

InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments

R. J. Bayardo Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk

Microelectronics and Computer Technology Corporation (MCC)
3500 West Balcones Center Drive
Austin, Texas 78759

<http://www.mcc.com/projects/infosleuth>
sleuth@mcc.com

Abstract

The goal of the InfoSleuth project at MCC is to exploit and synthesize new technologies into a unified system that retrieves and processes information in an ever-changing network of information sources. InfoSleuth has its roots in the Carnot project at MCC, which specialized in integrating heterogeneous information bases. However, recent emerging technologies such as internetworking and the World Wide Web have significantly expanded the types, availability, and volume of data available to an information management system. Furthermore, in these new environments, there is no formal control over the registration of new information sources, and applications tend to be developed without complete knowledge of the resources that will be available when they are run. Federated database projects such as Carnot that do static data integration do not scale up and do not cope well with this ever-changing environment. On the other hand, recent Web technologies, based on keyword search engines, are scalable but, unlike federated databases, are incapable of accessing information based on concepts. In this experience paper, we describe the architecture, design, and implementation of a working version of InfoSleuth. We show how InfoSleuth integrates new technological developments such as agent technology, domain ontologies, brokerage, and internet computing, in support of mediated interoperation of data and services in a dynamic and open environment. We demonstrate the use of information brokering and domain ontologies as key elements for scalability.

1 Introduction

Database research in the past has been focused on the relatively static environments of centralized and distributed enterprise databases. In these environments, information is centrally managed and the structure of data is consistent. Typically, the binding of concepts to specific sets of data

Permission to make digital/hard copy of part or all this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD '97 AZ, USA

© 1997 ACM 0-89791-911-4/97/0005...\$3.50

is known at the time a schema is defined and data access performance can be optimized using pre-computed indices.

The World Wide Web presents us with a different challenge. Here, there is more information and the information is spread over a vast geographic area. There is no centralized management of the information since anyone can publish information on the Web. Thus, there is minimal structure to the data and the structure may bear little relationship to the semantics. Therefore, there can be no static mapping of concepts to structured data sets, and querying is reduced to search engines that dynamically locate relevant information based on keywords.

The InfoSleuth project at MCC [19, 35, 37] is broadening the focus of database research to meet the challenge presented by the World Wide Web. A broadening of focus requires a re-thinking of fundamental requirements, a deep understanding of existing database technology, and a pragmatic approach to merging key technologies from database research and research from other computer disciplines. The InfoSleuth Project is developing technologies that operate on heterogeneous information sources in an open, dynamic environment. InfoSleuth views an information source at the level of its relevant semantic concepts, thus preserving the autonomy of its data. Information requests to InfoSleuth are specified generically, independent of the structure, location, or even existence of the requested information. InfoSleuth filters these requests, specified at the semantic level, flexibly matching them to the information resources that are relevant at the time the request is processed.

InfoSleuth is based on MCC's previously developed Carnot technology [7, 18, 36], which was successfully used to integrate heterogeneous information resources. The Carnot project developed semantic modeling techniques that enabled the integration of static information resources and pioneered the use of agents to provide interoperation among autonomous systems. Carnot, however, was not designed to operate in a dynamic environment where information sources change over time, and where new information sources can be added autonomously and without formal control.

The InfoSleuth project extends the capabilities of the Carnot technologies into dynamically changing environments, where the identities of the resources to be used may be unknown at the time the application is developed. InfoSleuth, therefore, rigidly observes the autonomy of its resources, and

does not depend on their presence. Information-gathering tasks are thus defined generically, and their results are sensitive to the available resources. InfoSleuth must consequently provide flexible, extensible means to locate information during task execution, and must deal with incomplete information and partial results.

To achieve this flexibility and openness, InfoSleuth integrates the following new technological developments in supporting mediated interoperation of data and services over information networks:

1. *Agent Technology.* Specialized agents that represent the users, the information resources, and the system itself cooperate to address the information processing requirements of the users, allowing for easy, dynamic reconfiguration of system capabilities. For instance, adding a new information source merely implies adding a new agent and advertising its capabilities. The use of agent technology provides a high degree of decentralization of capabilities which is the key to system scalability and extensibility.
2. *Domain models (ontologies).* Ontologies give a concise, uniform, and declarative description of semantic information, independent of the underlying syntactic representation or the conceptual models of information bases. Domain models widen the accessibility of information by allowing the use of multiple ontologies belonging to diverse user groups.
3. *Information Brokerage.* Specialized broker agents semantically match information needs (specified in terms of some ontology) with currently available resources, so retrieval and update requests can be routed only to the relevant resources.
4. *Internet Computing.* Java and Java Applets are used extensively to provide users and administrators with system-independent user interfaces, and to enable ubiquitous agents that can be deployed at any source of information regardless of its location or platform.

In this paper, we present our working prototype version of InfoSleuth, which integrates the aforementioned technologies with more classic approaches to querying (SQL) and schema mapping. We also describe a utilization of InfoSleuth in the domain of health care applications.

This paper is organized as follows. The overall architecture is described in section 2. Detailed descriptions of the agents are given in section 3. Section 4 describes the InfoSleuth and the domain ontology design. Brokering and constrained information matching is described in section 5. A data mining application in the health care domain is briefly presented in section 6. Related work is discussed in section 7. Finally, section 8 gives the conclusion and future work.

2 Architecture

2.1 Architectural Overview

InfoSleuth is comprised of a network of cooperating agents communicating by means of the high-level agent query language KQML [11]. Users specify requests and queries over specified ontologies via applet-based user interfaces. The dialects of the knowledge representation language KIF [13] and the database query language SQL are used internally to represent queries over specified ontologies. The queries

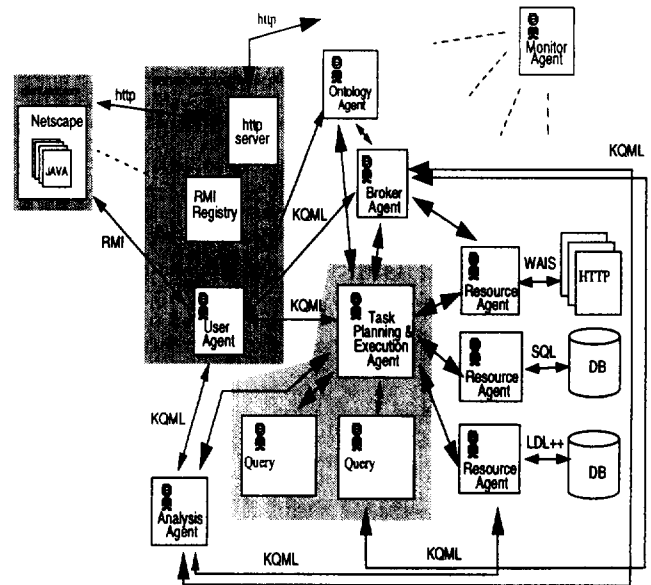


Figure 1: The InfoSleuth architecture

are routed by mediation and brokerage agents to specialized agents for data retrieval from distributed resources, and for integration and analysis of results. Users interact with this network of agents via applets running under a Java-capable Web browser that communicates with a personalized intelligent User Agent.

Agents advertise their services and process requests either by making inferences based on local knowledge, by routing the request to a more appropriate agent, or by decomposing the request into a collection of sub-requests and then routing these requests to the appropriate agents and integrating the results. Decisions about routing of requests are based on the "InfoSleuth" ontology, a body of metadata that describes agents' knowledge and their relationships with one another. Decisions about decomposition of queries are based on a domain ontology, chosen by the user, that describes the knowledge about the relationships of the data stored by resources that subscribe to the ontology.

Construction of ontologies for use by InfoSleuth is accomplished most easily by the use of the Integrated Management Tool Suite (IMTS, not discussed in this paper), which provides a set of graphic user interfaces for that purpose.

Figure 1 shows the overall architecture of InfoSleuth, in terms of its agents. The functionalities of each of the agents are briefly described below. Detailed descriptions are given in the following section.

User Agent: constitutes the user's intelligent gateway into InfoSleuth. It uses knowledge of the system's common domain models (ontologies) to assist the user in formulating queries and in displaying their results.

Ontology Agent: provides an overall knowledge of ontologies and answers queries about ontologies.

Broker Agent: receives and stores advertisements from all InfoSleuth agents on their capabilities. Based on this information, it responds to queries from agents as to where to route their specific requests.

Resource Agent: provides a mapping from the common ontology to the database schema and language native to its resource, and executes the requests specific to that resource, including continuous queries and notifications. It also advertises the resources' capabilities.

Data Analysis Agent: corresponds to resource agents specialized for data analysis/mining methods.

Task Execution Agent: coordinates the execution of high-level information-gathering subtasks (scenarios) necessary to fulfill the queries. It uses information supplied by the Broker Agent to identify the resources that have the requested information, routes requests to the appropriate Resource Agents, and reassembles the results.

Monitor Agent: tracks the agent interactions and the task execution steps. It also provides a visual interface to display the execution.

2.2 Agent Communication Languages

KQML [11] is a specification of a message format and protocol for semantic knowledge-sharing between cooperative agents. Agents communicate via a standard set of KQML performatives, which specify a set of permissible actions that can be performed on the recipient agent, including basic query performatives ("evaluate," "ask-one," "ask-all"), informational performatives ("tell," "untell"), and capability-definition performatives ("advertise," "subscribe," "monitor"). Since KQML is not tied to any one representation language, it can be used as a "shell" to contain messages in various languages and knowledge representation formats, and permit routing by agents which do not necessarily understand the syntax or semantics of the content message.

The Knowledge Interchange Format, KIF [13], provides a common communication mechanism for the interchange of knowledge between widely disparate programs with differing internal knowledge representation schemes. It is human-readable, with declarative semantics. It can express first-order logic sentences, with some second-order capabilities. Several translators exist that convert existing knowledge representation languages to and from KIF. InfoSleuth agents currently share data via KIF. Typically, an agent converts queries or data from its internal format into KIF, then wraps the KIF message in a KQML performative before sending to the recipient agent.

Both languages have been extended to provide additional functionalities required by the design of InfoSleuth.

2.3 Agent Interactions

In the following, we demonstrate a scenario of interaction among the InfoSleuth agents in the context of a simple query execution:

During system start-up, the Broker Agent initializes its InfoSleuth ontology, and commences listening for queries and advertisement information at a well-known address. Each Agent advertises its address and function to the Broker Agent using the InfoSleuth ontology.

When a Resource Agent initializes, it sets up its connection to its resource and advertises the components of ontology(ies) that it understands to the Broker Agent. One specialized Resource Agent, the Ontology Agent, deals with the information system's metadata.

A user commences interaction with InfoSleuth by means of a Web Browser or other Java applet viewer interacting with her personal User Agent. The user poses a query by means of the viewer applet. At this point, the User Agent queries the Broker Agent for the location of an applicable Execution Agent. The User Agent then issues the query to that Execution Agent.

On receiving a request, the Execution Agent then queries the Broker Agent for the location of the Ontology Agent (if it does not already know it), and queries the Ontology Agent for the ontology appropriate to the given query. Based on the ontology for the domain of the query, the Execution Agent queries the Broker Agent for currently appropriate Resource Agents. The Broker Agent may return a different set of Resource Agents if the same query is posted at a different time, depending on the availability of the resources.

The Execution Agent takes the set of appropriate Resource Agents, decomposes the query, and routes it appropriately. Each Resource Agent translates the query from the query domain's global ontology into the resource-specific schema, fetches the results from the resource, and returns them to the Execution Agent. The Execution Agent reassembles the results and returns them to the User Agent, which then returns the results to the user's Viewer applet for display.

The above scenario of a simple query execution is chosen for brevity. Other common scenarios of interactions in InfoSleuth would reflect complex queries with multiple-task plans and data mining queries that require knowledge discovery tasks.

3 Agent Design and Implementation

In this section, we describe the functionality, design rationale, and implementation of each of the InfoSleuth agents.

3.1 User Agent

The User Agent is the user's intelligent interface to the InfoSleuth network. It assists the user in formulating queries over some common domain models, and in displaying the results of queries in a manner sensitive to the user's context.

Upon initialization, the User Agent advertises itself to the broker, so that other agents can find it based on its capabilities. It then obtains information from the ontology agent about the common ontological models known to the system. It uses this information to prompt its user in selecting an ontology in which a set of queries will be formulated.

After a query is formulated in terms of the selected common ontology, it is sent to the task execution agent that best meets the user's needs with respect to the current query context. When the task execution agent has obtained a result, it engages in a KQML "conversation" with the user agent, in which the results are incrementally returned and displayed. The User Agent is persistent and autonomous; storing information (data and queries) for the user, and maintaining the user's context between browser sessions.

Implementation. The User Agent is implemented as a stand-alone Java application. As with the other agents in the architecture, explicit thread management is used to support concurrent KQML interactions with other agents, so that the User Agent does not suspend its activity while waiting for the result of one query to be returned. Currently, the

agents query the task execution agents using KQML with SQL content.

A user interface is provided via Java applets for query formulation, ontology manipulation, and data display, which communicate with the User Agent by means of Java's Remote Method Invocation (RMI). The applets provide a flexible, platform-independent, and context-sensitive user interface, where query formulation can be based on knowledge of the concepts in the relevant common ontology, the user's profile, and/or application-specific knowledge. Various sets of applets may be invoked based on these different contexts. The User Agent is capable of saving the queries created via applets, as well as results of queries. As the complexity of the InfoSleuth knowledge domain grows, this set of applets may eventually be maintained as reusable modules in a warehouse separate from the User Agent.

3.2 Task Execution Agent

The Task Execution Agent coordinates the execution of high-level information gathering tasks. We use the term "high-level" to suggest workflow-like or data mining and analysis activities. Such high-level tasks can potentially include global query decomposition and post-processing as sub-tasks carried out by decomposition sub-agents, where the global query is couched in terms of a common ontology; and sub-queries must be generated based on the schemas and capabilities of the various resources known to the system, and then the results joined.

The Execution Agent is designed to deal with dynamic, incomplete and uncertain knowledge. We were motivated in our design by the need to support flexibility and extensibility in dynamic environments. This means that task execution, including interaction with users via the user agents, should be sensitive both to the query context and the currently available information.

The approach we have taken for the Task Execution Agent is based on the use of *declarative* task plans, with *asynchronous* execution of procedural attachments. Plan execution is *data-driven*, and supports flexibility in reacting to unexpected events and handling incomplete information.

The declarative specification of the agent's plan and sub-task knowledge supports task plan maintenance, as well as the opportunity for collaborative task execution via the exchange of plan fragments. This declarative specification resides in the agent's knowledge base, and consists of several components, including: (1) Domain-independent information about how to execute task plan structures; (2) knowledge of when it is acceptable to invoke a task operator (including its preconditions) and how to instantiate it; (3) knowledge of how to execute the operator; (4) a Task Plan library; and (5) agent state.

The task plans are declarative structures, which can express partial-orders of plan nodes, as well as simple execution loops. Plans are currently indexed using information about the domain of the query and the KQML "conversational" context for which the task has been invoked.

Task Plan Execution Using Domain-independent Rules.

After an agent's knowledge base has been populated with operator descriptions and declarative task plans, it uses its domain-independent task execution knowledge to carry out the plans. Its knowledge, in the form of rules, supports the following functionality:

- Multiple plans and/or multiple instantiations of the same plan may concurrently execute.

- For a given node in a plan, multiple instantiations of the node may be created.
- Task execution is *data-driven*: a plan node is not executed until its preconditions are met [34].
- Execution of a plan node can be overridden by rules for unusual situations.
- Reactive selection of operations not in the current explicit plan can occur based on domain heuristics.
- Information-gathering operators [21, 8], and conditional operator execution are supported.

Each time a query from the user agent is received, a new instantiation of the appropriate plan from the plan library is initialized by the rule-based system. A task execution agent can concurrently carry out multiple instantiations of one or more plans, with potentially multiple instantiations of steps in each plan. The plan execution process is what defines the Task Execution Agent's behavior. The sequences of interactions with other agents are determined by the task plans the agent executes, and the conversations with a given agent are determined by the KQML protocols and supported primarily by the procedural attachments to the task operators. For example, a user agent can request that the results of the query be returned incrementally.

Example: General Query Task Plan. Executing a general query task plan causes the Task Execution Agent to carry out the following steps.

- Advertise to the Broker, using a *tell* performative, and wait to receive a reply (done at agent initialization).
- Wait to receive queries from User Agents. These will typically be encoded as KQML *directives*, such as *ask-all*, *standby*, or *subscribe*). The query as well as the domain context determines the task plan that is instantiated to process the query.
- Parse the query, and decompose it if appropriate¹. Parsing involves getting an ontological model from the Ontology Agent; once this model is obtained, it is cached for future use.
- Construct KIF queries based on the SQL queries' contents, and query the Broker using the KIF queries and the *ask-all* performative to find relevant resources.
- Query the relevant resource agents specified by the broker.
- Compose the results.
- Incrementally return the results to the user agent using a streaming protocol. Using this protocol, the user agent successively requests additional result tuples.

¹Only query union decomposition is performed at the task plan level. Previous work in the InfoSleuth project has focused on techniques for global query decomposition and post-processing. Work is currently in progress to port this functionality to the agent architecture while supporting the dynamic nature of resource availability, via decomposition agents invoked from the task level. See Section 8.3.

Implementation. The Task Execution Agent is implemented by embedding a CLIPS [32] agent in Java, using Java's "native method" facility. CLIPS provides the rule-based execution framework for the agent, and, as described above, the declarative specification of plan and operator knowledge. The Java wrapper supports procedural attachments for the plan operators, as well as providing the Java KQML communications packages used by all the agents in the InfoSleuth system. Thus, all communication with other agents takes place via procedural operator implementations.

A CLIPS/Java API has been defined to send information from CLIPS to the Java sub-task implementations, and for each plan operator (in CLIPS) that invokes a Java method, a new thread is created to carry out the sub-task, parameterized via this API. During sub-task execution, new information (in the form of CLIPS facts and objects) may be passed back to the CLIPS database, and this is how the Java sub-task methods communicate their results. The sub-task execution is asynchronous, and results may be returned at any time. Because the task execution is data-driven, new task steps will not be initiated until all the required information for those steps are available.

3.3 Broker Agent

The Broker Agent is a semantic "match-making" service that pairs agents seeking a particular service with agents that can perform that service. The Broker Agent, therefore, is responsible for the scalability of the system as the number and volume of its information resources grow. The Broker Agent determines the set of relevant resources that can perform the requested service. As agents come on line, they advertise their services to the broker via KQML. The Broker Agent responds to an agent's request for service with information about the other agents that have previously advertised relevant service. Details of the Broker protocols describing the exchanged information are given in section 5.2. In effect, the Broker Agent is a cache of metadata that optimizes access in the agent network. Any individual agent could perform exactly the same queries on an as-needed basis. In addition, the existence of the Broker Agent both reduces the individual agent's need for knowledge about the structure of the network and decreases the amount of network traffic required to accomplish an agent's task.

Minimally, an agent must advertise to the Broker its location, name, and the language it speaks. Additionally, agents may advertise meta-information and domain constraints based on which it makes sense to query a given agent. The purpose of domain advertising is to allow the Broker to reason about queries and to rule out those queries which are known to return null results. For example, if a Resource Agent advertises that it knows about only those medical procedures relating to heart surgery, it is inappropriate to query it regarding liver resection, and the Broker would not recommend it to an agent seeking liver resection data. The ontology used to express advertisements is called the "InfoSleuth" ontology because the metadata the Broker Agent is storing is a description of the relationships between agents.

Implementation. The Broker Agent is written in Java and the deductive database language LDL++ [38]. It supports queries from other agents using KQML for the communication layer and KIF for the semantic content (based on the "InfoSleuth ontology"). The constraint matching and data storage for the Broker Agent are implemented in

LDL++. The Broker translates the KIF statements into LDL++ queries and then sends them off to the LDL server to be processed. The use of the deductive database allows the broker to perform rule-based matching of advertisements to user requests.

3.4 Resource Agent

The purpose of the Resource Agent is to make information contained in an information source (e.g., database) available for retrieval and update. It acts as an interface between a local data source and other InfoSleuth agents, hiding specifics of the local data organization and representation.

To accomplish this task, a Resource Agent must be able to announce and update its presence, location, and the description of its contents to the broker agent. There are three types of contents information that are of potential interest to other agents: (1) metadata information, i.e., ontological names of all data objects known to the Resource Agent, (2) values (ranges) of chosen data objects, and (3) the set of operations allowed on the data. The operations range from a simple read/update to more complicated data analysis operations. The advertisement information can be sent by the Resource Agent to the broker at the start-up time or extracted from the Resource Agent during the query processing stage.

The Resource Agent also needs to answer queries. The Resource Agent has to translate queries expressed in a common query language (such as KQML/KIF) into a language understood by the underlying system. This translation is facilitated by a mapping between ontology concepts and terms and the local data concepts and terms, as well as between the common query language syntax, semantics and operators, and those of the native language. Once the queries are translated, the resource agent sends them to the information source for execution, and translates the answers back into the format understood by the requesting agent. Additionally, the resource agent and the underlying data source may group certain operations requested by other agents into an atomic (local) transaction. Also, the resource agent provides limited transactional capabilities for (global) multi-resource transactions.

The capability of a Resource Agent can be enhanced in many ways. For example, it may be able to keep the query context and thus allow for retrieval of results in small increments. Handling of event notifications (e.g., new data is inserted, an item is deleted) can be another important functionality of a Resource Agent.

The components of an example Resource Agent are presented in Figure 2. The communication module interacts with the other agents. The language processor translates a query expressed in terms of global ontology into a query expressed in terms of the Oracle database schema. It also translates the results of the query into a form understood by other agents. Mapping information necessary for this process is created during the agent installation time as it requires specialized knowledge of both the local data and the global ontology. The task of the event detection module is to monitor the data source for the events of interest and prepare the notifications to be sent to the agents interested in those events.

The InfoSleuth architecture has a specialized resource agent, called the ontology agent, which responds to the queries related to ontologies. It uses the same KQML message exchange as other agents, but unlike resource agents that are associated with the databases, it only interprets

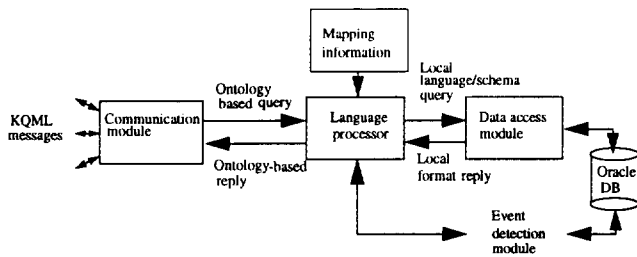


Figure 2: An example of a resource agent

KIF queries. This agent is designed to respond to queries concerning the available list of ontologies, the source of an ontology and searching the ontologies for concepts.

We are currently researching the possibility of adding inferring capability to respond to more sophisticated queries. There is also a need for maintaining different versions of the same ontology as the agent architecture is scaled up. These two capabilities become particularly relevant as the number of served ontologies increases, especially when multiple ontologies are integrated for more complex query formulation.

Implementation. The resource agent is written in Java and provides access to relational databases via JDBC and ODBC interfaces. We have run it successfully with Oracle and Microsoft Access and SQL Server databases. The functionality of the implemented agent covers advertisements about the data (both metadata information and ranges/values of the data contained in the database), and processing of queries expressed in either global ontology terms or local database schema terms. We implemented three types of query performances: ask-one, ask-all and standby, thus giving the other agents the option of retrieving one reply, all replies or all replies divided in smaller chunks.

4 Ontologies in the InfoSleuth architecture

The InfoSleuth architecture as discussed in the previous section is based on the communication among a community of agents, cooperating to help the user to find and retrieve the needed information. A critical issue in the communication among the agents is that of *ontological commitments*, i.e. agreement among the various agents on the terms for specifying agent context and the context of the information handled by the agents.

An ontology may be defined as the specification of a representational vocabulary for a shared domain of discourse which may include definitions of classes, relations, functions and other objects [15]. Ontologies in InfoSleuth are used to capture the database schema (e.g., relational, object-oriented, hierarchical), conceptual models (e.g., E-R models, Object Models, Business Process models) and aspects of the InfoSleuth agent architecture (e.g., agent configurations and workflow specifications). The motivations for using ontologies are two-fold:

1. *Capturing and reasoning about information content:* In an open and dynamic environment, the volume of data available is a critical problem affecting the scalability of the system. Ontologies may be used to:

- Determine the relevance of an information source without accessing the underlying data. This requires the ability to capture and reason with an

intensional declarative description of the information source contents. Object-oriented and relational DBMSs do not support the ability to reason about their schemas. Ontologies specified in a knowledge representation or logic programming language (e.g., LDL) can be used to reason about information content and hence enable determination of relevance.

- Capture new and different world views in an open environment as domain models. Wider accessibility of the data is obtained by having multiple ontologies describe data in the same information source.

2. *Specification of the agent infrastructure:* Ontologies are used to specify the context in which the various agents operate, i.e., the information manipulated by the various agents and the relationships between them. This enables decisions on which agents to route the various requests to. This information is represented in the InfoSleuth ontology and represents the world view of the system as seen by the broker agent. As the functionality of the various agents evolves, it can be easily incorporated into the ontology.

Thus, ontologies are used to specify both the infrastructure underlying the agent-based architecture and characterize the information content in the underlying data repositories.

4.1 A Three-layer Model for Representation and Storage of Ontologies

Rather than choose one universal ontology format, InfoSleuth allows multiple formats and representations, representing each ontology format with an ontology meta-model which makes it easier to integrate between different ontology types. We now discuss an enhancement of the 3-layer model for representation of ontologies presented in [20]. The three layers of the model (shown in Figure 3) are: Frame, Meta-model, and Ontology.

The Frame layer (consisting of Frame, Slot, and Meta-Model classes) allows creation, population, and querying of new meta-models. Meta-model layer objects are instances of frame layer objects, and simply require instantiating the frame layer classes. Ontology layer objects are instances of meta-model objects.

The objects in the InfoSleuth ontology are instantiations of the entity, attribute and relationship objects in the Meta-model layer. In our architecture, agents need to know about other entities, called "agents". Each "agent" has an attribute called "name" that is used to identify an agent during message interchange. The "type" of an agent is relevant for determining the class of messages it handles and its general functionality.

A key feature of the InfoSleuth ontology is that it is *self-describing*. As illustrated in Figure 3, the entity agent has ontologies associated with it. The entity ontology is an object in the meta-model layer and the various ontologies of the system are its instantiations. However in the case of the InfoSleuth ontology, the instantiation "InfoSleuth" of the ontology object is also a part of the InfoSleuth ontology. This is required as the InfoSleuth ontology is the ontology associated with the broker agent.

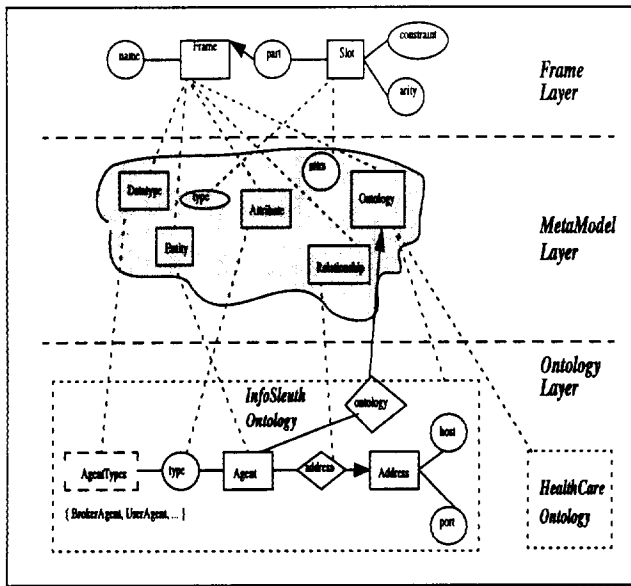


Figure 3: The three-layer ontology model

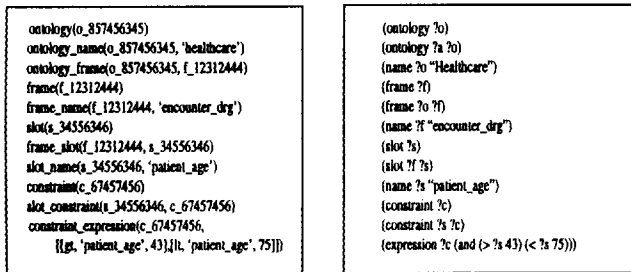


Figure 4: Multiple representation of same ontology

4.2 Utilization of Multiple Representations of Ontologies

One of the reasons for representing ontologies is the ability to reason about them. For this purpose, different agents might represent them in different languages depending on the type of inferences to be made. Figure 4 shows an example of the same piece of ontology represented by the resource agent in KIF and by the broker agent in LDL. The broker agent uses this representation to determine whether a resource agent is relevant for a particular query.

The Broker Agent utilizes a representation of the ontology exported by the Resource Agent (shown in Figure 4) in LDL [38]. The deductive mechanisms of LDL help determine the consistency of the constraints in the user query and those exported by the Resource Agent which in turn determines the relevance of the information managed by Resource Agent. The Resource Agent, on the other hand, translates this information into KIF expressions (as shown in Figure 4), and sends them to the Broker Agent.

5 Brokering in InfoSleuth

One of the valuable new features of the InfoSleuth technology is an intelligent brokering system that performs semantic as well as syntactic brokering of resources. Each agent in the system advertises its capabilities to the Broker Agent. The advertisements specify the agent's capabilities in terms

of one or more of the ontologies. From the user's perspective, semantic brokering enables requests to be specified in terms of the concepts in an ontology, and matches those semantic concepts to the resources that are currently best suited to handle those specific requests.

5.1 Capabilities Enabled by Semantic Brokering

Semantic brokering helps expand the functionality of InfoSleuth in the following ways.

Intelligent Routing. Through the use of brokering, InfoSleuth offers the ability to route information requests based on content, through the use of constraint matching on the ontology a resource claims expertise over. For instance, a resource may have access to information only about doctors in Houston and Austin. It would be fruitless to query this resource about doctors in Dallas and the use of constraints rules this resource out.

Currently constraint matching is an intersection function between the user query and the data resource constraints. If the conjunction of all the user constraints with all the resource constraints is satisfiable, then the resource contains data relevant to the user request. We should mention here that, following "the open world assumption", the Broker Agent always matches a query with unconstrained, yet relevant data sources, regardless of the constraints imposed by the query.

Note that the constraints for both the user request and the resource data profiles are specified in terms of some common ontology. It is the use of this common vocabulary that enables the dynamic matching of requests to applicable resources.

Dynamic Binding of Resources. An InfoSleuth broker accepts advertisements from new resources and notifications of resource unavailability at any time. Thus, InfoSleuth is able to keep up with an ever changing set of resources, which is not easily accomplished in a federated database. As resources come and go, the broker is made aware of this through KQML advertisements, and will thus only recommend appropriate resources to the agents doing the query planning. This means that the same user request may produce different results at different times, depending on which resources are available. Also, neither the user nor any agents acting on his behalf needs to know where or what resources are available when building a query plan, i.e. the user can query an open information space.

Scalability. There are several ways in which our approach to brokering impacts system scalability. First, decisions on which resources are likely to be relevant to specific user requests are made without actually accessing the resource. This greatly reduces the time and effort required to route a request. Secondly, the ease with which new resources may be added to the system makes scalability much less of an issue. To add a resource to the system it need only have a KQML/KIF interface for advertising its services; then other agents can make use of them immediately. Thirdly, as the number of agents in InfoSleuth grows, the different syntactic and semantic brokering functions can be factored out into separate agents.

5.2 Broker Protocols

The protocol for the Broker Agent currently supports two types of requests: advertisements and queries. Each is discussed in turn.

Advertisement. Advertisement is accomplished by means of the KQML performative *tell*. Modification of a service description is accomplished simply by issuing a new *tell* performative with appropriate fields altered to reflect the new state. Repudiation of a service is accomplished by issuing a *tell* performative with appropriate fields nulled out. Constraints on the agent's advertised ontology are expressed using KIF. The Broker currently accepts constraints on fields for single values, value ranges, and sets of values. A *tell* advertisement of a simple range constraint might look like that in Figure 5.

Broker Query. Queries to the Broker Agent about the InfoSleuth ontology are made by means of a KQML message that uses the *ask-one* and *ask-all* performatives and embeds a KIF query specifying constraints that should be met by the agents whose addresses are to be returned. Figure 5 illustrates an example of an *ask-all* query. The Broker reasons about these constraints by translating the KIF expressions into LDL++. Queries on constraints are currently restricted to expressions on open-ended ranges and set membership. Logical conjunction of constraints is currently possible; disjunction is not yet explicitly supported; however, an agent can achieve disjunction by issuing separate queries and concatenating the results.

A successful reply from the broker will return a list of tuples, each containing the name and address of the advertised agent, among other information. If the broker does not find an agent matching the request, the returned list will be empty. The reply to an *ask-one* query returns the first tuple found to meet the minimal requirements, and the reply to an *ask-all* query returns as many tuples as the broker finds which satisfies the query. In the future, we will be more flexible in allowing the querying agent to specify the type of information to be returned beyond a simple name and address in the aspect field of the KQML query. Likewise, the broker will also return the "best" match to an *ask-one*, and a ranked list of recommendations to an *ask-all*.

The query in Figure 5 represents an execution agent asking the Broker Agent for Resource Agents that understand SQL and have information about patients older than 65. The Broker Agent, based on the previous advertisement, would send a KQML reply performative indicating the address of the resource agent that matches the query.

6 Applications

In this section, we demonstrate the use of InfoSleuth in data mining applications, and we present a data mining example from the health care application domain.

6.1 Knowledge Discovery through InfoSleuth

Knowledge discovery in databases has recently received considerable attention due to the proliferation of large databases whose size prohibits effective analysis via traditional means. Fayyad, Piatetsky-Shapiro, and Smyth [12] describe the process of knowledge discovery as one involving five phases: data selection, data preprocessing, data transformation, data

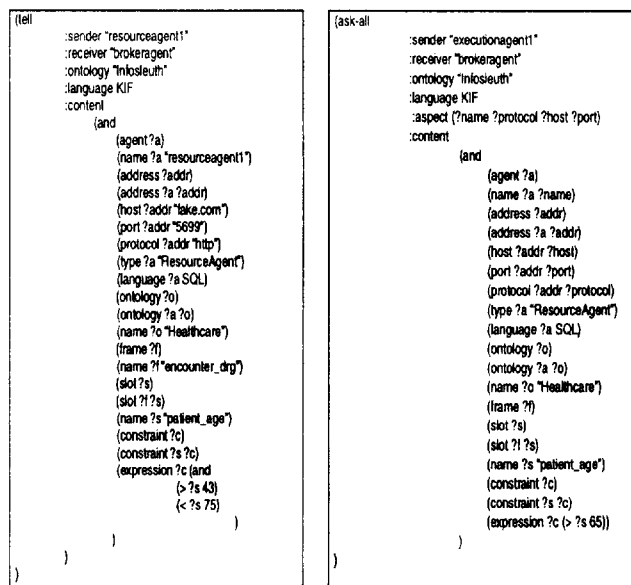


Figure 5: A *Tell* and *Ask-All* advertisements

analysis, and interpretation and evaluation. While most attention has been devoted to the analysis phase involving the mining of patterns from appropriately transformed data, they stress that other phases are also considerably important for knowledge-discovery systems to be successful in practice. The InfoSleuth project is experimenting with providing support for all knowledge discovery phases by tightly integrating data access, data analysis (mining), and data presentation tasks.

One difficulty in establishing a close connection between the data access and data analysis phases of knowledge discovery is a mismatch between how the data is structured within the individual databases, and how it is conceptualized by the user. This mismatch complicates query specification and can lengthen query retrieval time considerably. The InfoSleuth approach is to specify a common ontology for a domain, and local mappings from individual database schemas to the common ontology. These mappings can be thought of as views of the data that simplify query specification for selecting information. Given an appropriate set of mappings for a particular knowledge discovery task, the InfoSleuth system provides query support for selecting relevant information. Furthermore, it pre-processes and transforms the underlying database data into records whose attributes consist of concepts from the ontology; thereby minimizing query retrieval time. When knowledge discovery processes result in new, general concepts, these concepts can also be reflected in the ontology. In support of the data analysis phase, InfoSleuth provides generic analysis agents for performing data summarization [3], classification [9], and deviation detection [1, 29]. As illustrated in Figure 1, the execution of both data access and data analysis components is seamlessly controlled by the task planning and execution agent.

6.2 A Health care Application

The InfoSleuth project is collaborating with the Health care Open Systems Trials (HOST) consortium, partially funded by the U. S. National Institute of Standards and Technology, to develop advanced information technologies for Health care.

We are applying the InfoSleuth technology to help health care administrators in determining how to reduce costs and improve quality of care by providing querying, knowledge discovery, and workflow management capabilities on vast amounts of data stored in the distributed, heterogeneous databases of different hospitals and other care providers.

InfoSleuth is well suited to providing querying capabilities across the databases of different hospitals. The use of a common ontology for the health care domain enables queries to be specified with respect to the ontology, rather than to the idiosyncratic schemas associated with each hospital. Adding a new hospital to the system simply entails adding a new Resource Agent, along with its mapping to the common ontology. The brokering and query decomposition capabilities increase the efficiency of queries by directing local queries only to the databases that are likely to contain the requested information. InfoSleuth ontology-based brokering is the main technology enabling the development of an efficient Master Patient Index (MPI).

The goal of knowledge discovery in this application is to help hospital administrators compare outcomes of treatment (such as average length of stay and incidence of complications) across different care providers (such as hospitals and doctors) for patients with similar risks. Concepts like “patients with similar risks” and “outcomes of treatment” are not explicitly represented in the databases or in the ontologies created for integrating the databases. Therefore, knowledge discovery becomes a 2-step process. The first step involves discovering how to map the data in the databases onto these more abstract concepts and represent them in the ontology. The second step involves discovering how to predict one concept from another, such as predicting the outcomes of a particular treatment given different patient risk factors and health care providers. Association rules finding and deviation detection algorithms are used to carry out the first step, and Bayesian statistics techniques are used to carry out the second step.

7 Related Work

In the SIMS project [4], a model of the application domain is created using a knowledge representation system to establish a fixed vocabulary describing objects in the domain, their attributes, and relationships. For each information source a model is constructed that indicates the data model used, query language, network location, and size estimates, and describes the contents of its fields in relation to the domain model. Queries to SIMS are written in the high-level uniform language of the domain model. SIMS determines the relevant information sources by using the knowledge encoded in the domain model and the models of the information sources. These information sources are determined at run time based on their availability at that time. The InfoSleuth agent-based architecture is an attempt to capture the dynamic availability/unavailability of information sources in a Web-based environment.

TSIMMIS [16] is a system for integrating information. It offers a data model and a common query language that are designed to support the combining of information from structured and semi-structured sources. The emphasis in the TSIMMIS system is that of automatic generation of translators and mediators for accessing and combining information in heterogeneous data sources. The resulting information is expressed in an Object Exchange Model. The Distributed Interoperable Object Model (DIOM) [25] shares similar goals with the TSIMMIS project. Also, similar func-

tionality can be found in the resource agents in the InfoSleuth architecture. Additionally, we are considering approaches for automatic generation of translators and mediators for resource agents. In InfoSleuth, these are presently used by the resource agents to mediate between concepts in a rich domain model and the data stored in a variety of syntactic representations in the data repositories.

The DISCO project [33] provides support for integrating unstable data sources in a dynamic environment. DISCO provides a partial query evaluation scheme that accounts for source unavailability. Mediation in DISCO is based on the ODMG-93 standard object model. A collection of modeling tools are provided to facilitate the wrapping of data sources into ODMG first class objects. To facilitate query mapping and optimization from ODMG’s OQL to the source query languages, the ODMG model is slightly modified.

The Information Manifold [27] is a system for retrieval and organization of information from disparate (structured and unstructured) information sources. The architecture of Information Manifold is based on a knowledge base containing a rich domain model that enables describing the properties of the information sources. The user can interact with the system by browsing the information space (which includes both the knowledge base and the information sources). The presence of descriptions of the information sources also enables the user to pose high-level queries based on the content of the information sources. The focus in the Information Manifold project however is to optimize the execution of a user query expressed in a high-level language which might potentially require access to and combination of content from several information sources [24]. Similar functionality can be found to a limited extent in the Task Planning and Execution Agent in InfoSleuth. The focus in InfoSleuth is to model a dynamic web-based environment where resource agents may join and leave the system dynamically. This information is kept by the broker agent which enables the task planning agent to reformulate its plan to access the relevant information sources.

The OBSERVER project [28] represents an approach for query processing in Global Information Systems. Intentional metadata descriptions organized as domain specific ontologies are used to model and query the information content in various repositories. OBSERVER helps the user to observe a semantic conceptual view of a Global Information System by giving him the ability to browse multiple domain specific ontologies as opposed to individual heterogeneous repositories. Ontology-based interoperation is achieved by navigation of the synonym relationships between terms in the various ontologies. While the present version of the InfoSleuth system lacks ontology-based interoperation found in OBSERVER, it is better able to capture the dynamic nature of a web-based environment where information sources may join or leave the system, through its agent-based architecture.

8 Conclusions

8.1 Current Accomplishments

Our current InfoSleuth design is scalable and portable. This is accomplished through the use of collaborative agents, and the use of Java as a common agent wrapper. Java provides the portability that will be required if InfoSleuth agents need to be deployed dynamically in an unknown environment. Both User Agents, representing individual users, and Resource Agents, representing specific data resources, are

platform-independent. Furthermore, all GUIs are written as Java applets, which can be executed from any browser on any platform.

Internally, multi-threading supports concurrent KQML dialogs between the agents, and allows subtasks to be executed asynchronously. To facilitate communication between agents, we have also implemented KQML in Java.

Our current InfoSleuth release does the following:

- Dynamically integrates heterogeneous data sources while maintaining their local autonomy.
- Executes context-sensitive information-gathering tasks that are capable of dealing with dynamic and uncertain knowledge of the application domain. This is achieved through hybrid declarative/procedural task specifications using rule-based systems with Java procedural attachments.
- Accesses global information flexibly. This is achieved through the use of semantically precise, hierarchically organized ontologies to describe information and data resources. Ontological descriptions capture database schemas (e.g., relational, object-oriented, hierarchical) and conceptual models (e.g., E-R models, Object Models, Business Process models). Users query data based on their ontologies and without regard to the physical representation or the underlying conceptual model.

In addition to its basic functionality, InfoSleuth also provides a suite of GUI tools to perform data mining and statistical analysis, for both general and application-specific data evaluation. Also, InfoSleuth provides the Integrated Management Tool Suite, which provides a complete set of GUI tools for ontology creation and maintenance.

8.2 Lessons Learned

Our experience thus far with InfoSleuth has been very encouraging, and we are continuing to refine and expand its capabilities to meet new needs.

We have faced several issues regarding KQML which we hope to see addressed. First, the KQML specification makes seemingly contradictory assumptions regarding the transport layer. On the one hand, KQML is supposedly “neutral” with regard to transport layer, designed to accommodate TCP, SMTP, email, etc. But the KQML specification makes the assumption that KQML performatives delivered from a single agent to another will arrive in the order in which they were sent; this is an erroneous assumption for many transport layers, including email and TCP (where a single connection is opened and then closed for a single message). Since there is no provision at the KQML level for determining the correct order of messages, we have deviated significantly from the specification to implement streaming of large query results.

Secondly, we have frequently found it necessary to implement various brokering capabilities (*advertise*, *recommend*) at the content-language level rather than at the level of KQML. For example: the KQML “advertise” performative specifies that the content of advertise be another KQML performative of the form that can be accepted by the advertiser. The problem is that it is quite often the case in InfoSleuth that different agents can accept exactly the same performative, but return different results depending on the type of service provided by that agent. For instance, consider the following performative:

```
(ask-all
  :sender A
  :receiver B
  :language SQL
  :ontology healthcare
  :content "select drg_code from encounter"
)
```

If this performative is sent to a resource agent, the result will be the requested data from that single resource. But if it is sent to an execution agent, the result will be the requested data from all relevant resource agents. In normal usage, the sender will only want to send this message to an execution agent, which is capable of doing query decomposition and result integration. But in KQML, there is no way for the sender to distinguish the services provided by the two types of agents based on the advertisement alone. To compensate for this, we pushed most advertising information down into the content-level, reducing *advertise* and *recommend* to simple *tell* and *ask-one/ask-all* queries based on a system-wide InfoSleuth ontology, which represents information about agents and ontologies. As we add more advanced features such as subscription, facilitators, active brokering, etc., we suspect that we will continue to have the same difficulties.

In general, the KQML specification was ambiguous on other key points; it was often necessary to go to the KQML community for guidance on proper usage. As of this writing, an updated KQML specification including a formal semantics for the language is soon to see print, and is eagerly awaited [22].

On a more positive note, we have found ODBC and JDBC to provide true portability that significantly simplified our implementation of a generic resource agent. For example, even though the resource agent could run on Solaris and Windows NT platforms, but not on Sun OS (where Java is not supported), we were still able to access Sun OS Oracle databases thanks to ODBC/JDBC. Drivers for JDBC are not widely available yet for all databases, but availability is growing rapidly. Also, we could have used more transactional support in ODBC/JDBC. Our early experience with providing transaction support in InfoSleuth suggests that we adopt the X/Open XA interface which is currently widely complied with. To this end, we plan to develop a lightweight transaction monitor to support the XA interface.

Another experience that we have learned from using Java is the extent of the achievable code mobility in a system like InfoSleuth. Currently, only user interface applets will be accessible from general browsers like Netscape (as soon as RMI support, which is underway, is completed). This is because InfoSleuth applets do not use any native calls, just the way Java is intended to be used. Making InfoSleuth agents accessible from Internet browsers, however, is not possible under the heavy and necessary use of network communication calls and database interface libraries.

8.3 Future Work

The current design of InfoSleuth has been extensively tested, and successfully used in the health care application domain. Several extensions are currently being investigated and will be included in the future release. Areas of extension include expanding the scope of information that we can examine from InfoSleuth, and extending the scope of our brokering capabilities. Also, we plan to extend the functionality of the current task execution agent to support more complex

tasks. Finally, we plan to expand the functionality of the user agents

Our vision for expanding the information that can be deduced and/or examined includes adding new types of resource agents, including resource agents for LDL, text indexing and retrieval, ontologies, and possibly images. Also at the level of queryable information, we intend to add different data analysis agents, each of which can analyze a set of data in specific ways. These agents will be used in support of the data mining capability.

We intend to enhance the brokering capabilities, splitting the broker agent into a family of cooperating, specialized brokers. We will factor out the syntactic brokering capabilities into a separate type of broker agent, possibly implementing it as an ORB interface using CORBA [30]. Semantic brokering will be available at different levels—for example, local to the site, local to the enterprise, and between enterprises. Semantic brokering may include additional information on contents, and additional semantic information such as quality and cost of information. Furthermore, we plan to implement the capability for the broker to discover information, rather than relying on its being told everything explicitly through advertisement.

We are in the process of splitting the current execution agent into two separate agents, a query decomposition agent and a task execution agent. The task execution agent will develop execution plans based on user requirements using generative planning and plan retrieval utilizing case-based reasoning techniques [17, 31]. The task execution agent may interleave planing with information-gathering subtasks [2, 34, 23] and repair plans when unexpected situations are encountered [10, 26]. Plans will be specified as (transactional) workflows that can be executed by InfoSleuth. It will supervise the execution of the resulting workflows, including managing the transactions it generates. The query decomposition agent will be called by the task execution agent when it has a query over multiple resource agents. It will optimize and decompose queries over multiple resource agents, reassemble the results, and return them to the task execution agent.

We also plan to extend our event monitoring capabilities significantly. This includes developing a complex event specification language and the ability to decompose such events into simpler events on single resource agents. Complex events may include such properties as changes in the result of a query, and sets of simple events and/or operations that happen in a particular sequence or timing.

There are several important directions in which the user agent will be extended. These include developing queryable user profiles containing the user's preferences and a history of his sessions. We also plan to develop applets that aid in visual query specification [6], refinement, and pruning. The user agent will have additional support for security and collaboration.

Acknowledgments

The authors would like to thank M. Huhns, N. Jacobs, B. Perry, and M. Singh for their participation in the early design and implementation of InfoSleuth.

References

[1] A. Arning, R. Agrawal, and P. Raghavan, "A linear method for deviation detection in large databases", In

KDD-96 Proceedings, Second International Conference on Knowledge Discovery and Data Mining, 1996.

- [2] J. Ambros-Ingerson and S. Steel, "Integrating planning, execution and monitoring", In Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88), pages 83-88, St. Paul, MN, 1988.
- [3] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases", In Proceedings of the ACM SIGMOD Conference on Management of Data, 207-216, 1993.
- [4] Y. Arens, C. A. Knoblock and W. Shen, "Query Reformulation for Dynamic Information Integration", *Journal of Intelligent Information Systems*, 1996.
- [5] R. Agrawal, and K. Shim, "Developing tightly-coupled data mining applications on a relational database system". In KDD-96 Proceedings, Second International Conference on Knowledge Discovery and Data Mining, 1996.
- [6] C. Ahlberg, C. Williamson, and B. Shneiderman, "Dynamic queries for information exploration: An implementation and evaluation". In B. Shneiderman, editor, *Sparks of Innovation in Human-Computer Interaction*. Ablex Publishing, 1993.
- [7] "<http://www.mcc.com/projects/carnot>"
- [8] O. Etzioni and D. Weld, "A softbot-based interface to the internet". *Communications of the ACM*, 37(7):72-76, July 1994.
- [9] U. M. Fayyad, S. G. Djorgovski, and N. Weir, "Automating the analysis and cataloging of sky surveys", In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth and Ramasamy Uthurusamy (Editors) *Advances in Knowledge Discovery and Data Mining*, AAAI Press/The MIT Press Menlo Park, California, 1995.
- [10] J. Firby, "Task networks for controlling continuous processes", In Proceedings of the Second International Conference on AI Planning Systems, 1994.
- [11] T. Finin, R. Fritzson, D. McKay, R. McEntire, "KQML as an Agent Communication Language", Proceedings of the Third International Conference on Information and Knowledge Management, ACM Press, November 1994.
- [12] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, "From Data Mining to Knowledge Discovery: An Overview", In U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth (Editors) *Advances in Knowledge Discovery and Data Mining*, AAAI Press: Menlo Park, CA, 1-34, 1995.
- [13] M. R. Genesereth, R. E. Fikes, et al. "Knowledge Interchange Format Version 3 Reference Manual", Logic-92-1, Stanford University Logic Group, 1992.
- [14] M. Genesereth and S. Ketchpel, "Software Agents", *Communications of the ACM*, Vol. 37, No. 7, pp. 48-53, July, 1994.
- [15] T. Gruber, "A translation approach to portable ontology specifications", in *Knowledge Acquisition, An International Journal of Knowledge Acquisition for Knowledge-Based Systems*. 5(2), June 1993.

- [16] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman and J. Widom, "The TSIMMIS Approach to Mediation: Data Models and Languages", In Proceedings of the NGITS (Next Generation Information Technologies and Systems), June 1995.
- [17] K. J. Hammond, "CHEF: A model of case-based planning". In Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86), Philadelphia, PA, 1986.
- [18] M. Huhns, N. Jacobs, T. Ksiezyk, W. M. Shen, M. Singh and P. Canata, "Enterprise Information Modeling and Model Integration in Carnot", Enterprise Integration Modeling: Proceedings of the First International Conference, The MIT Press, 1992.
- [19] "<http://www.mcc.com/projects/infosleuth>"
- [20] N. Jacobs and R. Shea, "The Role of Java in InfoSleuth: Agent-based Exploitation of Heterogeneous Information Resources", IntraNet96 Java Developers Conference, April, 1996.
- [21] C. Knoblock, "Planning, executing, sensing, and replanning for information gathering", In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1995.
- [22] Y. Labrou, "Semantics for an Agent Communication Language", Ph.D. Dissertation, CSEE department, University of Maryland, Baltimore County, September 1996.
- [23] J. Lee, M. Huber, E. Durfee, and P. Kenny, "UM-PRS: An implementation of the procedural reasoning system for multirobot applications", In Proceeding of the AIAA/NASA Conference on Intelligent Robotics in Field, Factory Service and Space, pages 842-859, 1994.
- [24] A. Levy, A. Rajaraman and J. Ordille, "Querying Heterogeneous Information Sources Using Source Descriptions", In Proceedings of the 22nd VLDB Conference, September 1996.
- [25] L. Liu and C. Pu, "The Distributed Interoperable Object Model and its Application to Large-Scale Interoperable Database Systems", Fourth International Conference on Information and Knowledge Management, 1995.
- [26] J. E. Laird, D. J. Pearson, R. M. Jones, and R. E. Wray III, "Dynamic knowledge integration during plan execution", In Proceedings of the AAAI-96 Fall Symposium on Plan Execution: Problems and Issues, 1996.
- [27] A. Levy, D. Srivastava and T. Kirk, "Data Model and Query Evaluation in Global Information Systems", Journal of Intelligent Information Systems 5(2), September 1995.
- [28] E. Mena, V. Kashyap, A. Sheth and A. Illarramendi, "OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies", In Proceedings of the First IFICIS International Conference on Cooperative Information Systems (CoopIS 96), June 1996.
- [29] C. J. Matheus, G. Piatetsky-Shapiro, and D. McNeill, "Selecting and reporting what is interesting", In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth and Ramasamy Uthurusamy (Editors) Advances in Knowledge Discovery and Data Mining, AAAI Press/The MIT Press Menlo Park, California, 1996.
- [30] Object Management Group. "CORBA: The Common Object Request Broker: Architecture and Specification", Release 2.0, July 1995.
- [31] M. V. Nagendra Prasad, Victor. R. Lesser, and S. Lander, "Reasoning and retrieval in distributed case bases", Technical Report 95-27, UMASS, 1995.
- [32] G. Riley, "CLIPS: An Expert System Building Tool", Proceedings of the Technology 2001 Conference, San Jose, CA, December 1991.
- [33] , A. Tomasic, L. Raschid, and P. Valduriez, "Scaling Heterogeneous Distributed Databases and the Design of DISCO", Proceedings of the 16th International Conference on Distributed Computing Systems, Hong Kong, 1995.
- [34] M. Williamson, K. Decker, and K. Sycara, "Unified information and control flow in hierarchical task networks", Technical report, The Robotics Institute, CMU, 1996.
- [35] D. Woelk, M. Huhns and C. Tomlinson. "InfoSleuth Agents: The Next Generation of Active Objects", Object Magazine, July/August, 1995.
- [36] D. Woelk, P. Cannata, M. Huhns, N. Jacobs, T. Ksiezyk, R. Lavender, G. Merdith, K. Ong, W. Shen, M. Singh, and C. Tomlinson, "Carnot Prototype", in Object-Oriented Multidatabase Systems, O. Bukhres and A. Elmagarmid (editors), 1996.
- [37] D. Woelk and C. Tomlinson, "The InfoSleuth Project: Intelligent Search Management via Semantic Agents", Second International World Wide Web Conference, October, 1994.
- [38] C. Zaniolo, "The Logical Data Language (LDL): An Integrated Approach to Logic and Databases", MCC Technical Report STP-LD-328-91, 1991.