Journal of Intelligent Information Systems, 17, 5–21, 2001
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

A Framework for Method-Specific Knowledge Compilation from Databases

J. WILLIAM MURDOCK murdock@cc.gatech.edu
ASHOK K. GOEL goel@cc.gatech.edu
Intelligent Systems Group, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, USA

MICHAEL J. DONAHOO Jeff_Donahoo@baylor.edu
School of Engineering and Computer Science, Baylor University, P.O. Box 97356, Waco, TX 76798-7356, USA

SHAMKANT NAVATHE sham@cc.gatech.edu
Database Systems Group, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, USA

Received October 1, 1999; Revised March 7, 2001

Abstract. Generality and scale are important but difficult issues in knowledge engineering. At the root of the difficulty lie two challenging issues: how to accumulate huge volumes of knowledge and how to support heterogeneous knowledge and processing. One approach to the first issue is to reuse legacy knowledge systems, integrate knowledge systems with legacy databases, and enable sharing of the databases by multiple knowledge systems. We present an architecture called HIPED for realizing this approach. HIPED converts the second issue above into a new form: how to convert data accessed from a legacy database into a form appropriate to the processing method used in a legacy knowledge system. One approach to this reformed issue is to use method-specific compilation of data into knowledge. We describe an experiment in which a legacy knowledge system called INTERACTIVE KRITIK is integrated with an ORACLE database. The experiment indicates the computational feasibility of method-specific data-to-knowledge compilation.

Keywords: design, knowledge compilation, large-scale knowledge bases

1. Motivations, background, and goals

Generality and scale have been important issues in knowledge systems research ever since the development of the first expert systems in the mid sixties. Yet, some thirty years later, the two issues remain largely unresolved. Consider, for example, current knowledge systems for engineering design. The scale of these systems is quite small both in the amount and variety of knowledge they contain, and the size and complexity of problems they solve. In addition, these systems are severely limited in that their knowledge is relevant only to a restricted class of device domains (e.g., electronic circuits, elevators, heat exchangers) and their processing is appropriate only to a restricted class of design tasks (e.g., configuration, parameter optimization).

At the root of the difficulty lie two critical questions. Both generality and scale demand huge volumes of knowledge. Consider, for example, knowledge systems for a specific phase and a particular kind of engineering design, namely, the conceptual phase of functional design of mechanical devices. A robust knowledge system for even this very limited task may require knowledge of millions of design parts. Thus the first hard question is this: How might we accumulate huge volumes of knowledge? Generality also implies heterogeneity in both knowledge and processing. Consider again knowledge systems for the conceptual phase of functional design of mechanical devices. A robust knowledge system may use a number of processing methods such as problem/object decomposition, prototype/plan instantiation, case-based reuse, model-based diagnosis and model-based simulation. Each of these methods uses different kinds of knowledge. Thus the second hard question is this: How might we support heterogeneous knowledge and processing?

Recent work on these questions may be categorized into two families of research strategies: (i) *ontological engineering*, and (ii) *reuse, integration* and *sharing* of information sources. The well known CYC project (Lenat and Guha, 1990) that seeks to provide a global ontology for constructing knowledge systems exemplifies the strategy of ontological engineering. This bottom-up strategy focuses on the first question of accumulation of knowledge. The second research strategy has three elements: reuse of information sources such as knowledge systems and databases, integration of information sources, and sharing of information in one source by other systems (Neches et al., 1991). This top-down strategy emphasizes the second question of heterogeneity of knowledge and processing and appears especially attractive with the advent of the world-wide-web which provides access to huge numbers of heterogeneous information sources such as knowledge systems, electronic databases and digital libraries. Our work falls under the second category.

McKay et al. (1990) have pointed out that a key question pertaining to this topic is how to convert data in a database into knowledge useful to a knowledge system. The answer to this question depends in part on the processing method used by the knowledge system. The current generation of knowledge systems are heterogeneous both in their domain knowledge and control of processing. They not only use multiple methods, each of which uses a specific kind of knowledge and control of processing, but they also enable dynamic method selection. Our work focuses on the interface between legacy databases and legacy knowledge systems of the current generation.

The issue then becomes: given a legacy database, and given a legacy knowledge system in which a specific processing method poses a particular knowledge goal (or query), how might the data in the database be converted into a form appropriate to the processing method? The form of this question indicates a possible answer: *method-specific knowledge compilation*, which would transform the data into a form appropriate to the processing strategy. The goal of this paper is to outline a conceptual framework for the method-specific knowledge compilation technique. Portions of this framework are instantiated in an operational computer system called HIPED (for Heterogeneous Intelligent Processing for Engineering Design). HIPED integrates a knowledge system for engineering design called INTERACTIVE KRITIK (Goel et al., 1996b, 1996c) with an external database represented in Oracle (Koch and Loney, 1995). The knowledge system and the database communicate through IDI (Paramax, 1993).

2. The HIPED architecture

To avoid the enormous cost of constructing knowledge systems for design, HIPED reuses legacy knowledge and database systems, so that we can quickly and inexpensively construct large scale systems with capabilities and knowledge drawn from existing systems. To facilitate easy integration which, in effect, increases overall scalability, we restrict ourselves to making few, if any, changes to the participating legacy systems. The long-term goal is to allow a system to easily access the capabilities of a pool of legacy systems.

The architecture of figure 1 illustrates the general scheme. In this figure, arrowed lines indicate unidirectional flow of information; all other lines indicate bidirectional flow. Annotations on lines describe the nature of the information which flows through that line. Rectangular boxes indicate functional units and cylinders represent collections of data. The

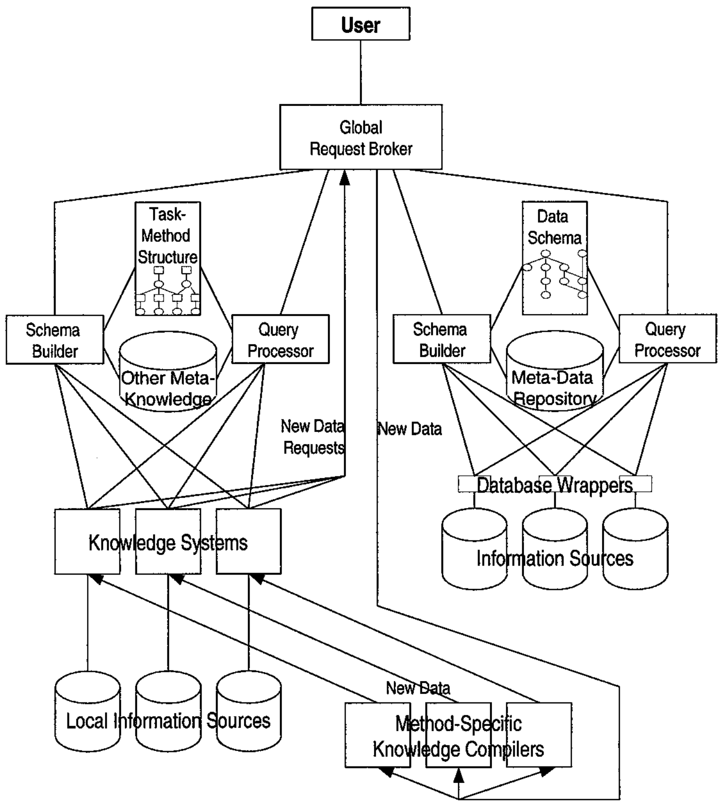

Figure 1. The HIPED architecture. The right side of the figure depicts the integration of databases through partially integrated schema. The left side of the figure shows an analogous integration of knowledge systems. The global request broker at the top provides a uniform interface to both sets of integrated systems so that a request for information may be retrieve from memory and/or deduced by a knowledge system without a specific demand from the user for one approach or the other.

architecture presented in this figure is a projected reference architecture and not a description of a specific existing system. In this section, we describe the entire reference architecture. In the following sections we will further elaborate on a particular piece of work which instantiates a portion of this architecture.

2.1. Database integration

An enormous amount of data is housed in various database systems. Unfortunately, the meaning of this data is not encoded within the databases themselves. At best, the database schema has meaningful names for individual data elements, but often it is difficult to infer all, if any, of the meaning of the data from the schema. This lack of metadata about the schema and a myriad of interfaces to various database systems creates significant difficulties in accessing data from various legacy database systems. Both of these problems can be alleviated by creating a single, global representation of all of the legacy data, which can be accessed through a single interface.

Common practice for integration of legacy systems involves manual integration of each legacy schema into a global schema. That is, database designers of the various legacy systems create a global schema capable of representing the collection of data in the legacy databases and provide a mapping between the legacy system schemas and this global schema (Batini et al., 1986). Clearly, this approach does not work for integration of a large number of database systems. We propose (see the right side of figure 1) to allow the database designers to develop a metadata description, called an augmented export schema, of their database system. A collection of augmented export schemas can then be automatically processed by a schema builder to create a partially integrated global schema which can be as simple as the actual database schema, allowing any database to easily participate, or as complicated as the schema builder can understand; see Navathe and Donahoo (1995) for details on possible components of an augmented export schema. A user can then submit queries on the partially integrated global schema to a query processor which fragments the query into sub-queries on the local databases. Queries on the local databases can be expressed in a single query language which is coerced to the local database's query language by a database wrapper.

2.2. Knowledge system integration

As with databases, a considerable number of knowledge systems exist for design each with their own abilities to perform certain tasks with various methods. Users wishing to access the capabilities of a collection of such systems encounter problems of different interfaces and knowledge representations. Most knowledge systems do not provide an externally accessible description of the tasks and methods they address. As with the database system, one way to integrate legacy knowledge systems is to gather together the designers and construct an ad hoc interface which combines the capabilities of the underlying systems. Once again, this approach does not work for integration of a large number of knowledge systems.

We propose (see the left side of figure 1) to allow knowledge system designers to develop a description, called a task-method structure, of the tasks each local knowledge system can perform (Stroulia and Goel, 1995; Murdock and Goel, 1999). In this approach, a set of knowledge systems, defined at the level of tasks and methods, are organized into a coherent whole by a query processor or central control agent. The query processor uses a hierarchically organized schema of tasks and methods as well as a collection of miscellaneous knowledge about processing and control (i.e., other meta-knowledge). Both the task-method structure and the other meta-knowledge may be constructed by the system designer at design time or built up by an automated schema builder.

The HIPED architecture builds on the concept of mediation systems (Wiederhold and Genesereth, 1997). The new element in HIPED is the task-method structure and the analysis of the local knowledge systems in terms of tasks and methods. It is this structure and analysis that enables method-specific knowledge compilation.

2.3. *Integrated access*

The dichotomy of knowledge systems and database systems is irrelevant to global users. Users simply want answers and are not concerned with whether the answer was provided directly from a database or derived by a process in a knowledge system. We propose the provision of a global request broker which takes a query from a user, submits the query to both knowledge and database systems and returns an integrated result. The broker is an instanve of the more general concept of a mediator (Weiderhold, 1992). Thus it is completely transparent to a global user how or from where an answer was derived.

Furthermore, the individual knowledge systems may, themselves, act as users of the integrated access mechanism. The knowledge systems each have their own local repositories of data but may also find that they need information from a database or another knowledge system. When they need external knowledge, they simply contact the global request broker which can either recursively call the general collection of knowledge systems to generate a response or contact the system of databases. When either a database or a knowledge system generates a response to a request from a knowledge system, the resulting answer is then sent through a method-specific knowledge compiler which does whatever specific translation is needed for the particular system.

2.4. *Method-specific knowledge compilation*

In this paper, we are concerned with the compilation of knowledge from external sources into a form suitable for use by a knowledge system method. Recall that we do not want to alter the knowledge system, so the form of the knowledge may be very specific to the particular method which executed the query; consequently, we call this a “method-specific knowledge compilation.” We will examine the mechanisms behind this component of the reference architecture in more detail in later sections.

2.5. *Information flow*

Consider a design knowledge system which spawns a task for finding a design part such as a battery with a certain voltage. In addition to continuing its own internal processing, the knowledge system also submits a query to the global request broker. The broker sends the query to the query processors for both integrated knowledge and database systems. The database query processor fragments the query into subqueries for the individual databases. The data derived is merged, converted to the global representation, and returned to the global request broker. Meanwhile, the knowledge query processor, using its task-method schema, selects knowledge systems with appropriate capabilities and submits tasks to each. Solutions are converted to a common representation and sent back to the global request broker. It then passes the output from both the knowledge and database system query processors through a method-specific knowledge compiler which coerces the data into a form which is usable by the requesting knowledge system. The resulting battery may be an existing battery which satisfies the voltage specification from a knowledge or database system information source or it may be a battery constructed from a set of lower voltage batteries by a knowledge system.

3. Knowledge compilation

The architecture described in the previous section raises an enormous variety of issues. The one which we want to focus on more closely here is that of knowledge compilation, i.e. the principled transformation of knowledge from one form to another. Large volumes of information can be found in existing databases of components, design schematics, etc. An intelligent design system can make use of this existing data by compiling it into a form which is suitable to its use. There are several closely interrelated perspectives on knowledge compilation as presented in "Knowledge Compilation: A Symposium" (Goel, 1991). A few of these perspectives relevant to this context are:

1. Knowledge implemented in one representational paradigm may be transformed into another. One example of this is the conversion of tuples in a relational database into objects in an object-oriented programming language (Castellanos et al., 1994; Ramanathan, 1994; Fahrner and Vossen, 1995). Another is the collection of information from diverse network resources into a semantically-annotated document in a browsable markup language, as in Stroulia et al. (2000).
2. Knowledge in a generally useful organization may be transformed into a different organization which is more efficient for a specific application. For example, a model of a device may be transformed into a set of rules for diagnosing faults of that device, as per Keller (1991).
3. Declarative knowledge may be transformed into procedural knowledge (Tong, 1991; Anderson and Fincham, 1994), i.e., a specification of a task may be compiled into a procedure for accomplishing that task. This is really an extreme form of the former approach; the result is a knowledge element which typically supports only one application, its execution, but presumably does so in the most efficient way that the compiler can generate.
4. Knowledge of patterns or categories can be inductively inferred from elements. This can be also be seen as an extension of point 2, above; knowledge of instances may be voluminous and difficult to apply to new situations and thus this knowledge is compiled into descriptions or rules which directly enable recognition, classification, etc. Virtually

all work done in the field of machine learning on the subject of classification can be viewed as knowledge compilation from this perspective, e.g., Quinlan (1986), Mitchell (1997).

In this work, we have limited our attention to the first of these topics. We believe that all of these approaches to knowledge compilation are interesting and important. We intend to address all of these concerns in our future research. However, for the purposes of supporting large scale, heterogeneous processing, it is clear that the first issue, that of transforming the structural details of the representation, is inherently fundamental; without a framework for such basic transformations, any of the more complex, sophisticated approaches to knowledge compilation are useless because they simply cannot access any knowledge to compile.

4. INTERACTIVE KRITIK

INTERACTIVE KRITIK is a legacy knowledge system which we have integrated into the HIPED architecture. INTERACTIVE KRITIK is a computer-based design environment. A major component of this system is KRITIK3, an autonomous knowledge-based design system. When completed, INTERACTIVE KRITIK is intended to serve as an interactive constructive design environment. At present, when asked by a human user, INTERACTIVE KRITIK can invoke KRITIK3 to address specific kinds of design problems. In addition, INTERACTIVE KRITIK can provide explanations and justifications of KRITIK3's design reasoning and results, and enable a human user to explore the system's design knowledge.

KRITIK3 evolves from KRITIK, an early multi-strategy case-based design system. Since KRITIK is described in detail elsewhere (see, for example, Goel and Chandrasekaran (1989) and Goel et al. (1996a), in this paper we only sketch the outlines of KRITIK3. One of the major contributions of KRITIK is its device modeling formalism: the Structure-Behavior-Function (SBF) language. The remarkable characteristics of SBF models are: (i) they are functional, i.e. they describe both basic components and complex devices in terms of the function they achieve; (ii) there are causal, i.e. they describe sequences of interactions which constitute the internal behavior of the device; and (iii) they are compositional, i.e. they describe how the function of the device emerges from the functions of the components.

KRITIK3 is a multi-strategy process model of design in two senses. First, while the high-level design process in KRITIK3 is case-based, the reasoning about individual subtasks in the case-based process is model-based; KRITIK3 uses device models described in the SBF language for adapting a past design and for evaluating a candidate design. Second, design adaptation in KRITIK3 involves multiple modification methods. While all modification methods make use of SBF device models, different methods are applicable to different kinds of adaptation tasks.

The primary task addressed by KRITIK3 is the extremely common functions-to-structure design task in the domain of simple physical devices. The functions-to-structure design task takes as input the functional specification of the desired design. For example, the functions-to-structure design of a flashlight may take as an input the specification of its function of creating light when a force is applied on a switch. This task has the goal of giving as output the specification of a structure that satisfies the given functional specification, i.e., a structure that results in the given functions.

KRITIK3's primary method for accomplishing this task is case-based reasoning. Its case-based method sets up four subtasks of the design task: *problem elaboration, case retrieval, design adaptation*, and *case storage*.

The task of problem elaboration takes as input the specification of the desired function of the new design. It has the goal of generating a probe to be used by case retrieval for deciding on a new case to use. KRITIK3 uses domain-specific heuristics to generate probes based on the surface features of the problem specification. The task of case retrieval takes as input the probes generated by the problem elaboration component. It has the goal of accessing a design case, including the associated SBF model whose functional specification is similar to the specification of desired design. KRITIK3's case memory is organized in a discrimination tree, with features in the functional specifications of the design cases acting as the discriminants. Its retrieval method searches through this discrimination tree to find the case that most closely matches the probe.

The task of design adaptation takes as input (i) the specification of the constraints on the desired design, and (ii) the specifications of the constraints on and the structure of the candidate design. It has the goal of giving as output a modified design structure that satisfies the specified constraints. KRITIK3 uses a model-based method of design adaptation which divides the design task into three subtasks: *computation of functional differences, diagnosis*, and *repair*. The idea here is that the candidate design can be viewed as a failed attempt to accomplish the desired specifications. The old design is first checked to see how its functionality differs from the desired functionality. The model of the design is then analyzed in detail to determine one or more possible causes for the observed difference. Lastly, KRITIK3 makes modifications to the device with the intent of inducing the desired functionality.

The method of repair used by KRITIK3 is *generate and test*. This method sets up two subtasks of the repair task: *model revision* and *model verification*. The task of model revision takes as input (i) the specification of the constraints on the desired design, and (ii) the model of the candidate design. It has the goal of giving as output a modified model that is expected to satisfy the constraints on the desired design. KRITIK3 knows of several model revision methods such as component replication or component replacement. KRITIK3 dynamically chooses a method for model revision at run time based on the results of the diagnosis task. Depending on the modification goals set up by the diagnosis task, the system may also use more than one model-revision method.

The task of model verification takes as input (i) the specification of the constraints on the desired design, and (ii) the specification of the structure of the modified design. It has the goal of giving as output an evaluation of whether the modified structure satisfies the specified constraints. KRITIK3 qualitatively simulates the revised SBF model to verify whether it delivers the functions desired of it.

The task of case storage takes as input (i) a specification of the case memory, and (ii) a specification of a new case. It has the goal of giving as output a specification of the new case memory with the new case appropriately indexed and organized in it. Recall that KRITIK3's case memory is organized in a discrimination tree. The system uses a model-based method for the task of storing a new case in the tree. This method sets up the subtasks of *indexing learning* and *case placement*. The SBF model of the new design case enables the learning of the appropriate index to the new case. This directly enables the task of case placement, which stores the model in KRITIK3's local information sources.

5. An experiment with HIPED

We have been conducting a series of experiments in the form of actual system implementations. Some of these experiments have focused on issues most closely related to the data end of the data to knowledge compilation process; these issues include data organization, access, transfer, etc. The experiment we focus on here, however, is more closely connected to the knowledge end of the process. This experiment examines the use of knowledge compiled at run-time in the context of the operation of INTERACTIVE KRITIK.

Figure 2 presents an architectural view of the experiment, in which a legacy knowledge system for design requests and receives information from a general-purpose database system. Since this experiment deals with only one knowledge system and only one database, we are able to abstract away a great many issues and focus on a specific question: method-specific knowledge compilation.

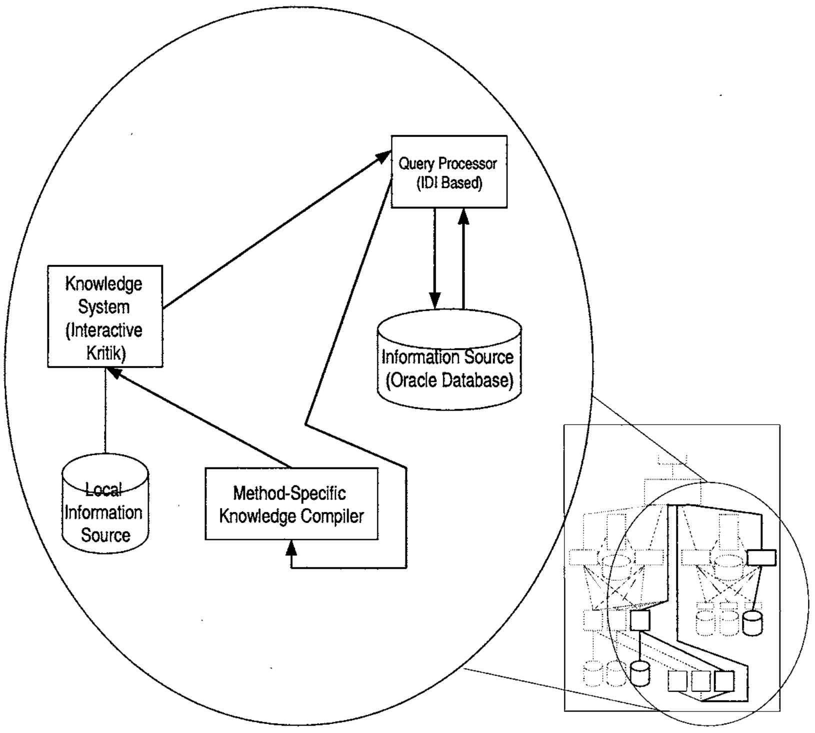
Query Processor (IDI Based)
Knowledge System (Interactive Kritik)
Information Source (Oracle Database)
~~Local~~ Information Source
Method-Specific Knowledge Compiler

Figure 2. The portion of the architecture relating to the proposed solution.

5.1. General method

The overall algorithm developed in this experiment breaks down into four steps which correspond to the four architectural components shown in figure 2:

Step 1: The *knowledge system* issues a request when needed information is not available in its *local information source*.
Step 2: The *query processor* translates the request into a query in the language of the *information source*.
Step 3: The *information source* processes the query and returns data to the *query processor* which sends the data to the *method-specific knowledge compiler*.
Step 4: The *method-specific knowledge compiler* converts the data into a knowledge representation format which can by understood by the *knowledge system*.

All four of these steps pose complex problems. Executing step one requires that a knowledge system recognize that some element is missing from its knowledge and that this element would help it to solve the current problem. Performing step two requires a mechanism for constructing queries and providing communication to and from the external system. Step three is the fundamental task of databases: given a query produce a data item. Lastly, step four poses a challenging problem because the differences between the form of the data in the information source and the form required by the knowledge system may be arbitrarily complex. We focus on the fourth step: method-specific knowledge compilation. The algorithm for the method-specific knowledge compiler implemented in our experimental system is as follows:

Substep 4.1: Database data types are coerced into to knowledge system data types.
Substep 4.2: Knowledge attributes are constructed from fields in the data item.
Substep 4.3: Knowledge attributes are synthesized into a knowledge element.

The particular legacy systems for which we have implemented these algorithms are INTERACTIVE KRITIK and a relational database (Codd, 1970) developed under Oracle. Thus the experimental system serves as an interface between INTERACTIVE KRITIK and our Oracle database.

5.2. An illustrative example

Our experiment takes place during a session in which INTERACTIVE KRITIK is designing an electric light circuit. It has retrieved from its case-memory a model of a circuit which produces light. However, in comparing the functional specification of the retrieved case with the desired functional specification, INTERACTIVE KRITIK determines that the retrieved case does not produce enough light. Consequently, it applies its diagnosis methods to determine components which might be responsible for the amount of light produced. One of the results generated by the diagnosis mechanism is that a higher capacity bulb will lead to the production of more light. Consequently, INTERACTIVE KRITIK may be able to apply the *component replacement* method of model revision. However, in order to apply this method, it must have knowledge of a light bulb of sufficient capacity. No such bulb is available in its local knowledge base. In earlier versions of this system, it would conclude that replacement of the bulb is impossible and thus either attempt to replace a different component or attempt a different model revision method altogether. However, in this version, INTERACTIVE KRITIK has access to an external source of knowledge via the HIPED architecture.

INTERACTIVE KRITIK sends a request for the desired light bulb to the query processor. The request is made as a LISP function call to a function named lookup-database-by-attribute which takes three arguments: a prototype, an attribute, and a value for that attribute. An example of such a call from the system is a request for a more powerful light bulb for which the prototype is the symbol 'L-BULB which refers to the general class of light bulbs, the attribute is the symbol 'CAPACITY, and the value is the string "capacity-more" which is internally mapped within INTERACTIVE KRITIK to a quantitative value.

As a side note, INTERACTIVE KRITIK makes use of both quantitative and qualitative values in its reasoning methods. The details of the interactions between these two kinds of information within the system are moderately complex and outside the scope of this paper. Obviously, the experiment would be more realistic if the external database used quantitative values. This would add another step to the method-specific knowledge compilation process (mapping quantitative to qualitative values) but would not significantly affect the process as a whole.

The query processor uses IDI to generate an SQL query as follows:

```
SELECT  DISTINCT RV1.inst_name
FROM    PROTO_INST RV1, INSTANCE RV2
WHERE   RV1.proto_name = 'l-bulb'
AND     RV1.inst_name = RV2.name
AND     RV2.att_val = 'capacity-more'
```

IDI sends this query to Oracle running on a remote server. Oracle searches through the database tables illustrated in Table 1. The first of these tables, INSTANCE, holds the components themselves. The second table, PROTO_INST, is a cross-reference table which provides a mapping from components to prototypes.

If Oracle finds a result, as it does in this example, it returns it via the method-specific knowledge compiler. In this case, the query generates the string "bigbulb" as the result. The

Table 1. The tables for the Oracle database.

Table INSTANCE			Table PROTO_INST	
Name	Attribute	ATT_VAL	INST_NAME	PROTO_NAME
littlebulb	lumens	capacity-less	littlebulb	l-bulb
bigmotor	watts	power-more	bigmotor	motor
bigbulb	lumens	capacity-more	bigbulb	l-bulb

prototype name and the value are also part of the result, but they are not explicitly returned by the database since they are the values used to select the database entry in the first place. The method-specific knowledge compiler converts the raw data from the database to a form comprehensible to INTERACTIVE KRITIK by using the algorithm described in Section 5.1. In Substep 4.1, the string "bigbulb" is converted from a fixed length, blank padded string, as returned by Oracle, to a variable length string, as expected by INTERACTIVE KRITIK. In Substep 4.2, the attributes of the new bulb are generated. The values "bigbulb" and 'L-BULB are used as the knowledge attributes name and prototype-comp; the values "'CAPACITY, 'LUMENS, and "capacity-more" are combined into a CLOS[1] object of a class named parameter and a list containing this one object is created and used as the parameters attribute of the component being constructed. Finally, in Substep 4.3 these three attribute values are synthesized into a single CLOS object of the component class. The end result of this process is an object equivalent to the one defined by the following statement:

```
(clos:make-instance 'component
  :init-name      "bigbulb"
  :prototype-comp 'L-BULB
  :parameters     (list (clos:make-instance 'parameter
                           :init-name 'CAPACITY
                           :parm-units 'LUMENS
                           :parm-value "capacity-more")))
```

These commands generate a CLOS object of the component class with three slots. The first slot contains the component name, the second contains the prototype of the component, and the third is a list of parameters. The list of parameters contains a single item which is, itself, a CLOS object. This object is a member of the parameter class and has a parameter name, the units which this parameter is in, and a value for the parameter. This object is then returned to INTERACTIVE KRITIK.

Once INTERACTIVE KRITIK has received the description of the bulb, it is consequently able to apply the component replacement method. It replaces the bulb in the case retrieved earlier with the new bulb returned by the query processor. The effects of this substitution are propagated through the model and INTERACTIVE KRITIK verifies that the adapted model does accomplish the requirements which were initially specified. Finally, INTERACTIVE KRITIK presents the revised model to the user and stores it into the case-memory for further reuse in a later problem-solving session.

6. Related research

Knowledge systems for design developed thus far appear incapable of supporting practical design. This critique surely is valid for all laboratory knowledge systems such as AIR-CYL (Brown and Chandrasekaran, 1989), COMADE (Lenz et al., 1996), and our own KRITIK series of systems described at earlier AI in Design conferences (Stroulia et al., 1992; Bhatta et al., 1994; Goel et al., 1996b). But it is also applicable to systems that have directly led to real applications such as R1 (McDermott, 1982), PRIDE (Mittal et al., 1986), VT (Marcus et al., 1988), CLAVIER (Hennessy and Hinkle, 1992), and ASK-JEF (Barber et al., 1992). The problem is the limited scale and scope of knowledge systems for design.

The *ontological engineering* approach to extending the scale and scope of knowledge systems has been the focus of some research. CYC (Lenat and Guha, 1990) pursues this approach at a global level, seeking to provide a single, coherent ontology for constructing knowledge systems. Ontolingua (Gruber, 1993) provides another, domain-specific example of ontological engineering. The bottom-up approach focuses on the second question of accumulation of knowledge: domain-specific and domain-independent ontologies may one day enable interactive knowledge acquisition from external sources and autonomous acquisition of knowledge through learning from experience. But ontological engineering requires the building of new systems based on the common ontology.

In contrast, our work falls under the *information integration* paradigm (Neches et al., 1991; Weiderhold, 1992). This approach emphasizes the reuse of *legacy* information sources such as databases, integration of the information sources, and sharing of information in one source by other systems. This top-down strategy focuses on one part of the scalability issue, namely, heterogeneity of knowledge and processing. Various projects on information integration have focused on different aspects of information integration. For example, KQML (Finin et al., 1994) provides a protocol language for communication among database systems, and KIF (Genesreth and Fikes, 1991) provides a meta-language for enabling translation between knowledge systems. In contrast, (Brodie, 1988) has emphasized the need for integrating knowledge systems and databases. McKay et al. (1990) in particular have pointed out that a key question is how to convert data in a database into knowledge useful to a knowledge system. The answer to this question clearly depends in part on the problem-solving method used by the knowledge system. The work presented here on method-specific knowledge compilation provides one perspective on this issue.

7. Discussion

The complexity involved in constructing a knowledge system makes reuse an attractive option for true scalability. However, the reuse of legacy systems is non-trivial because we must accommodate the heterogeneity of systems. The scalability of the HIPED architecture comes from the easy integration of legacy systems and transparent access to the resulting pool of legacy knowledge. Sharing data simply requires that a legacy system designer augment the existing local schema with metadata that allows a global coordinator to relate data from one system to another, providing a general solution to large scale integration.

While the experiment described in Section 5 shows how data in a general-purpose design database can be compiled into a specific kind of knowledge required by a particular problem-solving method, it raises an additional issue. If the number of the problem-solving methods is large, and each method requires a knowledge compiler specific to it, then the HIPED architecture would require the construction of a large number of method-specific data to knowledge compilers. In the case of knowledge systems for design, which typically contain many problem-solving methods, this itself would make for significant knowledge engineering.

The issue then becomes whether we can identify primitive building blocks from which we can rapidly construct individual method-specific knowledge compilers. In the example discussed in Section 5, it appears that the three steps of the specific method for converting data into knowledge described can all reasonably be considered to be relatively generic units of functionality. Consider Substep 4.1 in the example, coercion of database types into knowledge system types: it is not unreasonable to expect that a wide variety of methods might have the same data coercion requirements and thus be able to use the same data coercion routines in their method-specific knowledge compilers. Further, many knowledge systems for design use representations which are characterized as knowledge elements composed of a set of attribute-value pairs. The general framework for Substeps 4.2 and 4.3 of the algorithm (building attribute-value pairs and then combining them to form a knowledge element) probably can be applied to a wide variety of knowledge-based methods. Furthermore, to the extent that some methods have similar forms and mechanisms for constructing these elements, they might be able to share specific routines. Our experiment suggests that it may be possible to abstract generic components of method-specific compilations. Doing so may partially mitigate the problem of constructing large numbers of method-specific knowledge compilers as individual knowledge compilers might be built from a small and parsimonious set of components. But our experiments with HIPED have not yet demonstrated this; more research is required to fully explore this hypothesis.

The specific experiment described in Section 5 models only a small portion of the general architecture described in Section 2. In a related experiment, we have worked with another portion of the architecture (Navathe et al., 1996). Here, five types of queries that INTERACTIVE KRITIK may create are expressed in an SQL-like syntax. The queries are evaluated by mapping them using facts about the databases and rules that establish correspondences among data in the databases in terms of relationships such as equivalence, overlap, and set containment. The rules enable query evaluation in multiple ways in which the tokens in a given query may match relation names, attribute names, or values in the underlying databases tables. The query processing is implemented using the CORAL deductive database system (Ramakrishnan et al., 1992). While the experiment described in this paper demonstrates method-specific compilation of data into knowledge usable by INTERACTIVE KRITIK, the other experiment shows how queries from INTERACTIVE KRITIK can be flexibly evaluated in multiple ways. We expect an integration of the two to provide a seamless and flexible technique for integration of knowledge systems with databases through method-specific compilation of data into useful knowledge.

The complexity involved in constructing knowledge systems for practical design makes integration of legacy knowledge systems and legacy databases an attractive option. But, insofar as we know, past research on information integration has almost exclusively focused on heterogeneity of information, not on heterogeneity of processing. However, third-generation knowledge systems for design are heterogeneous in that they use multiple problem-solving methods, each of which uses a specific kind of knowledge and control of processing. Thus the goal of constructing third-generation knowledge systems for practical design requires support for heterogeneity of processing in addition to that of information. This itself is a hard and complex goal that requires long-term research. Building on earlier work on integrating knowledge systems and databases systems (Brodie, 1988; McKay et al., 1990), HIPED identifies some issues in supporting heterogeneous information processing and takes a first step towards achieving the goal. In particular, it identifies the task-method structure as providing an analysis of local knowledge systems in terms of the kinds of processing they use. This enables compilation of knowledge in a form appropriate for a specific method of processing.

The issues of generality and scale of intelligent design systems require robust answers to two questions: how to accumulate huge volumes of knowledge, and how to support heterogeneous knowledge and processing? Our approach suggests reuse of legacy design systems, integration of knowledge systems with legacy information sources, and sharing of the information sources by multiple design systems. In addition, given a reusable and sharable information source such as a legacy database, and given a legacy design system that uses a specific processing strategy, the approach suggests the strategy of method-specific knowledge compilers that convert the content of the information into the appropriate form for the processing strategy. HIPED represents a first step towards realizing this strategy.

Acknowledgments

This paper has benefited from many discussions with Edward Omiecinski. This work was funded by a DARPA grant monitored by WPAFB, contract #F33615-93-1-1338, and has benefited from feedback from Chuck Sutterwaite of WPAFB.

Note

1. CLOS stands for Common LISP Object System. CLOS is a mechanism within Common LISP which can be used to represent information in an object oriented framework.

References

Anderson, J.R. and Fincham, J.M. (1994). Acquisition of Procedural Skills from Examples, *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 20, 1322–1340.

Barber, J., Jacobson, M., Penberthy, L., Simpson, R., Bhatta, S., Goel, A.K., Pearce, M., Shankar, M., and Stroulia, E. (1992). Integrating Artificial Intelligence and Multimedia Technologies for Interface Design Advising, *NCR Journal of Research and Development*, 6(1), 75–85.

Batini, C., Lenzernini, M., and Navathe, S.B. (1986). A Comparative Analysis of Methodologies for Database Schema Integration, *ACM Computing Surveys*, 18(4), 325–364.

Bhatta, S., Goel, A.K., and Prabhakar, S. (1994). Analogical Design: A Model-Based Approach. In *Proceedings of the Third International Conference on Artificial Intelligence in Design*, Lausanne, Switzerland.

Brodie M. (1988). Future Intelligent Systems: AI and Database Technologies Working Together. In Mylopoulos and Brodie (Eds.), *Reading in Artificial Intelligence and Databases* (pp. 623–641). Morgan Kauffman.

Brown, D. and Chandrasekaran, B. (1989). *Design Problem Solving: Knowledge Structures and Control Strategies*. London, UK: Pitman.

Castellanos, M., Saltor, F., and García-Solaco, M. (1994). Semantically Enriching Relational Databases into an Object Oriented Semantic Model. In D. Karagiannis (Ed.), *Proceedings of the 5th International Conference on Database and Expert Systems Applications—DEXA-94*, Vol. 856 of *Lecture Notes in Computer Science*, Athens, Greece (pp. 125–134).

Codd, E. (1970). A Relational Model for Large Shared Data Banks, *CACM*, 13(6).

Fahrner, G. and Vossen, G. (1995). Transforming Relational Database Schemas into Object Oriented Schemas According to ODMG-93. In *Proceedings of the Fourth International Conference of Deductive and Object Oriented Databases* (pp. 429–446).

Finin, T., McKay, D., Fritzson, R., and McEntire, R. (1994). KQML: An Information and Knowledge Exchange Protocol. In K. Fuchi and T. Yokoi (Eds.), *Knowledge Building and Knowledge Sharing*. Ohmsha and IOS Press.

Genesreth, M.R. and Fikes, R. (1991). *Knowledge Interchange Format Version 2 Reference Manual*. Stanford University Logic Group.

Goel A.K. (Ed.) (1991). Knowledge Compilation: A Symposium, *IEEE Expert*, 6(2).

Goel, A.K. and Chandrasekaran, B. (1989). Functional Representation of Designs and Redesign Problem Solving. In *Proc. Eleventh International Joint Conference on Artificial Intelligence* (pp. 1388–1394).

Goel, A.K., Bhatta, S., and Stroulia, E. (1966a). Kritik: An Early Case-Based Design System. In M.L. Maher and P. Pu (Eds.), *Issues and Applications of Case-Based Reasoning to Design*. Lawrence Erlbaum Associates.

Goel, A.K., Gomez, A., Grue, N., Murdock, J.W., Recker, M., and Govindaraj, T. (1996b). Explanatory Interface in Interactive Design Environments. In J.S. Gero and F. Sudweeks (Eds.), *Proceedings of the Fourth International Conference on Artificial Intelligence in Design*, Stanford, California.

Goel, A.K., Gomez, A., Grue, N., Murdock, J.W., Recker, M., and Govindaraj, T. (1996c). Towards Design Learning Environments—I: Exploring How Devices Work. In C. Frasson, G. Gauthier, and A. Lesgold (Eds.), *Proceedings of the Third International Conference on Intelligent Tutoring Systems*, Montreal, Canada.

Gruber, T.R. (1993). A Translation Approach to Portable Ontology Specification, *Knowledge Acquisition*, 5(2), 199–220.

Hennessy, D. and Hinkle, D. (1992). Applying Case-Based Reasoning to Autoclave Loading, *IEEE Expert*, 7(5), 21–26.

Keller, R.M. (1991). Applying Knowledge Compilation Techniques to Model-Based Reasoning, *IEEE Expert*, 6(2).

Koch, G. and Loney, K. (1995). *Oracle: The Complete Reference*, 3rd edn. Osborne/McGraw Hill/Oracle.

Lenat, D. and Guha, R. (1990). *Building Large Knowledge Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley.

Lenz, T., McDowell, J., Kamel, A., Sticklen J., and Hawley, M.C. (1996). The Evolution of a Decision Support Architecture for Polymer Composites Design, *IEEE Expert*, 11(5), 77–83.

Marcus, S., Stout, J., and McDermott, J. (1988). VT: An Expert Elevator Designer that Uses Knowledge-Based Backtracking, *AI Magazine*, 9(1), 95–112.

McDermott, J. (1982). R1: A Rule-Based Configurer of Computer Systems, *Artificial Intelligence*, 19, 39–88.

McKay, D., Finin, T., and O'Hare, A. (1990). The Intelligent Database Interface. In *Proceedings of the Eight National Conference on Artificial Intelligence*, Menlo Park, CA (pp. 677–684).

Mitchell, T.M. (1997). *Machine Learning*. New York: McGraw-Hill.

Mittal, S., Dym, C., and Morjaria, M. (1986). PRIDE: An Expert System for the Design of Paper Handling Systems, *Computer*, 19(7), 102–114.

Murdock, J.W. and Goel, A.K. (1999). Towards Adaptive Web Agents. In *Proceedings of the Fourteenth IEEE International Conference on Automated Software Engineering—ASE-99*, Cocoa Beach, FL.

Navathe, S.B. and Donahoo, M.J. (1995). Towards Intelligent Integration of Heterogeneous Information Sources. In *Proceedings of the 6th International Workshop on Database Re-engineering and Interoperability*.

Navathe, S.B., Mahajan, S., and Omiecinski, E. (1996). Rule Based Database Integration in HIPED: Heterogeneous Intelligent Processing in Engineering Design. In *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications*.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., and Swarton, W.R. (1991). Enabling Technology for Knowledge Sharing, *AI Magazine*, 12(3), 37–51.

Paramax (1993). *Software User's Mannual for the Cache-Based Intelligent Database Interface of the Intelligent Database Interface*. Paoll, PA: Paramax Systems Organization. Rev. 2.3.

Quinlan, J.R. (1986). Induction of Decision Trees, *Machine Learning*, 1, 81–106.

Ramakrishnan, R., Srivastava, D., and Sudarshan, S. (1992). CORAL: Control, Relations, and Logic. In *Proceedings of the International Conference of the Internation Conference on Very Large Databases*.

Ramanathan, C. (1994). Providing Object, Oriented Access to a Relational Database. In D. Cordes and S. Vrbsky (Eds.), *Proceedings of the 32nd Annual Southeast Conference*, Held in Tuscaloosa, Alabama, New York.

Stroulia, E. and Goel, A.K. (1995). Functional Representation and Reasoning in Reflective Systems, *Journal of Applied Intelligence*, 9(1). Special Issue on Functional Reasoning.

Stroulia, E., Shankar, M., Goel, A.K., and Penberthy, L. (1992). A Model-Based Approach to Blame Assignment in Design. In J.S. Gero (Ed.), *Proceedings of the Second International Conference on Artificial Intelligence in Design*.

Stroulia, E., Thomson, J., and Situe, Q. (2000). Constructing XML-Speaking Wrappers for WEB Applications: Towards an Interoperating WEB. In *Proceedings of the 7th Working Conference on Reverse Engineering—WCRE-2000*, Brisbane, Queensland, Australia.

Tong, C. (1991). The Nature and Significance of Knowledge Compilation, *IEEE Expert*, 6(2).

Wiederhold, G. (1992). Mediators in the Architecture of Future Information Systems, *IEEE Computer*, 25, 38–49.

Wiederhold, G. and Genesereth, M.R. (1997). The Conceptual Basis for Mediation Services, *IEEE Expert*, 12(5), 138–155.