

## Database Connectivity Using an Agent-Based Mediator System

Larry M. Stephens and Michael N. Huhns  
Center for Information Technology  
Department of Electrical and Computer Engineering  
University of South Carolina  
Columbia, SC 29208  
[stephens@engr.sc.edu](mailto:stephens@engr.sc.edu), [huhns@engr.sc.edu](mailto:huhns@engr.sc.edu)  
Paper No. A38

**Keywords:** Agent-based information resource mediators, ontology agents for semantic interoperability.

**Category:** Regular research paper.

### Abstract

This paper describes a technique for creating mediators—information system components that operate among distributed applications and distributed databases. The databases are distributed logically and physically, typically residing on different platforms and having different semantics. A mediator automates the process of merging widely differing databases into a seamless, unified whole to be presented to the user. Our system's underlying strength is its homogeneity—all components are modeled as Java agents that communicate with each other in a common protocol (KQML) and with a common semantics (the ontology).

The mediator-based information system consists of a mediator-reasoning core, a universal ontology, wrappers to align the databases semantically with the ontology, and agents that handle connectivity issues. Each component is modeled as an agent and implemented as a JATLite process. The mediator agent operates as a forward-chaining, rule-based system written declaratively in Jess, the Java Expert System Shell. It is multithreaded to support simultaneous queries and interactions. The mediator's intelligence is embedded in a fact and rule base residing in a text file that is easily edited. By changing only the rule base, the mediator may be customized for any application.

# 1 Overview

Mediators are information system components that operate between distributed applications and distributed databases [Wiederhold]. The databases may be both logically and physically distributed, and they typically reside on different platforms and have different semantics. Although several of the databases might contain information for some of the same entities, the table and attribute names might not be common among the databases.

A mediator automates the process of merging widely differing databases into a seamless, unified whole to be presented to the user. Mediators mitigate the heterogeneity and distribution of information sources, but are difficult to construct. The major problems involve semantics and communications. The structure of a system for querying and updating heterogeneous databases is shown in Figure 1. The system consists of a mediator-reasoning core, a universal ontology, wrappers to align the databases semantically with the ontology, and agents that handle the connectivity issues. Our system enables the semiautomated construction and launching of mediators. Although the resultant mediators may be used in any domain, our specific domain of concern in this project was that of DARPA's military logistics data.

The information flow in the system consists of the following. Database and ontology agents communicate their semantics to the mediator agent. A user can formulate database commands in terms of the common ontology; the user's agent sends these commands to the mediator agent. The mediator reasons about schemas and ontologies to determine relevant databases or other information resources. The mediator communicates to each information resource's "wrapper" agent, which maps from terms in the common ontology to the resource's schema. Database accesses are made via the Java Database Connectivity (JDBC) protocol. For database queries, results are returned to the user in terms of the ontology.

Agents can be easily implemented with a package of Java programs called Java Agent Template [JATLite], which allows users to create software agents that communicate via KQML [Finin, *et al.*] over the Internet. The mediator agent operates as a forward-chaining, rule-based system written declaratively in the Java Expert System Shell [Jess], which is itself implemented in Java. The current version of Jess extends the functionality of (but is compatible with) CLIPS [CLIPS]. Jess is important because it provides a reasoning engine to Java applications.

## 2 Background

### 2.1 *The problem*

Millions of dollars and much time are wasted during military deployments, inventory resupplies, and warehouse restocking through shipping errors and other miscommunication about supplies. Much of this waste could be eliminated if all parties in a chain of supply had efficient access to correct information. To meet this goal, problems must be overcome that range from low-level connectivity issues to high-level semantic issues. Much of the information that our government and commercial organizations maintain is in a diverse set of legacy databases.

Legacy databases typically have mutually incompatible syntactic conventions and, worse, widely disparate semantics. However, these databases contain a wealth of information and have active users. Also, migration to newer systems often has very high infrastructure and training costs. For these reasons, ways must be found to coherently access information from multiple such databases.

## **2.2 The solution**

We have investigated an approach based on cooperative information systems, which is applicable to this and other such problems. In a cooperative information system, each major component of the system is represented by an agent. Solving the semantic mapping problem is the key to accessing heterogeneous databases. The agents in a cooperative information system provide the appearance of homogeneity. They communicate with each other in a common protocol (KQML) and with common semantics (the ontology). The common ontology of concepts includes both entities and relationships among the entities.

A system user formulates database commands in terms of the common ontology. Each database is accessed through a “wrapper” that maps from terms in the common ontology to the database schema. A mediator takes the ontology and database schema information and maps the user command to the appropriate database wrappers. The database wrappers are agents that translate the command into the local database schema. The wrappers return results to the mediator in terms of the common ontology. For query commands, the mediator gathers the results and passes them to the user.

## **3 Software Architecture**

The mediator is based on a uniform interaction among the components of the information system, with each component modeled as an agent. There are separate agents for each user interface, the ontology, and each database wrapper. Agents communicate with each other using the KQML protocol. A KQML message includes identifiers for the sender, receiver, type of message, ontology, and message content.

Each agent is a JATLite process running as a Java thread. Stanford University’s JATLite (Java Agent Template) is freeware consisting of a set of Java packages that facilitate the development of agents. JATLite provides basic communication tools (a hub agent for

message routing) and the high-level communication constructs of KQML messages [Finin, *et al.*]. Essentially, the KQML protocol reduces ambiguity in communication by explicitly classifying a message into such categories as “ask” or “tell,” and identifying the sender, receiver, a message language, and message content.

For simplicity, all interagent communication occurs through a special agent called the router. The agent router allows Java applets to exchange messages with *any* registered agent on the Internet. (Netscape’s security restrictions prohibit a given agent from communicating with an agent not spawned on the same host.) The agent router allows any registered agent to send messages to any other registered agent by making a single socket connection to the agent router. Messages are forwarded without the sending agent having to know the receiving agent’s absolute address and making a separate socket connection as with the usual Agent Name Server (ANS) infrastructure. Like an e-mail server, the agent router buffers all messages so that they are not lost due to network transient problems. If individual agents go down or logout, they may return for their messages at a later time.

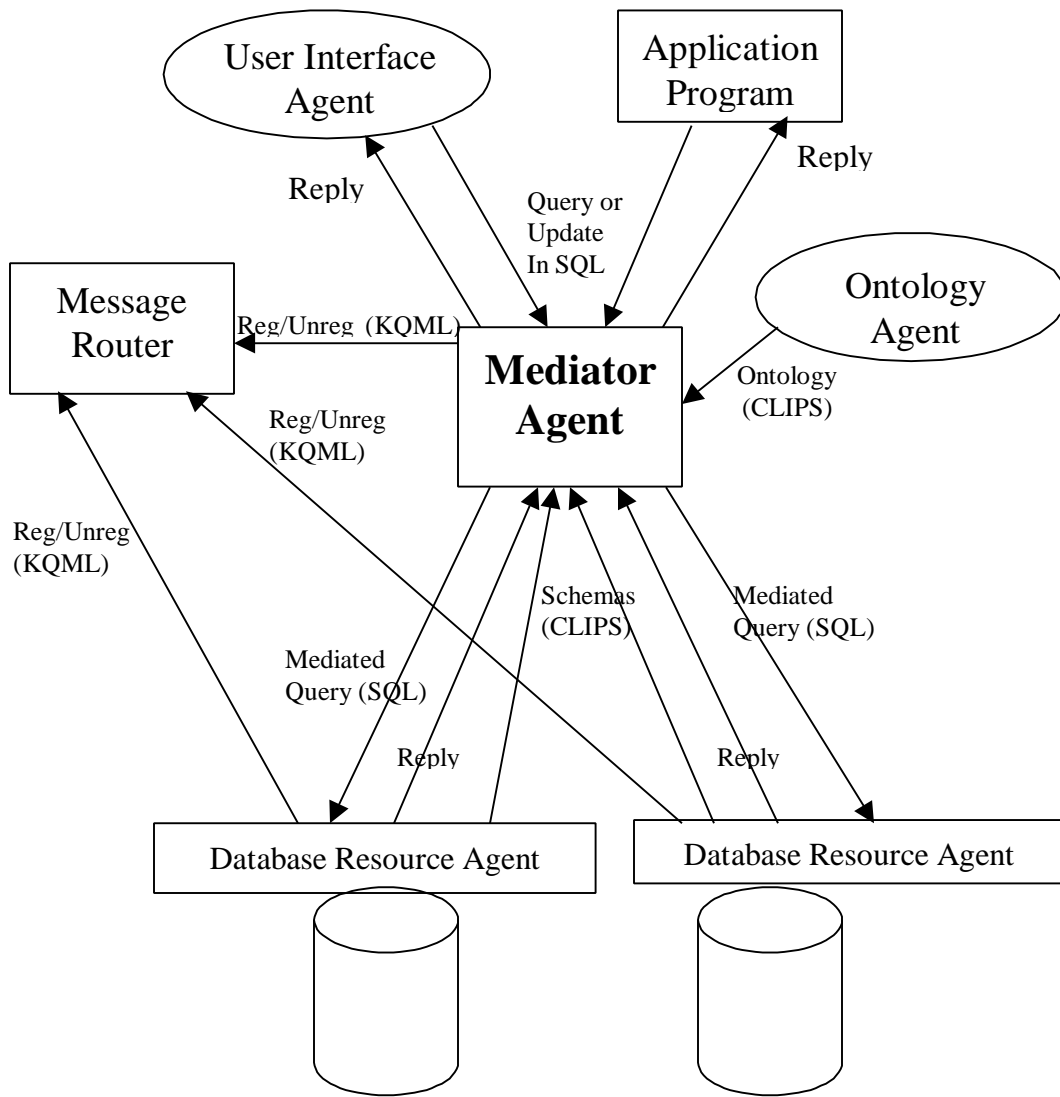
The mediator operates as a forward-chaining, rule-based system written declaratively in Jess, which is itself implemented in Java. Jess implements a version of the CLIPS system. The mediator, implemented as a JATLite agent, invokes Java code that implements the functionality of the CLIPS forward-chaining engine. The mediator’s logical behavior is determined by a fact and rule base residing in a text file. The behavior of the mediator can be changed by changing the rule base, without necessarily changing the JATLite agent or the Java code. Thus, a mediator may be specialized for a particular application by creating a new rule base.

In addition to its rules in Jess, the mediator contains additional Java classes that provide it with communication capabilities. These allow it to register and unregister with the router agent and exchange messages with other agents using KQML, and enable concurrent asynchronous operations on multiple databases. When resources come online, they register with a router agent. The ontology agent and the mediator register as well. The router agent keeps address information for each of the registered agents, and agents communicate with each other via the router.

At set-up time, the mediator receives the domain ontology information from the ontology agent. This information is sent to the mediator as a set of assertions in CLIPS syntax. This communication is accomplished using the KQML performatives “ask” and “tell.” Similarly, database schemas are received from the database resource agents at set-up time. These schemas are also represented as assertions in CLIPS and sent in KQML format.

The mediator contains rules that logically connect the input ontology and database schemas. When these rules fire, a mediator agent is instantiated for the given input of ontology and schemas. The resulting agent has a mapping between the domain ontology and the database schemas. This mapping is the basis for the query mediation.

At run time, an application program submits an SQL command to the interface agent, which forwards the command to the mediator agent. The mediator agent parses the query, reasons about which databases may have relevant information, reasons about any necessary decomposition of the query, and then sends the decomposed query to the resource agent for the relevant databases. These resource agents are JATLite agents. They accept the SQL commands and, using JDBC, connect to, open, and query the database.



## 4 System Implementation

The user-interface agent, mediator agent, ontology agent, and database wrappers are implemented as JATLite agents. Interagent communication in JATLite is via a router, which acts as an Agent Name Server (ANS). JATLite communication is based on KQML performatives.

The sequence of operations of the system is as follows:

1. On start-up, all agents connect to the router. The ontology agent and all database resource agents send their schemas to the mediator agent. All message passing is done through the router.
2. The mediator waits for a user agent to send a KQML message with the performative *ask-one* and the content field containing an SQL query (or update).
3. On receiving a query (or update) from a user agent, the mediator invokes Jess. Jess is executed on a file containing the rules that define the functionality of the mediator. The rules help the mediator determine which database resource agents to access and, based on the common ontology, the form of the SQL commands to be sent to the resource agents. The mediator translates the original SQL command to one or more commands for different database resource agents.
4. Each database resource agent receives a KQML message from the mediator with the content field containing the user agent's name and the SQL command for the database.
5. The database resource agents access their database using JDBC and send the results back to the mediator along with the user agent's name.
6. The mediator concatenates the results it receives from the various database resource agents and sends the combined result to the user agent that sent the original query.

### 4.1 The Ontology Agent

Ontology agents are essential for interoperability. (See [Huhns and Singh] for a discussion of ontologies and their uses in cooperative information systems and [JOE] for an example of a graphical tool for creating, browsing, and editing ontologies.) Ontologies

- Provide a common context as a semantic grounding, which agents can then use to relate their individual terminologies
- Provide (remote) access to multiple ontologies

- Manage the distributed evolution and growth of ontologies. A common context in the form of an ontology or model of the domain can provide such semantic grounding.

The following conventions in terminology are used in relating the user command (in SQL), the database, and the common ontology:

SQL Command	Concept	Attribute
<b>Database</b>	Table	Field
<b>Ontology</b>	Class	Slot

For example, to represent that Persons is a specialization of Mammals in the ontology, we assert

```
(subclassOf Persons Mammals)
```

We also permit slot specialization [Lenat and Guha] [Woods]:

```
(slot ID)
(slot StudentID)
(subslotOf StudentID ID)
```

Appendix 1 contains a simple ontology formatted as assertions of facts in CLIPS syntax.

## 4.2 Reasoning by Mediator Agents

All reasoning about concepts and relationships is done in terms of the common ontology [Dowell *et al.*]. Tables and fields in the databases are mapped to corresponding terms in the ontology. For convenience here, the mapping is made directly when building the rule base; however, a separate set of mappings from database terms to ontology terms can be used. (The database is assumed have tables that are not specialized with respect to each other, and the same assumption holds for database fields. Specialization reasoning is done only in terms of the ontology.) Thus, to indicate that database 1 has table *Students* and field *LastName*, we use

```
(hasTable db1 Students)
(hasField db1 LastName)
```

where *Students* and *LastName* are terms in the ontology.

Suppose that we have the following information from the ontology agent:

```
(subclassOf Students Persons)
(hasTable db1 Students)
(hasTable db2 Persons)
```

```
(subslotOf LastName Name)
```

and the following information from resource agents:

```
(hasField db1 LastName)
(hasField db2 Name)
```

Using our mediator, a query issued about Persons and Names would be mapped to separate queries about Persons and Names in database 1 and Students and LastNames in database 2. A query about the Names of Students would be mapped only to database 1, which has the specialized field LastName. Similarly, a query about the LastName of Persons would be mapped only to database 1, based on the specialization of Persons to Students.

The restrictions above illustrate that generalized queries are not permitted in our system; we guarantee sound results, which may be incomplete. For example, a query about Persons can be specialized to Students; the results, even if incomplete, would correctly represent Persons. However, a query about Students, if generalized to Persons, would give results that might not be relevant to Students.

To assist in reasoning about database tables and fields in terms of ontological concepts, the mediator has general knowledge about how concepts might relate to each other. For example, the mediator understands the transitivity of subclasses and subslots. The following rules encode this information:

```
(defrule class-transitivity
  (subclassOf ?x ?y)
  (subclassOf ?y ?z)
=>
  (assert (subclassOf ?x ?z)))
```

```
(defrule class-reflexivity
  (class ?x)
=>
  (assert (subclassOf ?x ?x)))
```

```
(defrule slot-transitivity
  (subslotOf ?x ?y)
  (subslotOf ?y ?z)
=>
  (assert (subslotOf ?x ?z)))
```

```
(defrule slot-reflexivity
  (slot ?x)
=>
  (assert (subslotOf ?x ?x)))
```

In general, mediators are specialized execution agents that

- Determine which resources might have relevant information using help from brokers
- Decompose queries to be handled by multiple agents
- Supervise query execution
- Operate as script-based agents to support scenario-based analyses
- Monitor and execute workflows, which might extend over the web and might be expressed in a format such as the one specified by the Workflow Management Coalition.
- Combine the partial responses obtained from multiple resources
- Translate between ontologies.

### **4.3 Information Resource Agents**

Information resource agents come in a variety of common types, depending on which resource they are representing, and provide the following capabilities:

- Wrappers implement common communication protocols and translate into and from local access languages. For example, a local data-manipulation language might be SQL for relational databases or OSQL for object-oriented databases.
- SQL database agents manage specific information resources
- Data analysis agents apply machine learning techniques to form logical concepts from data or use statistical techniques to perform data mining

Resource agents apply the mappings that relate each information resource to a common context to perform a translation of message semantics. At most  $n$  sets of mappings and  $n$  resource agents are needed for interoperation among  $n$  resources and applications, as opposed to  $n(n-1)$  mappings that would be needed for direct pairwise interactions among  $n$  resources without agents.

### **4.4 User Agents**

User agents have the following characteristics:

- Contain mechanisms to select an ontology
- Support a variety of interchangeable user interfaces, such as query forms, graphical query tools, menu-driven query builders, and query languages
- Support a variety of interchangeable result browsers and visualization tools
- Maintain models of other agents
- Provide access to other information resources, such as data analysis tools, workflows, and concept learning tools.

## **5 Conclusion**

Diversity is healthy, and a set of diverse agents can lead to more robust systems by enabling a variety of viewpoints to be represented and exploited. However, agents are typically complex pieces of software, so would not a set of different agents unnecessarily add to the complexity of a system? Yes, but the complexity can be mitigated by constructing the agents out of a common infrastructural toolkit, making them different only at the knowledge level, and limiting the types of agents to the standard ones.

It is a sign of the maturity of agent technology that common types of agents are emerging [Nodine]. It is reminiscent of the later years of research on expert systems, when a variety of reasoning methods and task types were identified by Chandrasekharan, Brown, Sticklen, and Clancy. The results were useful not only for characterizing research in expert systems, but also in constructing applications of expert systems.

For cooperative information systems, the resultant architectures based on standard agent types are much easier to develop, understand, and use. Perhaps most important of all, they will make it easier for separately developed information systems to interoperate.

Agents of specialized types will serve as standardized building blocks of future information systems. There are two trends we have observed that lend credence to such a prediction. First, software systems in general are being constructed with larger components, such as those based on ActiveX, Java Beans, and (soon) Jini, which are on the path to being agent architectures themselves. Second, cooperative information system architectures have been evolving a set of standard types of agents, which we have described herein. Essentially, each of these trends is raising the abstraction level at which software systems are being designed, described, and deployed.

## Appendix 1: Simple Ontology

```
; Define Ontology

(deffacts ontology

; class hierarchy
(class Things)
(class Mammals)
(class Persons)
(class Students)
(subclassOf Mammals Things)
(subclassOf Persons Mammals)
(subclassOf Students Persons)

; slot hierarchy
(slot Attribute)
(slot Name)
(slot LastName)
(slot ID)
(slot StudentID)
(subslotOf Name Attribute)
(subslotOf LastName Name)
(subslotOf ID Attribute)
(subslotOf StudentID ID)
)
```

## Appendix 2: Simple Database Schemas

```
:: Define schemas for databases 1 and 2

(deffacts schemas
(hasTable db1 Students)
(hasField db1 StudentID)
(hasField db1 LastName)
(hasTable db2 Persons)
(hasField db2 ID)
(hasField db2 Name)
)
```

## References

- [Dowell *et al.*] Michael L. Dowell, Larry M. Stephens, and Ronald D. Bonnell, "Using a Domain-Knowledge Ontology as a Semantic Gateway among Information Resources," in *Readings in Agents*, [M. N. Huhns and M. P. Singh, eds.], pp. 255-260, Morgan Kaufmann Publishers, Inc., 1998.
- [Finin *et al.*] Tim Finin, Richard Fritzon, Don McKay, and Robin McEntire, "KQML as an Agent Communication Language," *The Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*, ACM Press, November 1994.
- [Huhns and Singh] Michael N. Huhns and Munindar P. Singh, "Ontologies for Agents," *IEEE Internet Computing*, vol. 1, no. 6, pp. 81-83, November-December 1997.
- [JATLite] JATLite, the Java Agent Template: <http://cdr.stanford.edu/ABE/JavaAgent.html>
- [Jess] Jess, the Java Expert System Shell: <http://herzberg.ca.sandia.gov/jess/>
- [JOE] Java Ontology Editor: <http://www.ece.sc.edu/Labs/HIIT/html/imts-new.html/>
- [Lenat and Guha] Doug Lenat and R. V. Guha, *Building Large Knowledge-Bases Systems: Representation and Reasoning in the Cyc Project*, Addison-Wesley Publishing Company, Inc., 1999.
- [CLIPS] NASA, Lyndon B. Johnson Space Center, *CLIPS User's Guide, and CLIPS Reference Manual (Volumes 1: Basic Programming Guide)*, Version 6.05, 1997.
- [Nodine] Marian H. Nodine, "The InfoSleuth Agent System," in *Proceedings 2<sup>nd</sup> International CIA-98 Workshop: Learning, Mobility, and Electronic Commerce for Information Discovery in the Internet*, Paris, July 1998.
- [Wiederhold] Gio Wiederhold, "'Mediators in the architecture of future information systems," *IEEE Computer*, vol. 25, no. 3, 1992.
- [Woods] W. A. Woods, "Understanding Subsumption and Taxonomy," in *Principles of Semantic Networks: Exploration in the Representation of Knowledge*, (J. F. Sowa, ed.), Morgan Kaufmann Publishers, Inc., 1991.