From Data to Knowledge: Method-Specific Transformations *

Michael J. Donahoo, J. William Murdock, Ashok K. Goel, Shamkant Navathe, and Edward Omiecinski

Georgia Institute of Technology
College of Computing
801 Atlantic Drive
Atlanta, Georgia 30332-0280, USA

Source: Proceedings of the 1997 International Symposium Symposium on Methodologies for Intelligent Systems.

Abstract. Generality and scale are important but difficult issues in knowledge engineering. At the root of the difficulty lie two hard questions: how to accumulate huge volumes of knowledge, and how to support heterogeneous knowledge and processing? One answer to the first question 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 answer. HIPED converts the second question 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 answer to this reformed question is to use method-specific transformation of data into knowledge. We describe an experiment in which a legacy knowledge system called INTERACTIVE KRITIK is integrated with an ORACLE database using IDI as the communication tool. The experiment indicates the computational feasibility of method-specific data-to-knowledge transformations.

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 both domain-specific in that their knowledge is relevant only to a limited class of domains, and task-specific in that their processing is appropriate only to a limited class of tasks.

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

* 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. We appreciate the support.

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 [?] 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. 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.

[?] 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 transformation* (or MST), 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 MST 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 [?, ?] with an external database represented in Oracle [?]. The knowledge system and the database communicate through IDI [?].

2 HIPED Architecture

Figure 1 illustrates the general scheme. We describe this architecture in the following subsection by decomposing it into database, knowledge system, and user components.

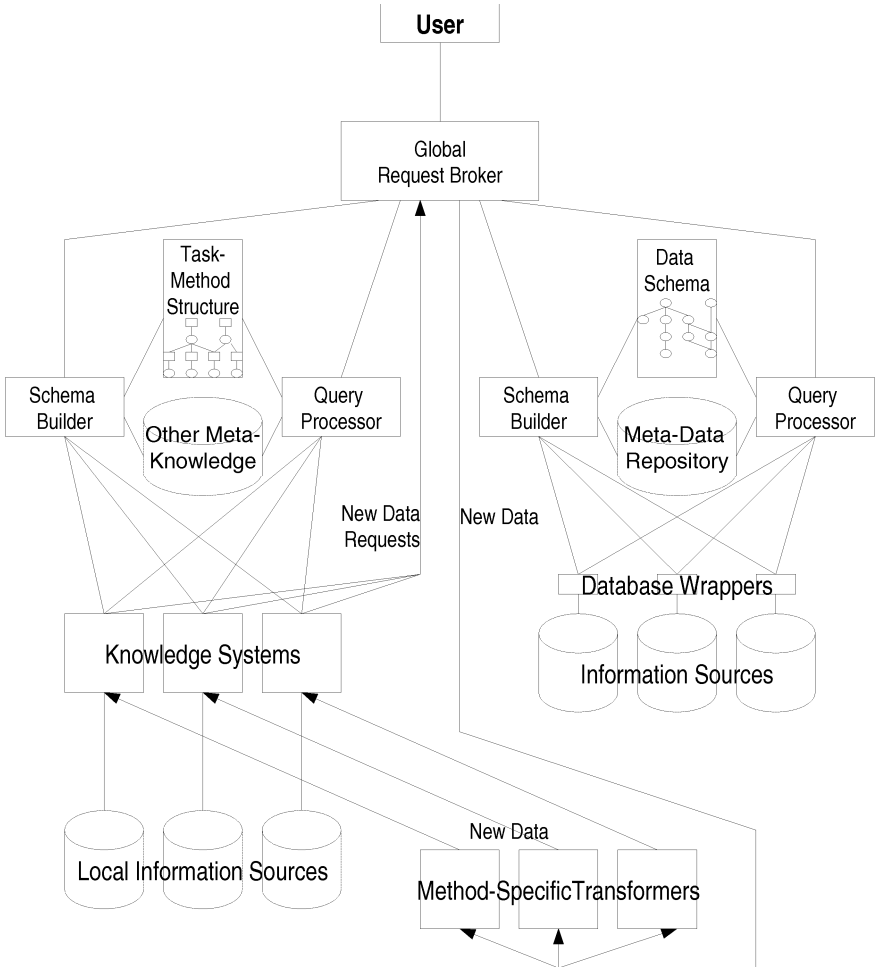

Fig. 1. The HIPED architecture (Arrowed lines indicate unidirectional flow of information; all other lines indicate bidirectional flow. Annotations on lines describe the nature of the information which flows through that line. Rectangular boxes indicate functional units and cylinders represent collections of data).

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. 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 [?]. 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[2] 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 [?] 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 queries on the local databases. Queries on the local databases can be expressed in a single query language which is coerced to the local databases query language by a database wrapper.

2.2 Knowledge System Integration

As with databases, a considerable number knowledge systems exist. Most knowledge systems do not provide an externally accessible description of the tasks and methods they address. We propose (see the left side of Figure 1) to allow knowledge system designers to develop a description, called a "task-method schema," of the tasks each local knowledge system can perform [?]. In this approach, a set of knowledge systems, defined at the level of tasks and methods, is 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.

2.3 Integrated Access

Transparent access to data and knowledge is important. 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. Knowledge systems needing data not available in their local repositories may act as users themselves.

2.4 Method-Specific Transformation

In this paper, we are concerned with the transformation of knowledge from external sources into a form suitable for use by a knowledge system method. A naive approach

[2] A mechanism for complete, automated integration is unlikely.

involves writing a transformation function for every permutation of knowledge system and database. Clearly, this limits the overall scalability of the system.

We propose to leverage the partially integrated global representation of the knowledge and database systems by creating a method-specific transformation for each knowledge system which transforms knowledge from the partially integrated global schema into a knowledge system specific representation. The number of necessary method-specific transformations is linear with respect to the number of knowledge systems, increasing the scalability of our approach.

2.5 Information Flow

Consider a 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 to the Global Request Broker. It then passes the output from both the knowledge and database system query processors through a method-specific transformer which coerces the data into a form which is usable by the requesting knowledge system. The resulting battery may be an actual 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 An Experiment with HIPED

We have been conducting a series of experiments in the form of actual system implementations. Figure 2 presents an architectural view of one such experiment, in which a legacy knowledge system 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 transformation.

3.1 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 transformer*.

Step 4 The *method-specific transformer* converts the data into a knowledge representation format which can by understood by the *knowledge system*.

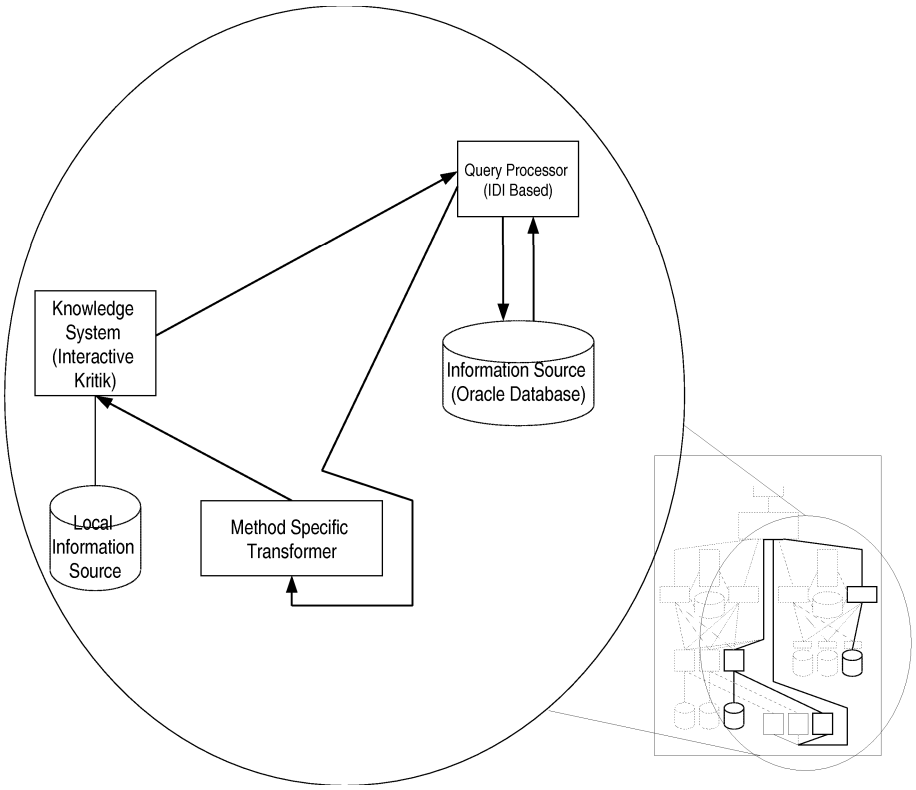

Fig. 2. The portion of the architecture relating to the proposed solution

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 problem of databases: given a query produce a data item. Lastly, step four is 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 transformation. The algorithm for the method-specific transformer 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.

3.2 Integration

The particular legacy systems which we combined in our implementation were Interactive Kritik and a relational database system [?] developed under Oracle. Interactive Kritik is a knowledge system which performs conceptual design of simple

physical devices and provides visual explanations of both the reasoning processes it goes through and the design products it produces. It is an inherently multi-strategy knowledge system. It uses case-based reasoning as a general process for performing design and it also uses an assortment of model-based methods for doing specific design tasks such as diagnosis and repair.

The experimental system which we have written serves as an interface between INTERACTIVE KRITIK and our Oracle database. It is used when INTERACTIVE KRITIK is attempting to redesign a device by component substitution, one redesign strategy in its library of strategies. As a simple example, consider the situation in which the system has determined that a flashlight is not producing enough light and has decided that a more powerful bulb is needed. When the system identifies a single component whose replacement could potentially solve the design problem, it consults its library of components to see if such a replacement exists; in the example, it would check to see if it knows about a more powerful bulb and would make a substitution only if it did. In earlier implementations, the library of components was stored entirely within INTERACTIVE KRITIK itself in the form of data structures in memory. In our experiment, these data structures are not present in memory and the request for an appropriate component takes place through our partial implementation of the pieces of the HIPED architecture illustrated in Figure 2.

INTERACTIVE KRITIK sends its request 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 value, 18 lumens. 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.

Table 1. The tables for the Oracle database

Table INSTANCE		
NAME	ATTRIBUTE	ATT_VAL
littlebulb	lumens	capacity-less
bigmotor	watts	power-more
bigbulb	lumens	capacity-more

Table PROTO_INST	
INST_NAME	PROTO_NAME
littlebulb	l-bulb
bigmotor	motor
bigbulb	l-bulb

If Oracle finds a result, as it does in this example, it returns it via the method-specific transformer. In this case, the query generates the string "bigbulb" as the result. The 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 transformer converts the raw data from the database to a form comprehensible to INTERACTIVE KRITIK by using the algorithm described in Section 3.1. 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 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 which is now able to continue with its processing.

4 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.

The specific experiment described in Section 3 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 [**?**]. 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 [?].

While the experiment described in this paper demonstrates method-specific transformation 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 transformation of data into useful knowledge.

This article was processed using the LaTeX macro package with LLNCS style