

# From Diagrams to Design: Overcoming Knowledge Acquisition Barriers for Case Based Design

**Michael E. Helms and Ashok K. Goel**  
*Georgia Institute of Technology, USA*

One important and often overlooked implication of case based design systems is the acquisition of cases, which are typically hand crafted by system designers. Not only does this make case acquisition difficult and time consuming, but it may also introduce unanticipated biases and limitations into the system. In this paper we demonstrate a case based design system capable of automatically acquiring cases from diagrams, and of using those cases to solve new design challenges. Moreover, we demonstrate that our representation framework enables general teleological reasoning in design that transcends some domain borders, greatly increasing the versatility and usefulness of the cases thus acquired.

## Motivation and Goals

Previous research establishes that case based design systems can synthesize new models from existing design cases to meet new functional requirements. One important and often overlooked implication of these systems, however, is the acquisition of cases. Typically, cases are hand crafted by system designers, and co-evolve with the system that will use them. Not only does this make case acquisition difficult and time consuming, but it may also introduce unanticipated biases and limitations into the system.

Perhaps the most obvious limitation is that hand crafting takes valuable time and requires skilled knowledge engineers. While a prototype system may only require a handful of cases to prove its theoretic capability, when deployed in more complex and varied domains, resource demands for large numbers of hand crafted design cases may not scale.

A second limitation is the flexibility with which design cases can be specified. Even with a syntactically well defined representational

language, modelers may provide arbitrary specifications for a given model. As models and systems evolve together, specification biases emerge that both increases the difficulty of acquiring new knowledge and limits the range of the cases a system can handle.

Third, while cases crafted for a system may span different domains such as fluid dynamics and electrical circuits, designers may select domains with underlying similarities that facilitate system implementation, but that impose unintentional limitations. For example the domains just mentioned both have a concept of substance flow (electricity flow, fluid flow) with no straightforward analogy in mechanical systems. A system built using cases of electrical and fluid systems may inherently rely on substance flow for problem solving (a system bias), and will encounter difficulties with problems in mechanical system design in which substance flow may be not explicitly modeled.

One result of these limitations in design computing has been that most design systems have been small, almost toy, in their size, including some of our own earlier work (e.g., the Kritik system, [1], [2]). The question then becomes *how can we enable the construction of large scale design systems that may one day address real-world problems?*

Of course this question is not confined to design computing: the same question has arisen earlier in knowledge-based AI. In general, AI has taken two approaches to address the problem. The first approach, exemplified by CYC [3] has been to construct a large general-purpose knowledge base, which can use different inference engines to address different tasks such as understanding design drawings and generating new designs. The second approach, illustrated by KIF [4] and KQML [5], has been to first build wrappers around individual systems (e.g., for diagram understanding and design generation) and to then enable the systems to communicate with one another.

In this paper, we take the second approach toward the construction of large-scale design systems capable of automatic case acquisition. In particular, we begin with two legacy systems: the Archytas system interprets design drawings of physical systems by analogy [6], [7], and the IDeAL system generates new conceptual designs of physical devices also by analogy [8], [9], [10]. Although the underlying ontology for knowledge representation has the same origin, because of the differences in the tasks they address there are important differences between their knowledge representations. Thus, the research issue in this work is how might IDeAL and Archytas communicate with one another, with Archytas serving as a knowledge acquisition front-end for IDeAL and IDeAL serving as an evaluation mechanism for Archytas. More precisely, how might we bridge

the ontological gap between Archytas and IDeAL, where the two systems serve as exemplars of diagram understanding and design generation?

Here, we describe our work to overcome the barrier for design case acquisition by demonstrating a system for case acquisition from diagrams, supported by a very small number of “bootstrap” cases. Validation for the cases so acquired comes from integrating the cases with an existing case library and using these new cases to solve closely related but complex design problems. Our research also demonstrates the usefulness of the SBF knowledge framework to facilitate case acquisition and to generalize across arbitrary design domains.

## **Approach**

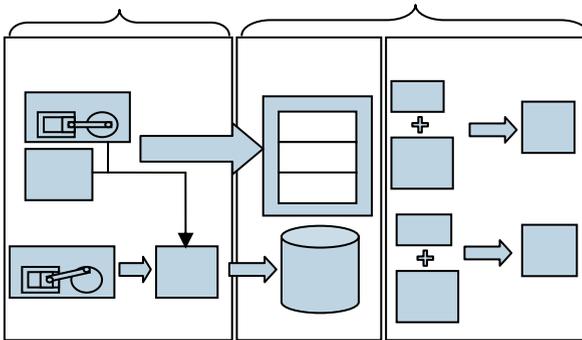
We organized our approach into three distinct phases: 1) Bootstrap and Model Acquisition, 2) Data Extraction and Case Building, and 3) Case Evaluation. Figure 1 provides an overview of the process, as well as the systems involved. The first phase of our approach uses the Archytas system. An initial diagram and DSSBF model are used to bootstrap the system (1a), in order to acquire an SBF model from a new diagram (1b). In the second phase, we extract a limited amount of domain knowledge from the bootstrap case, which is appended to the case based reasoning system (2a), and we also make routine modifications to the newly acquired model that are required to load the model into the existing case library (2b). We will spend the majority of this paper discussing the second phase, as it highlights many of the central challenges of case acquisition. In the third phase, we use the case based reasoning system IDeAL to test the validity of the acquired model.

In subsequent sections, we provide a brief overview of the representation language used by each system, review the fundamental operations of Archytas and IDeAL, and provide additional details on the modifications required to integrate the two in the second phase. We follow with a discussion implementation details and briefly describe the successful use of a model built from a new diagram to solve a complex design problem.

## **SBF and DSSBF Representations**

In earlier work on case-based design [1], [2] we developed Structure-Behavior-Function (SBF) models of physical devices. Both Archytas and IDeAL use the SBF language to represent design cases. The SBF language

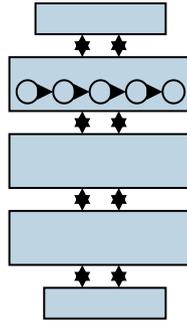
provides conceptual primitives for representing and organizing knowledge of the structures, behaviors and functions of a device. In this representation, the **structure** of a device is constituted of *components* and *substances*. *Substances* have locations in reference to the *components* in the device, while components are related to other components through *connections*. A **function** in the SBF model is a behavioral abstraction and is represented as a schema specifying the behavioral state the function takes as input, the behavioral state it gives as output, and a pointer to the internal causal behavior of the device that achieves the function. The internal causal **behaviors** in the SBF model of a device explicitly specify and explain how the functions of structural elements in the device get composed into device functions. Annotations on the state transitions express the causal, structural and functional contexts in which the transformation of state variables, such as substance, location, properties, and parameters, can occur [11].



**Fig. 1.** A graphical depiction of the approach showing the three main phases and the scope of the systems Archytas and IDEAL.

Previous research [8], [9], and [10] establishes the capability of SBF to 1) represent systems across different domains, 2) enable systems to solve design problems, and 3) represent both case-specific models and case-independent models. However, to facilitate the mapping and transfer of diagrammatic representations to SBF models, more recent work [6], [7] extends the SBF language to include drawing and shape representations. This extended language, called the Drawing-Shape-Structure-Behavior-Function language (DSSBF) uses structure as an intermediate abstraction to relate shapes and spatial relations in a drawing to the behaviors and functions of the device depicted in that drawing. Figure 2 depicts the five tier hierarchical representation of a DSSBF model. Note that a single system may be represented by multiple drawings and multiple shapes,

which are not depicted here for clarity. We bootstrap the initial system with an initial DSSBF model, using the DSSBF specification to establish relationships between diagrams, shapes and their SBF model counterparts. Using analogical reasoning, we can then extract SBF models from new, similar diagrams.



**Fig. 2.** The hierarchical schema of the DSSBF language (adapted from [6]).

In this way, SBF provides a common representational language used to transform drawings to models, and to link those models into a case based reasoning system to create new designs.

## Archytas

Provided with a single source diagram and DSSBF model, the Archytas system constructs an SBF model from a diagram of a physical device. Unlike other approaches that apply domain specific rules e.g. [12], Archytas uses analogical transfer of the SBF model of the device in a similar drawing to build a model from a target diagram. Archytas, however, does not implement a model retrieval task. It is assumed when a diagram is entered, that the proper diagram is assigned from which to transfer the model. The source case contains both a 2-D vector-graphics line diagram of a physical device and a teleological model of the device in the form of a DSSBF model. Given a similar diagram of the same device, the Archytas system aligns the diagrams, and transfers relevant structural, behavioral and functional elements to the new diagram. This process can be repeated for a large number of variations on the initial diagram, including variations on the device state, orientation, dimension and the number of components [6] Archytas cannot account for variations in component shape or drawing perspective. By bootstrapping Archytas with a single instance of a diagram and DSSBF model, the system is capable of

creating 1) the same model from very similar diagrams, and 2) new models for similar systems with different component configurations.

For example given the source diagram Fig 3(a), Archytas reproduces the same model given the target diagram 3(b), even with the slight differences in state, dimension, and polygon composition. Given the same source diagram, Archytas also creates a new model for diagram 3(c) that successfully accounts for the variation in the number of components, in this case two piston assemblies instead of just one.

The Archytas system requires several levels of abstraction. The structural model of the device in the source diagram specifies the components of the depicted device and their interconnections. This model is related to the diagram via an intermediate abstraction of the shape model. Archytas first gathers lines, circles and intersection points into shapes and then finds mappings between the source and the target at this level of abstraction. Then it groups these mappings and transfers shapes from the source to the target. Next it finds a mapping at the shape level, and finally transfers the structural model. Mapping and transfer thus happen iteratively in a loop, working up to each subsequent level.

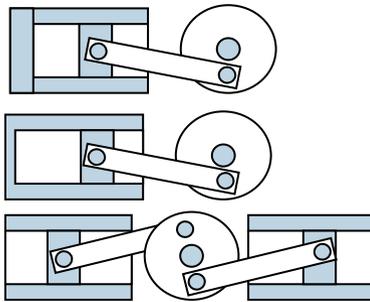
## **IDeAL**

IDeAL is a case based reasoning system that implements an analogical theory of creative design called model-based analogy (MBA). In this theory, case-specific SBF models of past designs enable abstraction of generic teleological mechanisms (GTMs), such as feedback loops and cascading, which are later recalled and adapted in a new candidate design [8],[9]. The computational process of MBA takes as input a specification of functional and structural constraints on a desired design (the target design problem) and gives as output a structure (the solution) that realizes the specified function while satisfying the constraints. The design cases in the case library are case-specific SBF models that specify the functions delivered by past designs, the structure of the design, and the causal behaviors of the design. Cases in memory are indexed by both function and the constraints they satisfy. Information requirement for IDeAL will be discussed in the next section.

## **Information Requirements and Integration**

While our primary research goal is demonstrating a system that enables diagram-based case acquisition and use for design, neither the conversion

of diagrams into SBF models nor the use of SBF cases to solve design challenges were at issue. The Archytas and IDeAL systems addressed each problem independently. The primary challenge of our research was to integrate the two systems, shedding light on the gaps that developed between the two systems. The primary achievement of this research is in understanding the information requirements of the case-based reasoning system as a linked whole, rather than as two independent halves. In so doing, our research exposes information differences in each system, which we categorize as: 1) domain specific differences, 2) representation differences, and 3) system requirements differences.



**Fig. 3.** (a) and (b) illustrate a target and source diagram for the same system, with slight variations in state and dimension. The U-shaped assembly in (a) is made up of three rectangles, but is a single polygon in (b). In (c) two pistons are shown connecting to the same crankshaft, resulting in a different model from (a) and (b) (adapted from [6]).

### Domain Specific Differences

Previous work on IDeAL demonstrates the ability of both SBF and the IDeAL system to operate effectively over multiple domains. Previous cases in IDeAL were taken from domains with straightforward, consistent, and explicitly represented substance flows. For instance, the substance electricity flows from one end of a battery, through a circuit, to the other end of the battery. It is by backward-chaining through the state changes in these substances (via explicitly described transitions) from one point of the system to another, that IDeAL is able to identify causally related components, and ultimately reason about design adaptations.

Archytas cases were taken from the domain of kinematic systems. In these systems, the substances are “motion-types” (e.g. piston-motion, connecting-rod-linear-motion, etc.) that are component specific and do not flow from one component to another. Fig. 4a shows this graphically. The

initial Archytas bootstrap cases have no notion of consistent substance flow.

IDeAL, therefore, cannot reason causally about the Archytas models. Instead, we have to change the point of view of the model to align with IDeAL's requirements.

We do this by thinking of the components as sharing a common flow of kinetic energy, and explicitly describing the flow of energy through points in the system. In the case of a piston-crankshaft assembly, kinetic energy flows from the piston into the connecting rod (shown in Fig 4b.), and from the connecting rod into the crankshaft. This creates a flow characterization and that meets IDeAL's backward-chaining requirements.

Additional complications occur with the assignment of important "points" in the model. Archytas' use of the symbol "points" has to do with the geometry of the diagram. The closest equivalent to IDeAL's notion of points in Archytas is the parametric description of the vectors used to describe *substances*. Within these descriptions, important points are defined as the points in a system on either side of which a component may switch states. For example a piston changes direction (a characteristic of its state) at the top and bottom of the cylinder. IDeAL, however, relies on points to represent the states between which a substance flows. For example, in the case of a piston within a cylinder, IDeAL prefers a point "point-piston" to represent the energy when it is "in" the piston. It uses the point "point-connecting-rod" to represent the place (the connecting rod), to which the energy must flow.

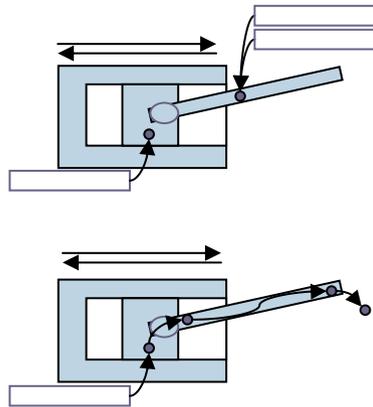
While the notion of energy flowing through points in a system may not be precisely (quantitatively) accurate, for qualitative reasoning about the design dynamics this representation of energy flow is useful.

We classify these problems as domain specific differences, because it is the nature of the domains that shapes the initial representations and creates the gaps. This highlights both a strength and weakness of the SBF representation language. Depending on the context (domain) for which the model will be used, many different valid representations are possible for the same system. For reproduction from diagrams, a component-centric view was adopted. For reasoning to solve a design problem, an energy-transfer view was more productive.

## Representation Differences

Two key representational differences were found in the course of the research: causal loops, and substance specifications. These representation differences occur as arbitrary differences in the way the models were

created, either of which can be viewed as correct, but which we conform to IDeAL standards.



**Fig. 4.** In (a) The Archytas representation of a system is shown to contain substances (e.g. Piston-Linear-Motion) within the component. In (b) The IDeAL representation demonstrates the movement of a common substance throughout the system. The crankshaft was omitted for simplicity.

Causal loops occur when a component or substance begin at some time  $t_0$  in state  $S_0$ , and at some time later, return to a state identical to  $S_0$ , but at time  $t_n$ . For Archytas, using the exact same state object to represent the state of  $S_0$  at time  $t_0$  and at time  $t_n$  is valid. However, because of the backward chaining of IDeAL, this would create an infinite loop during the search for changeable components. As a result, all causal loops represented in Archytas using the same beginning state require a unique state  $S_n$  representation in IDeAL. Because in all Archytas cases the state in question was the terminal state in the model, it was quite easy to automatically alter the models Archytas created as output.

Additional representational choices were made in Archytas, distinguishing an object-type *quantity* from the object-type *substance*. The only difference between the two types is that *quantities* were special *substances* that remained within a given component. Because there were no such types in IDeAL, and moreover, because substance flow between components is required in IDeAL, all such *quantities* were converted to a comparable *substance* object.

## System Requirements Differences

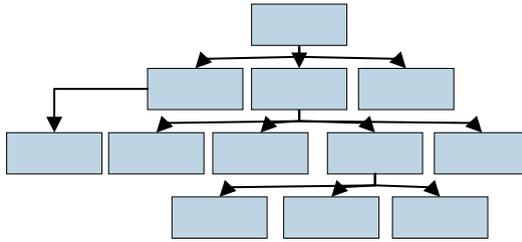
The third kind of difference occurred as a result of the information that IDeAL required to properly index substances and functions, which is generally not included in the SBF and DSSBF language specifications. In particular, because Archytas explored kinematic systems whose main substances were energy expressed as quantities of linear and angular motion, these new substances had to be accounted added to IDeAL's memory. Fig. 5 represents a partial description of the required substance taxonomy. While this taxonomy had to be hand-crafted, it need only be crafted once for each new domain that is added.

Similarly, additional functional indexes are required, but these short, hand crafted indexes are easily added to the Archytas system at a rate no greater than once per bootstrap model created. These functional indexes describe the intended, or primary, function of the system and are common across a class of similar systems (e.g. the heat-exchange functional index is used for all systems whose primary function is to exchange heat between two substances).

Finally, IDeAL is a qualitative reasoning system. As such, it needs to have knowledge about qualitative relationships among system components and substances. While the knowledge *of* causal relationships between components and substances is provided by Archytas' representation, the knowledge of the *kind of* casual relationships is missing. For example, Archytas builds models that represent the linkage (through transitions) between a piston and crankshaft, but it does not necessarily represent their proportionality. Because the knowledge simply doesn't exist (or is at least not required to) it must be added by hand to the IDeAL model.

Therefore the information requirements for converting a diagram using Archytas to a model usable by IDeAL are: 1) successfully reworking the bootstrap DSSBF case into an IDeAL usable form, 2) modifying the indexing structures of IDeAL to successfully work in the new domain, and 3) eliminating any causal loops from Archytas output. Once this has been accomplished and a diagram converted by Archytas, we can test the system by posing a design question to the IDeAL system and analyzing the output.

This sets a different standard for building models from diagrams in that evaluation of those models should also prove application, not just correctness. It also alters the standard for building case-based reasoning systems for design in that they should be capable of not only handling the domains with which they evolved, but also of handling multiple, arbitrary domains.



**Fig. 5.** New substance branches were added for abstract (energy) including kinetic energy, linear motion, friction, and angular motion. In this representation, the three main engineering substances information, energy and material are included, although information has not yet been required.

## Implementation Details and Design Results

The goal of our implementation was to both create a working union of the two systems and, perhaps more importantly, to understand the key knowledge representation differences. While changing the code of Archytas or IDeAL may have created a smoother union or more elegant algorithms, it would have confounded our second goal. Therefore, we made a conscious effort not to change Archytas or IDeAL, but rather to create a bridge between the two. Aside from some minor changes and bug-fixes, nothing in the core algorithms or data structures of IDeAL or Archytas were altered.

Our technical implementation consisted of developing algorithms for converting a model from Archytas output to IDeAL input. The new model is evaluated using IDeAL to successfully solve design problems by using the new model. Here we provide a high-level description of the overall set of algorithms that we applied to an Archytas model to make it compatible with IDeAL. We follow with a discussion of design problem formulation and close with a brief description of the design solution generated by IDeAL.

Our implementation used the initial bootstrap diagram and a converted model for a piston-crankshaft device from the example provided in Figure 3a. We developed the algorithm in Table 1 for conversion of SBF models created by Archytas to IDeAL. This algorithm corresponds to Phase 2: Data Extraction and Case Building, of our system. We follow with a brief description of each step, and some examples of before and after model code.

1. IDeAL requires a separate component memory to draw from during design. We extract and add the names of components from the (*def-component* [component]) line in the Archytas SBF model code.
2. As discussed earlier we identify all new substances (*def-substance* [substance]) and quantities (*def-quantity* [quantity]) and handcraft their inclusion in the substance hierarchy.
3. The relationships among components and substances required by IDeAL for qualitative reasoning processes do not exist in the Archytas models. An SBF modeler with knowledge of the Archytas bootstrap system must handcraft such relationships. For example, the following code expresses the directly proportional relationship between the *linear-motion velocity* of the *piston* (*pmv2*) and the *linear-motion velocity* of the *connective-rod* (*crlv0*):

**Table 1** Algorithm for Phase 2: Data Extraction and Conversion.

|   |
|---|
| <ol style="list-style-type: none"> <li>1) Add components to component memory</li> <li>2) Add substances to substance hierarchy</li> <li>3) Establish principles, principle relationships, and relative values of variables.</li> <li>4) Create Structural Model             <ol style="list-style-type: none"> <li>a) Components</li> <li>b) Substances</li> <li>c) Structural-relationships</li> <li>d) Quantity-relationships</li> </ol> </li> <li>5) Create Behavioral Model             <ol style="list-style-type: none"> <li>a) Transition extraction &amp; loop identification</li> <li>b) State conversion</li> <li>c) Behavioral conversion</li> </ol> </li> <li>6) Establish substance-point correspondences for components</li> <li>7) Create Functional Model</li> <li>8) Create Design Case Index</li> </ol> |
|---|

```
(setf principle-velocity3
(make-instance
  'principle
  :name 'principle-velocity3
  :relation 'directly-proportional-to-substance-property
  :part-1
  '(linear-motion velocity "crlv0")
  :part-2
  '(linear-motion velocity "pmv2"))))
```

The handcrafting of these relationships is the largest bottleneck in the process, largely because the system must be well understood before they can be encoded. There is no reason in principle that the Archytas bootstrap model could not encode these relationships and transfer them with the new model.

In addition at this stage we establish the relative values of variables, so we can know  $crlv0 > crlv1$ , for example.

1. With the exception of point assignment, discussed below, creation of the structural model is straightforward, as correspondences are generally one-to-one. Table 2 demonstrates this.

**Table 2** Code level comparison of Archytas and IDeAL structural elements for the same piston-crankshaft system.

|                                  | Archytas Model Code   | IDeAL Converted Model Code  |
|----------------------------------|---|---|
| a)<br>Components                 | <pre>(def-component piston (piston-crankshaft) :primitive-functions ((allow piston-motion)))</pre>  | <pre>(setf piston (make-instance 'component :name 'piston :points '() :set-of-functions (list allow-piston-motion)))</pre>                        |
| b)<br>Substances                 | <pre>(def-quantity piston-motion (piston-crankshaft) :parameters ((piston-velocity vector (downward upward)) (piston-position vector (cyl-bot cyl-top))))</pre> | <pre>(setf linear-motion (make-instance 'substance :name 'linear-motion))</pre>   |
| c)<br>Component<br>Relationships | <pre>(def-connection piston-crankshaft (cylindrical-joint piston-cylinder))</pre>   | <pre>(setf piston-cylinder-reln (make-instance 'structural-relation :relation 'cylindrical-joint :component1 piston :component2 cylinder))</pre>  |
| d)<br>Substance<br>Relationships | <pre>(def-quantity-relation piston-crankshaft (contains piston piston-motion))</pre>  | <pre>(setf piston-linear-motion-reln (make-instance 'structural-relation :relation 'contains :component1 piston :component2 linear-motion))</pre> |

There are several of items worth noting. First, in the component model, there is no “point” equivalent between Archytas and IDeAL, so we leave the list of points associated with piston empty (we will address this later).

Second, *piston-motion* has been transformed to *linear-motion* a non-component-centric view of substance. We use our substance hierarchy, created in 2, above, to accomplish this mapping. Third, the parameterization of substances that is seen in the Archytas code is used in establishing substance parameters when building the substance hierarchy.

2. Behavioral model conversion is slightly more complicated, as the data structures are different between Archytas and IDeAL. Archytas combines into a single behavior object what in IDeAL are separate objects for state, transition and behavior. We show the differences in Table 3, leaving out the trivial conversion to IDeAL behavior, for brevity.

Close inspection of the code reveals the transition loop in the Archytas piston motion behavior *trans (p-s1 p-s2)* and *trans (p-s2 ps1)*. The causal chain is perpetuated outside of the piston-motion behavior through *under-condition-transition* slot, providing a pointer into the *rod-linear-motion* behavior. Notice, our converted code for IDeAL uses the *next-transition* slot in the state-schema to hand control to the *rod-linear-motion* behavior, and in so doing, breaks the causal loop. Unfortunately the cost is the loss of the notion of the piston returning back to its original state, which is disconcerting.

**Table 3** Code level comparison of Archytas behavior object and converted IDeAL state and transition objects for the same piston-crankshaft system.

| Archytas Model Code   | IDeAL Converted Model Code   |
|---|--|
| <pre>(def-quantity-behavior piston-crankshaft   (piston-motion piston-velocity piston-position)   :state   (p-s1 (piston-position cyl-top)         (piston-velocity downward))   :transition   (trans (p-s1 p-s2)          :using-function          ((allow piston piston-motion))          :under-condition-structure          ((cylindrical-joint piston cylinder))          :due-to-stimulus          (piston-downforce pf))   :state   (p-s2 (piston-position cyl-bot)         (piston-velocity upward)))</pre> | <pre>(setf ps2   (make-instance    'state-substance-schema    :location "piston"    :next-state rls1    :next-transition trans-ps2-rls1    :prev-transition trans-ps1-ps2    :main-substance    (make-instance 'substance                   :name 'linear-motion                   :concept-schema                   kinetic-energy                   :property-list                   '((velocity                     "downward")                    (position "cyl-top")))))</pre> |

|  |  |
|--|--|
| <pre> :transition (trans (p-s2 p-s1)   :using-function   ((allow piston piston-motion))   :under-condition-structure   ((cylindrical-joint piston cylinder)    (revolute-joint piston connecting-rod))   :under-condition-transition   ((rod-linear-motion rl-s2 rl-s1))) </pre> | <pre> (setf trans-ps1-ps2   (make-instance    'transition    :using-function    (list     (make-instance      'using-function      :component piston      :function allow-piston      -motion))    :under-condition-structure    (list piston-cylinder-reln)    :due-to-stimulus    (list (make-instance     'due-to-stimulus     :name 'piston-downforce     :stimulus 'piston-downforce     :magnitude 'pf))) </pre> |
|--|--|

3. We already discussed the differences between point representations in the Archytas model and the IDeAL model. In step 4d We establish the correspondence between the quantity *piston-motion* (*linear-motion*) and the component to which it belongs, *piston*. Additionally, because the substance will flow in and out of the *piston*, we must establish one “inflow” point, and one “outflow” point. Structure points are first class objects in IDeAL, so we must first establish both points, along with its relationship with the substance *piston-motion*

```

(setf piston-point-in
  (make-instance 'structure-point
    :name "piston-point-in"
    :substance-name 'linear-motion
    :input-end '(piston end1)))

(setf piston-point-out
  (make-instance 'structure-point
    :name "piston-point-out"
    :substance-name 'linear-motion
    :output-end '(piston end2)))

```

We then append the points to *piston* object construct.

4. The creation of the high level functional model is trivial, as seen from the side by side code below.
5. The design case index is used by IDeAL to access design cases. It is defined in the IDeAL SBF model, including indexes into structures, behaviors and functions.

We converted an original bootstrap model as outlined in the previous section. Substance taxonomies were added to IDeAL per the representation in Figure 7. We loaded the converted model directly to IDeAL's case library.

For our test, the target design problem posed to IDeAL was to design a piston-crankshaft device that delivered twice as much energy to the crankshaft as the original design (from Archytas) provided. IDeAL was limited, in this case, to using only parts and substances available to it in that design domain (e.g. an alternative part, for instance a larger piston that delivered the required amount of energy, was not available). Our aim was to test whether or not IDeAL was now capable of retrieving a GTM from a different domain and successfully applying it in the new domain, to a model acquired from Archytas.

Prior to asking the design question, IDeAL was asked to solve a similar problem for an electrical system. In this case, IDeAL used the cascade GTM (learned previously from cases involving heat exchangers) to design a system with two batteries in series so that the light bulb in the circuit would produce twice as much light. These are cases that IDeAL is already capable of solving, demonstrating both cross-domain transfer and the ability to solve the amplification problem using the cascade mechanism.

IDeAL's output for the complex design question we posed is represented graphically in Figure 3c, which demonstrates the use of two pistons to deliver twice the energy to the crankshaft. IDeAL was successfully able to identify the nature of the problem, understand that the cascade mechanism was applicable, find the part capable of delivering the additional power, and arrange an SBF model that correctly delivers that additional power to the crankshaft. This test provides a single positive example of how a diagram can be used to automatically produce a model, populate a case-based reasoning tool, and successfully apply cross-domain principles to solve a complex design problem.

## **Issues and Future Research**

The research is limited to the acquisition and testing of a single model from one diagram. This provided a promising proof-of-concept, and exposed a number of deeper issues concerning differences between extracting a model from a diagram and model use in design. Application of the proof-of-concept system to additional diagrams and domains will enable greater generalization of these differences.

The creation and reuse of new designs is a fundamental and necessary strategy for building a scaleable, real-world system. While IDeAL is capable of learning from the new designs it creates, this capability was not validated for the model created by IDeAL in this test case. That is, although a new design solution was created by IDeAL, no effort was made to solve further problems using that new solution.

This research raises more questions than it answers about context-specific model creation. Both Archytas and IDeAL created different models for the same system, despite the common underlying representation framework. Do human designers have multiple versions of the same system represented in their own mental models? In the extreme, do they require a model for each complex task they wish to undertake with a system? Or instead, is there some universal framework with which we represent a system, limited aspects of which we access depending on the context? These questions have important implications as we begin building larger, scalable, multi-functional design systems.

## **Related Work**

We have already discussed in previous sections the preceding work on Archytas and IDeAL. The origin of Ideal's SBF models lies in Chandrasekaran's Functional Representation (FR) scheme for representing the functioning of devices [13], [14]. More recent work by Chandrasekaran has posited multimodal internal representations as a central element to cognitive architecture, for which DSSBF might be viewed as an example. In cognitive engineering, Rasmussen [15] developed similar SBF models for aiding humans in trouble shooting complex physical systems. In computer-aided engineering, Tomiyama developed similar FBS models [16] for aiding humans in designing mechanical systems. In design cognition, Gero [17] developed similar FBS models for understanding the mental information processing of designers in general. In their analysis of verbal protocols of designers working in a variety of domains, [18] found that while (1) novice designers spend most of their time on the structure of the design solutions, spending relatively little time on the design functions or behaviors, (2) expert designers spend significant amounts of time on all three major elements of FBS models: function, behavior, and structure.

The SketchIT systems [19] takes as input a 2D sketch of a physical device, and gives as output multiple designs, augmented with state-transition diagrams. In contrast to SketchIT, which use model construction for diagram understanding, our systems relies on analogical reasoning.

GeoRep [12] is a diagrammatic reasoning system that gives as output a symbolic description of the physical processes represented in a 2D vector-graphics line drawing. Geo-Rep uses a high-level domain-specific domain describer to apply domain-specific rules to produce a final description.

The Structure-Mapping Engine (SME) is a powerful, but content-free analogical mapping system. JUXTA uses SME to compare two nearly identical drawings of a physical process. JUXTA first uses GeoRep for deriving structure from shape, then uses SME to compare the two structures looking for differences in associated attributes, and based on these differences, draw candidate inferences. In contrast, our system constructs a model by analogical transfer from a bootstrap case, and further reasons over that model using model-based analogy.

## Conclusions

End-to-end knowledge acquisition and design is a long-term goal of the DCC community. Here we demonstrate a complete design system capable of both rapid case acquisition by analogy, and design problem solving using those same cases. By working on the end-to-end process, rather than on fragmented processes in small domains, our exploration exposes a number of interesting research issues that will need to be addressed if we are to realize our long-term goals of scalable, multi-functional complex design systems.

A necessary condition for large, scalable case-based design systems is the rapid acquisition of initial cases that minimizes reliance on human engineered cases. Two general frameworks have been developed for development of such large-scale applications: 1) large, general knowledge frameworks disassociated from specific functions, and 2) smaller, functionally dependent representations that can be shared across applications by a knowledge interface framework. In demonstrating a successful deployment of the second approach, we highlight three aspects of a knowledge interface framework that must be addressed for design systems: 1) domain specific differences, 2) system representation differences, and 3) function dependent differences.

In the case of domain specific differences, we believe that crafting new bootstrap cases can account for domain-specific biases that occur in the original development of a system. By relying on bootstrapping and analogical transfer, minimal domain specific hand crafting of a few systems can be leveraged across all diagrams within that domain. The same approach can be applied to system representation differences,

redeploying changes in the bootstrap case throughout the analogically derived cases. While functional dependent differences cannot be analogically transferred from bootstrap cases, we observed that such differences are limited to small, simple extensions of taxonomies, which once developed for a domain, need not be readdressed. In effect, we've shown that once a single bootstrap case is developed and deployed for a system, rapid case acquisition becomes possible for many instances and variations of that system.

Furthermore, while we show that SBF is valid for representing models of systems for both diagrammatic case acquisition and design problem solving, the flexibility of the SBF language allows for many valid representations for the same system. Because each model develops with a bias to meet specific functional requirements, this raises the question of how might we build a representation that meets the needs of large, multi-functional systems?

Because of this representational ambiguity, while a model can be constructed from a diagram, and that model can be shown to be syntactically accurate in that language, that does not guarantee usefulness of the model in an end-to-end system. By demonstrating a more complete system, that is, a system that acquires cases or models from diagrams, and solves problems with those same models, we provide the community with a new standard of evaluation for diagrammatic model acquisition.

## Acknowledgements

This research has been supported by a NSF IIS grant (award number 0534266) on Multimodal Case-Based Reasoning in Modeling and Design.

## References

1. Goel AK, Chandrasekaran B (1989) Functional representation of designs and redesign problem solving. In Proc. 11<sup>th</sup> International Joint Conf. on Artificial Intelligence (IJCAI-89), Morgan Kaufmann: 1388-1394
2. Goel AK, Bhatta S, Stroulia E Kritik (1997) An Early Case-Based Design System. In *Issues and Applications of Case-Based Reasoning in Design*, M. Maher and P. Pu (editors), Mahwah, NJ: Erlbaum, pp. 87-132.
3. Lenat DB, Guha RV, Pittman K, Pratt D, Shepherd M (1990) CYC: Toward programs with common sense. *Communications of the ACM* 33(8): 30-49.

4. Genesereth MR, Fikes RE (1992) Knowledge interchange format, version 3.0 Reference Manual. Technical Report Logic-92-1, Computer Science Department, Stanford University
5. Finin T, Weber J, Wiederhold G, Genesereth M, Fritzson R, McKay D, McGuire J, Pelavin P, Shapiro S, Beck C (1992) Specification of the KQML agent-communication language. Technical Report EIT TR 92-04, Enterprise Integration Technologies, Palo Alto, CA
6. Yaner PW, Goel AK (2006) (a) From diagrams to models by analogical transfer. In D Barker-Plummer et al (eds), *Diagrams 2006*, LNAI 4045: 55-69
7. Yaner PW, Goel AK (2006) (b) From form to function: from SBF to DSSBF. In JS Gero (ed), *Design Computing and Cognition'06*, Springer, Netherlands: 423-441
8. Bhatta S, Goel AK (1996) (a) From design cases to generic mechanisms. *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, Special Issue on Machine Learning 10:131-136
9. Bhatta S, Goel AK (1996) (b) Model-based indexing and index learning in engineering design. *International Journal of Engineering Applications of Artificial Intelligence*, Special issue on Machine Learning in Engineering 9(6):601-610
10. Goel AK, Bhatta S (2004) Design patterns: a unit of analogical transfer in creative design. *Advanced Engineering Informatics* 18(2):85-94
11. Bhatta S, Goel AK (1997) A functional theory of design patterns. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, Morgan Kaufmann: 294-300
12. Ferguson RW, Forbus KD (2000) GeoRep: a flexible tool for spatial representation of line drawing. In *Proc 17<sup>th</sup> National Conf. on Artificial Intelligence (AAAI-2000)*, Austin, Texas. AAAI Press
13. Sembugamoorthy V, Chandrasakaran B (1986) Functional representation of devices and compilation of diagnostic problem-solving systems. In J. Kolodner and C. Riesbeck (eds), *Experience, Memory and Reasoning*, Lawrence Erlbaum, Hillsdale, NJ: 47-73
14. Chandrasekaran B, Goel AK, Iwasaki (1993) Functional representation as design rationale. In *Euro. Workshop on Case Based Reasoning*: 58-75
15. Rasmussen J (1985) The role of hierarchical knowledge representation in decision making and system management. *IEEE Trans. Systems, Man and Cybernetics* 15: 234-243
16. Umeda Y, Takeda H, Tomiyama T, Yoshikawa H (1990) Function, behavior, and structure. In Gero, JS (ed), *Applications of Artificial Intelligence in Engineering V*, Vol. 1, Springer-Verlag, Berlin: 177-193
17. Gero JS, McNeill T (1998) An approach to the analysis of design protocols. *Design Studies* 19(1): 21-61
18. Gero JS, Tham KW, Lee HS (1992) Behavior: A link between function and structure. In Brown, DC, Waldron, MB, and Yoshikawa, H (eds), *Intelligent Computer Aided Design*, North-Holland, Amsterdam: 193-225
19. Stahovich TR, Davis R, Shrobe H (2001) Generating multiple new designs from a sketch. *Artificial Intelligence* 104(1-2): 211-264