Proceedings of ICAD2000
First International Conference on Axiomatic Design
Cambridge, MA – June 21-23, 2000

TOWARDS A UNIVERSAL KNOWLEDGE DATABASE FOR DESIGN AUTOMATION

Hod Lipson and Nam P. Suh
`hlipson@mit.edu`
Mechanical Engineering Department, Massachusetts Institute of Technology
77 Massachusetts Ave., Cambridge MA 02139-4307, USA

ABSTRACT

This paper proposes an architecture for a design knowledge database that is aimed at providing a substrate for automation in the engineering design process. We address issues of knowledge representation, knowledge source, and design processes that make use of this knowledge. The knowledge representation is based on the Axiomatic Design view of design as a top-down hierarchical problem solving process. Every step in the decomposition process is regarded as a building block, and is an element of the database. The design step is thus a unit of design knowledge; it may sometimes be a physical component, feature or parameter, but it can be any aggregation of hardware, software and disembodied abstract concepts. A design step primarily contains knowledge describing modifiable parameters, physical behavior and interfaces. All information is stored in a mathematical form, in attempt to alleviate many of the ambiguity and incoherence problems associated with earlier work involving textual, keyword and symbolic representations. Furthermore, the computational representation makes the information more systematically accessible and amenable to fully automated design.

Keywords: design automation, functional representation, axiomatic design, object-oriented.

1 INTRODUCTION

Design methodologies have been a subject of extensive research for several decades. However, it has been acknowledged that one of the main hurdles on the way to design automation is the lack of a scientific foundation and a computational design-space in which design-search could be carried out systematically (Navinchandra, 1991; Suh, 1990). In this paper we set to propose the architecture of a design knowledge database that will provide a substrate for enabling automation in the engineering design process. The approach is based on the Axiomatic Design view of design as a hierarchical top-down problem solving process with each stage in the decomposition being a building block – *a design step*. The design step is thus a unit of design knowledge; it can be a physical component, feature, or parameter, but it can also be any aggregation of hardware, software and disembodied abstract concepts. We shall refer to the content of a design step as *design component*. The design knowledge associated with the component primarily describes modifiable parameters, physical behavior and interfaces of the design component. It also contains new constraints and requirements this design step introduces, and indication as to functional requirements the design step may be used to satisfy. In attempt to create a computational domain for design search based on this view, we address two aspects: (a) the universal representation of *functional* and *integration* information of these components, and (b) the search/assembly approaches that will use this information for problem solving. Fundamental to our approach is that the entire method be fully computational, based on rudimentary principles of design and interactions between components. This is in attempt to alleviate many of the ambiguity and synthesis problems associated with earlier work involving textual, keyword and symbolic representations of knowledge (e.g. Suh and Sekimoto, 1990). At this stage, we focus our representation at describing *elementary* knowledge; we explicitly refrain from attempting to represent high-level concepts and complex designs like automobiles and airplanes. Just as a child must learn to spell before it can write, or to add before it can solve differential equations, we will begin with simple design problems and elementary design knowledge. If we can demonstrate how an unforeseen application of a design principle emerges spontaneously, then the path to truly automated conceptual design will follow

1.1 STRUCTURE OF THIS PAPER

In the following *background* section we analyze the requirements of a design database and automated design process and decompose it into several constituents. We provide an overview and critique of related work on these issues, and indicate where we believe progress can be made. Next, in the *design-step architecture* section we describe our proposed architecture. We then provide details on our ongoing implementation efforts and describe some early results.

2 BACKGROUND

It is generally agreed that engineering design is a process that starts from a need, and follows through specification and conceptual solution, to embodiment and detailed design (French, 1994). Over the last several decades, several successful design methodologies have been developed and used in a variety of applications. Most of these are *prescriptive*, and lead the designer through a sequence of procedures that are geared to ensure important aspects of a design are considered. They provide methods for rank-ordering alternative designs using criteria specified by the designer. Some of the more known methods are the Quality Function Deployment (QFD) method, Pugh matrix and Taguchi loss function (Pahl and Beitz, 1994). While these methods are widely accepted and taught in many schools, they rely heavily on designer creativeness and experience in generating the initial design alternatives to be assessed and in providing criteria for comparison and selection of a particular design. Axiomatic Design (Suh, 1990) also relies on designer knowledge and creativeness when coming up with DPs for a set of given FRs. In fact, it is this *creative* step that precludes most design approaches from being automated – because *a human designer is always required in the process*. Due to time and cognitive constraints, a human designer can conceive and consider only very few of the vast number of possibilities to embody a design, and hence designers need to prune many options in a top-down process based on their knowledge and experience.

2.1 COMPUTATIONAL DESIGN

In a more algorithmically oriented approach, design can be formulated as a search in a problem space (Chandrasekaran, 1990). The design space is then the set of all possible designs. To make this space tractable, a design can be viewed as a finite configuration of components (not necessarily hardware components). The interaction and relationships among the set of components determines whether the design satisfies particular requirements and constraints. In some cases, when the set of components is fixed and the configuration known, a design problem may reduce to a parameter optimization problem. Various well-established optimization techniques such as hill-climbing can then be used (Wilde, 1978). More robust stochastic methods like genetic algorithms may prove better suited in some cases (Wallace *et al*, 1996), as well as fuzzy representations of decision criteria (Antonson *et al*, 1994). However, true conceptual design is *non-parametric:* it is inherently open ended. At the conceptual stage, the set of parameters – *the design space* – is yet unknown, and so the set of parameters to be optimized is thus unspecified.

In contrast to methodological top-down design, *bottom-up* design methods like natural evolution try very many designs and in this respect are inefficient, but do not require, and can even *generate knowledge*. Recently, there have been reports from various fields on open-ended evolutionary design, where systems can evolve by adding more and more components in various ways. Among these, Bently's work on evolutionary design of tables and optical systems (Bently, 1996) and Sim's work on evolution of robot bodies (Sims, 1994) are some examples. Although these works do introduce open-ended designs by permitting the evolutionary process to add more and more parameters, the *type* of parameter that can be added is of a fixed repertoire. Consequently it is convenient to characterize such design spaces as *semi-parametric*. Unlike parametric optimization, such semi-open-ended methods can indeed produce so-called 'creative' solution that the designer has not foreseen. However, the solution domain is still not fully open-ended.

In this work we go a step further: by making the repertoire of components open too, in attempt to establish a truly open-ended design space that is effectively *non-parametric*[1].

2.2 DESIGN AUTOMATION REQUIREMENTS

In order to establish an open ended and fully automated computational design process there are three fundamental issues that need to be addressed:

- **The component representation** – defining a universal component *representation* that can describe virtually any engineering object and knowledge and permit interaction between components to take place in configurations not known *a-priori*.
- **The source of components** – systematically *supplying* the components and knowledge to establish the database.
- **The design algorithm** – finding efficient ways to *integrate* and *evaluate* many configurations of general components.

Universal representations

Perhaps the main obstacle in addressing the first aspect might be that due to the very general scope of design it is a highly unstructured domain, and so reaching a universal framework is difficult. Indeed, at the intersection of Artificial Intelligence and Engineering, much work has been done in providing a representational foundation for engineering knowledge (Forbus, 1988). Primarily, however, this work has yielded keyword-based and symbolic representations that suffer from ambiguity or require strict coordination, respectively. Ideas relevant to describing the architecture of a component can be found in the framework of *functional representation* research (Modarres, 1997). Functional representation tries to provide a framework for representing how a product works, and focuses on what a device is intended to do, the causal process by which it does it, and how component functions interconnect and enable a process. A language of symbolic primitives is used for formatting this information into predicates. Interesting behaviors are selected and associated with named functions, and these are associated with devices and modes (states) and transitions between them. The transitions are annotated with explanatory and predictive information. Hodges (1992) and Goel (1989) both provide a set of basic functions of mechanical interactions. Chittaro (1994) and Lind (1994) provide elements for describing flow related domains, and Keuneke (1991) proposes several, more general categories of functions explicitly dealing with activities such as *making, controlling, preventing,* etc., as means to enforce specific relations between functional predicates. Based on this model, Iwasaki and Chandrasekaran (1993) define the Causal Functional Representation Language (CFRL) to capture and organize the functional, structural and behavioral knowledge of general systems at various level of detail. Similar ideas of abstraction of function and behavior have been proposed by Umeda *et al* (1990). Although these models have been found useful in a variety of domain-specific applications, our basic criticism is that because of the use of symbolic predicates, a component can only be used in one of preconceived or foreseen applications, and only in anticipated configurations provided in the form of templates. Although preconceived templates are important for representing existing design knowledge, components must also be able to connect in *unanticipated* ways if we are to permit new *creative* configurations to emerge. Moreover, although

---

[1] Strictly speaking mathematically, even an infinite pool of parametric components still gives rise to only a semi-parametric space, but for practical purposes it approaches a truly non-parametric domain.

functional representations are very useful for documenting devices, the use of states forces a discrete-like behavior; this limits description to top-level abstract concepts rather than actual physical behavior which is often continuous.

Component Databases

Once a universal component architecture has been formulated, it is necessary to create a database of such components. A design process will then access this database as a source of building blocks for a design. Indeed, there are numerous applications in almost every field of engineering that store and reuse engineering knowledge from databases. However, rarely is the representational architecture universal enough to represent general devices, and rarely is there a realistic mechanism that enables the database's contents to expand and be used outside its original environment and by uncoordinated users. Beyond having a universal representation, making the database general and expandable involves providing a consistent and scalable source of design knowledge. The NIST Design, Process Planning and Assembly Repository (Regli and Gaines, 1997) is one example of a database with these challenges in mind, although components there contain mostly geometric and manufacturing data, and not design rationale information. Similar commercial component databases are also beginning to appear. Miller *et al* (1997) describe a shareable engineering knowledge database based on the functional representation discussed earlier. They explicitly try to allow reuse across different processes, and overcome some limitations of CFRL in this respect. Shin *et al* (1998) describe a repository for software components based on the same principles. Although they address only software objects, they describe similar problems in searching, matching and categorizing representation. Urban *et al* (1996) describe a multidatabase environment containing components in STEP format (Standard for the Exchange of Product Data). Although this database is useful for manual design, it does not contain information that allows an automated process to integrate components together and achieve new functionalities. Fishpick (1997) presents an Object-Oriented Physical Modeling (OOPM) methodology for web-based simulation. He defines a formal approach to constructing both natural and artificial systems using an extension of the classical object-oriented framework. The end result of OOPM design is a model repository, which is integrated with the web and made available to others on the Internet so that models can be constructed in a "plug and play" fashion. We believe that web-based databases provide a scalable medium for distributing information and computation, as well as a natural mechanism for protecting proprietary knowledge and providing commercial incentive. All the above works rely, directly or indirectly, to encode the design knowledge manually.

Design search algorithms

Finally, there is need for a search process that scans the database, integrates design steps in meaningful ways and evaluates the resulting assembled product. There are two main problems here: (a) how to connect (integrate) components, (b) how to search the extremely large space efficiently yet thoroughly. Of particular interest is the DARPA RaDEO project, carried out at the Ohio State University (Chandrasekaran *et al*, 1998). Briefly, the objective of that project is to develop technologies for representing and using object-oriented component libraries for various design tasks. In combining the CFRL language and the shareable engineering knowledge database cited above, they specifically address object representation, exploration of very large design spaces and automated design analysis and criticism. Their work involved applications to design of hybrid-electric vehicles, conceptual designs of process plants for chemical synthesis and mechanical gear design for helicopter transmission. The large domain has been reduced significantly by a dominance-filtering process where designs dominated by other designs over all dimensions of evaluation are removed. The problem of finding out how to interface components to yield valid configurations is overcome using *templates*. These are generic-components that show how certain components can come together. Hence components can only be assembles in ways pre-coded into templates[2].

To summarize, it is evident that there is a recognized need to formulate a general computational domain containing building blocks for design. However, various attempts to do so have not been able to demonstrate that both (a) their approach is *general*, and not custom tailored to a specific domain, and (b) it is actually capable of producing *new concepts* that have not been explicitly pre-coded into the system in one way or another. Hence we see the challenge in

- A universal representation,
- A knowledge source
- A synthesizing design process

2.3 AXIOMATIC DESIGN THEORY

We base our approach on Axiomatic Design Theory (Suh, 1990) which is briefly described here. The guiding principle of Axiomatic Design is that design can be based on a rigorous scientific foundation, rather than on accumulated training. Hence design can be carried out using a few simple axioms and derived theorems, without the need of extensive design experience. The framework of axiomatic design views design as a mapping between several problem domains: the customer domain, the functional domain, the physical domain and the process domain, as illustrated in Figure 1 (Suh, 1997). In each domain, the design is specified using different elements, namely, customer attributes (CAs), functional requirements (FRs), design parameters (DPs), and process variables (PVs), respectively. In addition there are constraints (Cs). The design process starts in the identification of customer needs and attributes, and formulating them as FRs and constraints. These FRs are then mapped onto the physical domain by conceiving a design embodiment and identifying the DPs. There may be more than one solution to this mapping. Each DP is then mapped onto a set of PVs that define it. Each DP typically introduces new FRs, DPs and PVs at a lower level, and so the mapping process iterates by *zigzagging* between domains, until the design can be implemented without further decomposition.

---

[2] This might explain why, as the authors note, no conceptually new design has emerges although over 2 million designs were evaluated.

At the basis of the theory are two axioms that guide the mapping process: the Independence axiom, stating that a mapping should be sought such that DPs satisfy FRs independently, and the Information axiom, which specifies overall information content of the design should be kept to a minimum (alternatively, that the probability of success of the design should be maximized). Axiomatic design has been applied to numerous real-life cases (Suh, 2000).

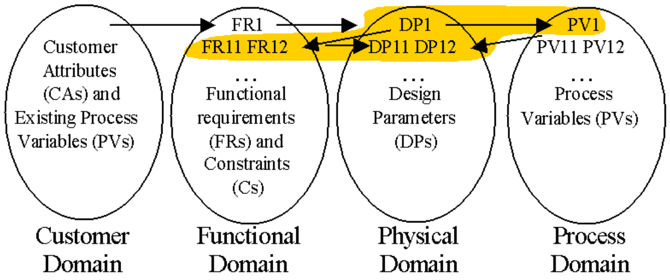

Figure 1. Domain mapping and zigzagging. Shaded area represents a design step

3 PROPOSED DESIGN-STEP ARCHITECTURE

We base our approach on the hierarchical decomposition view taken by axiomatic design theory. In this view the design process can be seen as recursive problem solving where at every stage the current problem is decomposed into smaller sub-problems. Each sub problem can be addressed separately, but only the emergent behavior of the components as a system satisfies the given requirements and constraints. We see each stage in this decomposition process as a building block – *a design step*. In axiomatic design theory, a single cycle of the zigzagging between domains can be viewed as a design component. One such cycle is highlighted in Figure 1.

Provided all domains are defined mathematically, then both the independence axiom and the information axiom can be computed explicitly even without explicit design matrices[3]: The design matrix itself (and consequently dependency measures) can be computed by differentiating functional requirements with respect to design parameters, and information content can be assessed by computing the probability of success of the design using component behavior functions. Thus, the axiomatic design approach lends itself to algorithmic implementation as it provides computational criteria for distinguishing among good and bad designs, as well as an algorithmic procedure for constructing it, given FR-DP sets. The challenge, thus, is to formulate FRs, DPs and PVs in a purely mathematical way, and to provide an algorithmic mechanism to locate and interface sets of DPs and evaluate them as solutions to given FRs.

3.1 A UNIVERSAL REPRESENTATION

A design component can be regarded as a system, having an interface to other components, which are in turn systems too. The components are arranged in a hierarchy based on their functionality: The function of a component is attained jointly by its subcomponents and in unison with its sibling components. A component is defined by its behavior, constraints and requirements all specified explicitly as algorithms or mathematical equations (e.g. differential equations) in terms of state variables (parameters). Table 1 lists the primary attributes of a design component. In addition it has interfaces by which it can connect to other components in the hierarchy. Some of these state variables are local to the component (e.g. its geometry or rigid body parameters), while other are global (e.g. time) or with limited scope. Some of the variables are constants, while others might change. Some might be internal while others might describe interfaces. In absence of any applicable design knowledge, any of the variables might be considered as design-parameter. However, typically a component will contain additional component-specific knowledge as to which variables are primary parameters, and how they can be used effectively,[4] along with a verbal description to be used for documentation.

Table 1. Primary attributes of a design component

Attribute	*Meaning*
Parameters	State variables that describe this component, and their physical units
Interfaces	List of interfaces of the component by which it connects to other components, and corresponding parameters
Behavior	A description of how the state variables of the component change according to the behavior of the component and applicable physical laws (e.g. a set of differential equations or an algorithm)
Constraints	New constraints introduced by this component, in terms of state variables
Requirements	New functional requirements introduced by this component, in terms of state variables. Provides an indication as to whether a particular requirement is satisfied
Knowledge	Component-specific design knowledge, e.g. which parameters are primary, preferred configurations with other components, and FRs satisfied.

The fundamental integration criterion is that interfaces among connected systems be compatible (i.e., of matching physical units). The units are derived from basic units of physics, namely time, distance, mass, information, charge, etc, their derivatives and combinations. Thus, given a set of components, the domain of theoretically possible designs becomes all the arrangements of the components in which interfaces are matched with corresponding interfaces of the same type. The computational domain of possible configurations is well defined. Each interface is assigned a state variable. The state of the system as a whole and the internal state of a component is then a function

---

[3] In a fully automated design process there might not be schematic FR-DP design matrices (X's and O's), but rather an explicit set of equations in terms of system state variables that describe the behavior of the system DPs and satisfaction rate of FRs.

[4] Exploration versus exploitation: In absence of any design knowledge all parameters are of equal importance, and so the number of design permutations to be tried is enormous but potential for creativity is maximized. Evolution in nature is an example of this case. On the other hand, when design knowledge is plentiful, experience guides to consider only selected variables as parameters. Design is then efficient, but creativity is limited. In allowing components to integrate in arbitrary (unforeseen) ways yet permitting explicit domain-specific knowledge to prefer certain parameters and configurations, we can move between these two extremes.

of these variables only. Functional requirements (FRs) and constraints can thus be specified in terms of these variables. Newly introduced functional requirements and sub-components can then be specified in terms of the variables of new interfaces. Issues pertaining as to whether a particular interface is input or output, valid ranges and whether its must be interfaced or can be left under-interfaced, are all constraints specified in terms of the corresponding state variables.

Table 2. Example: descriptions of attributes of an electric linear servo design component

Attribute	*Example*
Parameters	Geometry, mass properties, performance (range, thrust and power), current actuation state.
Interfaces	End effector and base (structural), electric position specification and feedback (control) and power, geometry (contact and collision), possibly also thermal, RFI, magnetic, vibration effects if manufacturer considers them important.
Behavior	Length/force as function of control and time, plus rigid body dynamics and energy consumption.
Constraints	Performance ranges, workable environmental conditions, valid control signals.
Requirements	Power source, structural stability.
Knowledge	Primary parameter to change is performance. Works nicely with specific controller. Alternative specific actuators to try. Doesn't work well with extensive duty cycles.

The information is provided by a component as static attributes or as functions (methods) returning or requesting data. Design components may provide information upon request, and may interrogate the calling process, if they represent a family of possible objects (for example, if they represent a job shopper with a variety of options).

It is important to note that a design component is not necessarily a physical component; it may be an aggregation of hardware, software and disembodied abstract concepts. It may be entirely abstract, specifying how a particular functionality can be achieved using other functions. Abstract components (lacking a physical embodiment) can be thought of as pure design knowledge. For example, while a physical circular-to-linear motion conversion component describes a black-box assembly with an input shaft and output stroke, an abstract component can show how such conversion may be achieved using a nut-and-screw assembly and the constraints they must meet to do so. Another might show a sprocket-and-chain assembly.

Auxiliary attributes

The primary attributes of a component as described above are sufficient to establish a meaningful design space in which it is possible to integrate components and evaluate designs, and ultimately carry out design search. However, additional attributes might be necessary for practical reasons. Briefly, these attributes address the need for a human engineer to extract, interpret, understand, and use solution provided by a search engine. These include (a) unique identification of components, (b) verbal description (documentation) of each of the primary attributes, (c) component visualization, (d) user interface for designer to directly manipulate and examine the behavior of a component, (e) CAD embedding information, (f) e-commerce and legal information.

Inheritance

A fundamental property of a design-component is *inheritance*. Rather than having to define the full functionality of each component, a component will typically inherit most of its properties from a simpler component on which it is based. For example, a *wheel* component is based on a *rigid body* object, and inherits the attributes and internal behavior of a rigid body. It then adds to that the special properties of wheel, such as a constraint on the geometry and (possibly) a shaft interface. Next, a *gearwheel* may inherit most of its properties from a *wheel* component, adding only the properties that distinguish it from its base[5]. Multiple inheritance is also possible: For example, a *telephone* might be derived from both *rigid body* and *communication* base objects. An abstract object will not be derived from a rigid body component. Following down to the base, all objects are derived from the design-component object. Some examples are illustrated in Figure 2.

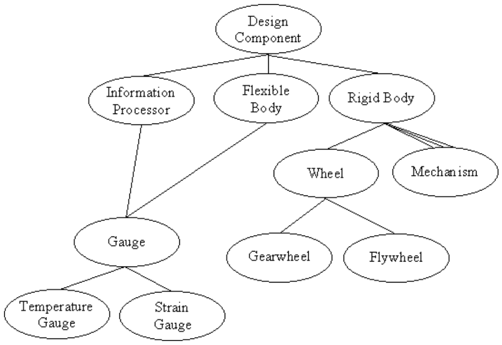

Figure 2. Object diagram for design components

Interface variables are also subject to inheritance (Figure 3). The base interfaces are the physical units, such as time, distance, mass, voltage, charge, temperature and information, and compounds of these such as force or power. More elaborate types of variables can be derived from these: *AC-Power* is derived from *power*, and *mains supply* is derived from *AC-Power* with some constraints on voltage and current. *Real-value* is derived from information, *Fluid-level* is derived from *real value*, and *level-of-fresh-water-tank* may be derived from *fluid-level*, and so forth. A physical implementation may require multiple inheritance: to implement a *water-level* gauge, it might be necessary to have an information channel carried over a voltage channel. Similarly, functional requirements and constraints may also be subject to inheritance themselves.

The inheritance mechanism has several advantages:

1. It lends itself to reuse – one needs to define only those attributes that are unique to one's product. For example, a manufacturer of a *telephone* need not understand rigid body dynamics, although a telephone obeys rigid body dynamics.

---

[5] Note that class names are only as means to reuse mathematical code; the actual name carries no significance in itself and is not used as a search criteria.

2. Derived objects can always serve as base objects. For example, in searching for *wheel* objects, one can use *flywheels* and *gearwheels*. A Telephone can serve as a weight. This is a basic inherent property of design space, and can be used to prune the design space during search: once a certain object is adequate, all objects derived from it need not be considered.
3. It provides a framework for encoding generalizations and design knowledge. When useful assemblies of components are identified, they can be encoded as new complex compound objects.

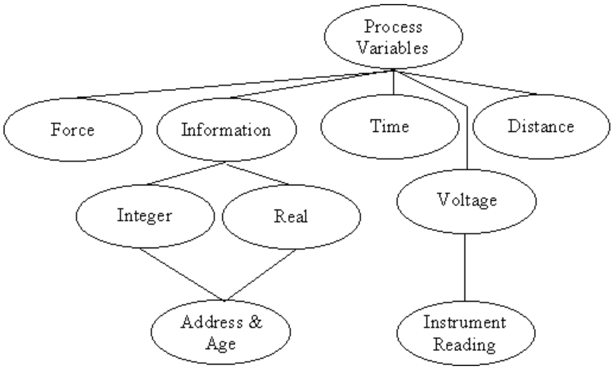

Figure 3. Object diagram for component interfaces and process variables

3.2 KNOWLEDGE SOURCE

Three alternatives are proposed as a source for generating knowledge (design steps) and populating the database. First two are human-dependent, in the sense that they reply on human engineers to provide the knowledge either directly by keying in data or indirectly by sample designs. The third approach is one that relies on a self-discovery of design knowledge using evolutionary techniques.

- **Direct manufacturer key in.** The repository of design components will be Internet based, to be provided by manufacturers of components and knowledge providers (consulting services). By placing such components on the Internet, manufacturers will be able to specify their products' function, interface and required sub-components in a computer-readable format, as well as information regarding possible use contexts of their products. As more components are introduced on the net, so will design engines become more productive and explorative, and more paradigms can be put to test. As design engines become more sophisticated, incorporating new design paradigms, incorporating learning and improved search techniques, so does the incentive grow for manufacturers to put their products in the appropriate format, and to describe their product in more than one way. The rationale behind having a Internet distributed component is twofold. First, it is typically the manufacturer of a product or component that is in the best position to accumulate and provide information about a product. Moreover, the manufacturer typically has the incentive to construct various abstract components that will show how his/her product can be used in different contexts. It is therefore important to allow abstract components to 'recommend' physical embodiments, both to accelerate convergence and to increase manufacturer incentive. These recommendations may be ignored. Second, having the components distributed permits the calculations to be distributed as well, so that computational power increases with problem complexity. Computation is then made more efficient by having manufacturers specialize in computation of their own products' behavior. Distributed (remote) computation also protects proprietary data by releasing the need to disclose all product model data.
- **Learning from examples:** Alternatively, by analyzing existing designs described in a suitable mathematical format, a process might try do identify structure in the design (subcomponents and interfaces) use these elements to populate the database.
- **Self-discovery:** As mentioned earlier, evolutionary design processes are highly inefficient in terms of many futile trials, yet they are capable of actual discovery of design principles and thus generation of knowledge. Its is therefore conceivable that such an evolutionary process, based on a minimal set of 'atom' building blocks, might be used to generate design knowledge and populate the design database automatically.

3.3 THE DESIGN PROCESS

The design search process is the gradual breakdown of the initial design problem into simpler and lower-level design components. There are various approaches suitable for implementing a search process, ranging from the brute-force exhaustive search through the design space on one hand, to a knowledge-intensive approach that requires all of the solution knowledge to be pre-coded into the system, on the other hand. Both of these extreme approaches are not good: While the first extreme is simply impractical at any scale, the second requires databases that cover all possibilities and will tend to eliminate possibility for any innovative solutions. And so there are several approaches in between these two extremes that combine exploration with knowledge exploitation, as well as forms of learning and search.

- **Knowledge based search** Dynamic programming approach that builds multiple decomposition trees while expanding promising branches first, until a complete solution is found. The figure of merit assigned to particular solutions/branches is based on external knowledge in the form of heuristics (which might be specified by the component themselves) or global rules such as the design axioms.
- **Learning** Different forms of learning can be incorporated into the search process, such as assigning figures of merit to known solutions, identifying and reusing subsystems, etc. A system will then learn from its own design experience and improve.
- **Stochastics** Stochastic processes might compensate in absence of design knowledge or might be used to introduce innovation. The "Blind Watchmaker" is an extreme example of this approach.

Typically, a combination of these methods should be used. During the entire search process individual components simulate according to their supplied functions, and the overall system behavior emerges[6].

4 IMPLEMENTATION

In this section we report two efforts in implementation of the design-component architecture. Note that these examples demonstrate only partial aspects of the proposed architecture coupled with a evolutionary source

4.1 ELECTROMECHANICAL LOCOMOTION

The first implementation involves a database containing simple machine and control elements. These design-components were used along with their corresponding constraints and knowledge attributed to automatically design a mechanism capable of locomotion, using an stochastic evolutionary algorithm developed by Lipson and Pollack (2000). Hence this example demonstrates the use of the proposed design architecture coupled with a direct key-in information source and stochastic evolutionary design process.

The machine elements consisted of round elastic bars, linear actuators, ball joints, and step-function control elements. Each of these components was entered into the database with specification of its parameters, behavior and interfaces, as well as constraints, requirements and design knowledge as applicable (see Table 1). The interfaces were defined so that control elements could interface with other control elements, bars and actuators could interface with ball joints, and control elements could interface to actuators. While bars and actuators had a physical embodiment, joints and control elements remained abstract (schematic). No power considerations were modeled. Table 3 lists the primary attributes of two prevailing components.

Each evaluation of a given configuration was obtained by iteratively applying the behavior of each component until the entire system reaches relaxation. The computation was carried out using a physical and control simulator originally developed for robotic simulation (Lipson and Pollack, 1999). Because of the relaxational method of this solver, it cannot account for high momentum dynamics, but only quasi-static motion. Nevertheless, this simplification allows rapid evaluation of candidate designs while retaining a rich repertoire of physical effects, such as friction, material elasticity and failure, collision and contact.

Table 3. Attributes of actuator and control elements used in experiment

ACTUATOR COMPONENT

Attribute	*Values*
Parameters	Length, radius, density, elasticity, yield. Primary **DP**: actuation range.
Interfaces	Position within actuation range [m]
Behavior	Rigid body dynamics, length changes as function of interface value.
Constraints	Collision. Material failure. Manufacturing constraints: Actuation range must be smaller than shaft housing.
Requirements	Must be connected to one or two ball joints and actuation control
Knowledge	Needs to connect to a control unit to move. Pythagoras equation to create a right angle triangle with two other bars.

CONTROL UNIT

Attribute	*Values*
Parameters	Interface amplifications and internal step threshold
Interfaces	Connections to other units [information]
Behavior	Sum of amplified connection and threshold
Constraints	Output in range –1 to 1
Requirements	None
Knowledge	Needs a feedback loop to generate oscillations.

A single functional requirement was specified: *Locomotion*. Mathematically the locomotion FR was formulated as the distance traveled by the center of the designed machine over a fixed period of time. The design process implemented is basically a knowledge-based search, coupled with a stochastic element to cover in absence of design knowledge. The search process started with a null (empty) design. Design components were integrated into the design iteratively, either according to design knowledge (for example: “attach a control unit to an actuator”, or “attach 3 bars to form a triangle” – see Table 3), or by joining components with matching physical interfaces, as well as by modifying component parameters (again, according to database knowledge or in random if no information exists). The more successful designs (according to the FR) were automatically selected to continue to the next stage, and so on, until a certain satisfaction rate of the FR was attained. The design axioms were not implemented in this experiment.

Results

Since relatively little external knowledge has been provided, the “creativity” factor was high: One of the resulting designs is shown in Figure 4 below. This is basically a relatively symmetrical tetrahedral structure, with a freely joined bar dragged on the floor. One of the sides of the tetrahedron is an actuator, controlled by an oscillating control circuit (the circuit and wiring are shown only schematically). When the actuator oscillates, the free bar ratchets against the floor and pushes the entire mechanism forward. This result was obtained after only 4014 evaluations, over 55 minutes of processing on a 500MHz computer.

This design contains two apparently redundant actuators on the base. This redundancy might be attributed to the fact that the Information Axiom was not applied in this experiment. At this point it is also relatively difficult to follow the somewhat alien reasoning in the unfolding design tree. It is important to emphasize, however, that the principle of this design – the

---

[6] One of the main attacks on this approach is that there is no guarantee that simulation of individual components joined together will predict correctly overall system performance. In fact, so-called *complex systems* are often characterized by the converse. However, we hypothesize that to do any design at all, even human engineers typically need to speculate about properties of system configuration based on knowledge of individual components. They succeed because it is specifically those predictable aspects of systems that we use in design; Moreover, Axiomatic Design Theory asserts that design can and should be carried out while focusing on only a single DP for every FR, and that this simplification is crucial.

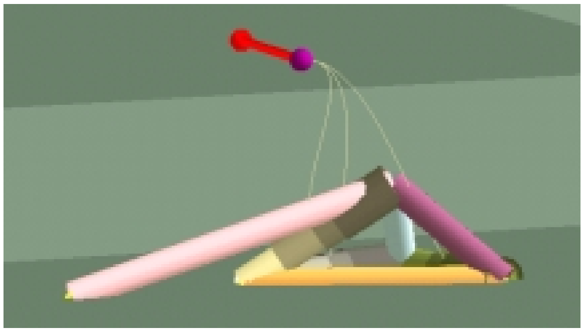

Figure 4. One of the automatically generated designs for the task of locomotion

tetrahedral structure and the ratcheting motion – were not coded anywhere in the system nor provided in any direct or indirect way. We therefore consider this design to be a truly creative solution[7].

4.2 LARGER SCALE DESIGNS

We are now carrying out a second (ongoing) implementation of a design database. This database includes a larger variety of design components comprised of structural elements, control elements, gearwheels and shafts, joints, electric and pneumatic power, along with established design knowledge, and a more realistic physical simulation. With these components we intend to enable a richer universe of Lego™-like designs, and enable more elaborate experimentation. Figure 5 shows a snapshot of the system in development.

5 CONCLUSIONS

In this paper we have proposed a universal design-database architecture, with the intent of creating large-scale design databases containing generically reusable design knowledge. The architecture was developed in accordance with the Axiomatic Design view of design as a hierarchical problem solving process governed by DP-specific knowledge as well as generally applicable rules (axioms and theorems). Fundamental to our approach is that the entire method be fully computational, based on elementary principles of design and interactions between components. We hypothesize that for efficient yet innovative design to occur automatically, a database must support both use of exiting design knowledge, as well as allow for integration of components *without* pre-coded templates. We start with elementary knowledge. Although the design we have demonstrated is still far from being practical, we believe that even this simple form of creativity may indicate a path to truly automated conceptual design in the future.

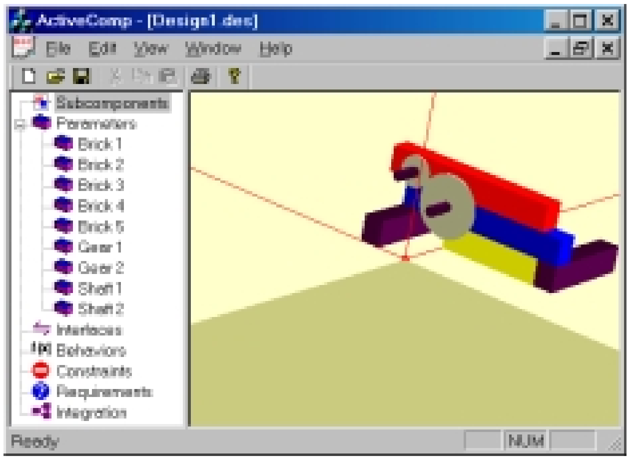

Figure 5. Design molder and database in development

REFERENCES

Antonsson E. K., 1997, "Imprecision in Engineering Design", Proceedings of IDEA"97, Symposium on Intelligent Design in Engineering Applications, *5th European Congress on Intelligent Techniques and Soft Computing* - EUFIT"97, September 97, Aachen, Germany

Bently P. J., 1996, *Generic evolutionary design of solid objects using a genetic algorithm,* Ph.D. thesis, University of Huddersfield, UK..

Chandrasekaran B., Josephson J. R., 1997, "Representing functions as effects", in *Proceedings of the fifth international workshop on advances in functional modeling of complex technical systems,* Paris, France, July 1997, pp. 3-16

Chandrasekaran B., Josephson J. R., Davis J. F., Rizzoni G., Carroll M., Iyer N., Elsass M., Miller D., Miller T. A., 1998, "Functional and diagramatic representation for device libraries", *Annual progress report for period May 1st 1997 – April 30th 1998*. DARPA Order #D591

Chittaro L., Tasso C., Toppano E., 1994, "Putting functional knowledge on firmer ground", *International journal of applied artificial intelligence,* Vol. 8, pp. 239-258

Fishwick, Paul A. *Web-based simulation,* Winter Simul Conf Proc, p 100-102 , 1997

Forbus, K. D., 1988, "Qualitative Physics: Past, present and future", *Exploring Artificial Intelligence*, Shorbe H., (Ed.), Morgan kaufman, pp. 239-296

French M. J., 1994, *Invention and Evolution: Design in Nature and in Engineering*, 2nd Ed., Cambridge University Press

Goel A. K., 1989, *Integration of case based reasoning and model based reasoning for adaptive design problem solving,* Ph.D. Dissertation, Ohio State University

Hodges J., 1992, "Naïve mechanics: A computational model of device use and function in design improvisation", *IEEE Expert*, Vol. 7 No. 1, pp. 14-27

Iwasaki Y., Chandrasekaran B., 1992, "Design verification through function and behavior oriented representation: bridging the gap between function and behavior", in *Artificial Intelligence in Design '92,* Gero J. S., (Ed.), Kluwer Academic Publishers, pp. 597-616

Keuneke A., 1991, "Devive representation: The significance of functional knowledge", *IEEE Expert,* pp. 22-25

Kim S. J., Suh N. P., Kim S.-G., 1991, "Design of software systems based on axiomatic design", *Annals of the CIRP*, Vol. 40 No. 1.

Lind M., 1994, "Modeling goals and functions of complex industrial plants", *Applied artificial intelligence,* Vol 8 No. 2, pp. 259-283

Lipson H. and Pollack J.B., 2000, "Towards Continuously Reconfigurable Self-Designing Robotics", *Proceedings of IEEE.conf. on Robotics and Automation*, SF CA, USA

Miller D. C., Davis J. F., Chandrasekaran B., Josephson J. R., Elsas, M. J., 1997, "Shareable engineering knowledge databases for intelligent system applications, *Computers and Chemical Engineering*, Vol. 21, S77-S82.

Modarres M., (Ed.), 1997, Proceeding of functional modeling workshop, Proceedings of the fifth international workshop on advances in functional modeling of complex technical systems, Paris, France, July 97, published by the Center for Technology Risk Studies, University of Maryland

Pahl, Beitz, 1996, *Engineering Design,* 2nd Ed., Springer Verlag.

Regli, William C. Gaines, Daniel M. *Repository for design, process planning and assembly*, Comput Aided Des v 29 : n 12 , p 895-905 , 1997

Rich E., Knight K., 1991, *Artificial Intelligence*, McGraw-Hill Inc.

Shin, Hyun-Jeong Choi, Il-Woo Kim, Soo-Dong Rhew, Sung-Yul *Design of object-oriented framework repository,* Proc IEEE Int Conf Syst Man Cybern **v 3** , p 2686-2691 , 1998

Sims, K. 1994. Evolving 3d morphology and behavior by competition. In Brooks, R. and Maes, P., editors, *Proceedings 4th Artificial Life Conference*. MIT Press.

Stahovich T. F., Davis R., Shrobe H., "Generating Multiple New Designs from a Sketch," in Proceedings Thirteenth National Conference on Artificial Intelligence, AAAI-96, pp. 1022-1029, 1996

Suh, N.P., Sekimoto, S., 1990, Design of Thinking Design Machine, Annals of CIRP 39/1

Suh N. P., 1990, *The principles of design*, Oxford University Press, New York

Suh N. P., 2000, *Axiomatic Design: Advances and Applications*, to be published by Oxford University Press

Umeda Y., Takeda H., Tomyama T., Yoshikawa H., 1990, "Function, behavior and structure", *Artificial Intelligence in Engineering*, Computational mechanics publications and Springer Verlag, pp. 177-193

Urban, Susan D. Shah, Jami J. Liu, Hong Rogers, Mary 1996, *Shared design manager: interoperability in engineering design,* Integr Comput Aided Eng v 3 : 3 , p 158-176

Wallace D. R., Jakeila M., Flowers W., "Design search under probabilistic specifications using genetic algorithms", *Computer Aided Design*, Vol. 28, 1996