie (Eds.), Machine Intelligence 4, pp 463–502, Edinburgh University Press, Edinburgh, 1969.

[Monk, 1976] Monk, J. D.: *Mthematical Logic, Springer-Verlag, 1976.*

[Mostow, 1985] Mostow, J.: Toward Better Models of the Design Process, The AI Magazine, Spring 1985.

[Navichandra, 1990] Navichandra, D.: *Innovative Design Systems, Where are we and where do we go from here?,* Technical Report CMU-RI-TR-90-01, Robotics Institute, Carnegie Mellon University, 1990.

[Navichandra, 1991] Navichandra, D., Sycara, K. P., and Narasimhan, S.: *Behavioral Synthesis in CADET, a Case-Based Design Tool,* Proc. of the 7th Conference on Artificial Intelligence Applications, pp 217–221, 1991.

[Navichandra, 1991a] Navichandra, D.: *Exploration and Innovation in Design,* Springer-Verlag, 1991.

[Pareto, 1896] Pareto, V.: *Cors d'Economie Politique,* Rouge, Lausanne, Switzerland, 1896.

[Pasztor, 1991] Pasztor A.: *Proving Safety Properties of Programs With and Without Past – A Temporal Approach,* Proceedings of the Ulam Mathematics Conference, April

[Pasztor et al., 1991] Pasztor, A. and Stary, Ch.: *Towards a Logic for User Interface Specifiction,* published in: Proceedings IEEE International Conference on Man, Machine, and Cybernetics, Charlottesville, VA, October 1991.

[Pasztor et al., 1991a] Pasztor, A. and Stary, Ch.: *LUIS – A Logic For Knowledge-Based User Interface Specification,* submitted.

[Pasztor et al., 1991b] Pasztor, A. and Stary, Ch.: *Many-Sorted Temporal Logic for Knowledge-Based User Interface Specification,* Proceedings of Artificial Intelligence and Mathematics, Jan 5-8, 1992, to appear.

[Sacerdoti, 1973] Sacerdoti, E. D.: *Planning in a hierarchy of abstract spaces,* Proceedings of the Third (IJCAI) International Joint Conference on Artificial Intelligence, pp 412–422, 1973.