A Distributed Architecture for Environmental Information Systems

Martin Purvis, Stephen Cranefield, and Mariusz Nowostawski
Information Science Department
University of Otago
PO Box 56, Dunedin, New Zealand
{mpurvis, scranefield, mnowostawski}@infoscience.otago.ac.nz

Abstract

The increasing availability and variety of large environmental data sets is opening new opportunities for data mining and useful cross-referencing of disparate environmental data sets distributed over a network. In order to take advantage of these opportunities, environmental information systems will need to operate effectively in a distributed, open environment. In this paper, we describe the New Zealand Distributed Information System (NZDIS) software architecture for environmental information systems. In order to optimise extensibility, openness, and flexible query processing, the architecture is organised into collaborating software agents that communicate by means of a standard declarative agent communication language. The metadata of environmental data sources are stored as part of agent ontologies, which represent information models of the domain of the data repository. The agents and the associated ontological framework are designed as much as possible to take advantage of standard object-oriented technology, such as CORBA, UML, and OQL, in order to enhance the openness and accessibility of the system.

1 Introduction

Advancements in electronic data acquisition technology and their widespread adoption have resulted in a heterogeneous data landscape, the characteristics of which must be considered when developing environmental information systems (EIS):

- Data sets are of large size. Because of the ease of automated data acquisition, data sets of huge proportions have been accumulated and continue to grow.
- Data sets are scattered over a network of heterogeneous platforms. Files may be stored on a number of different hardware platforms and under various operating systems.
- Data are stored as different media types and according to differing storage formats.
- The associated metadata of data sets are organized according to differing standards and can have differing interpretations for the same metadata terms.
- Data and processed information must be accessed from a variety of platforms, including Web browsers.

In light of this situation, we have developed a distributed, agent-based architecture designed for robust and extensible environmental information systems. This work is part of the New Zealand

Distributed Information Systems project (http://nzdis.otago.ac.nz), and in the following the architecture presented is referred to as the NZDIS architecture.

First, it is worth emphasizing that environmental information systems involve more than simply accessing and gathering data. Often information processing and analysis algorithms must be applied to the data (possibly from multiple data sources) in order to arrive at the desired information or knowledge. In many cases, a specified environmental analytical task involves the necessity of

- locating and accessing the relevant data sources
- pre-processing and formatting the data to match the input requirements of the analysis module
- processing the input data (this may involve multiple algorithms chained together in possibly novel ways that are specific to each problem presented to the EIS).
- distributing the output information to the appropriate people so that further iterations of the problem-solving approach may be applied quickly.

From this perspective, locating and accessing the data is only one aspect of the overall problem-solving task. Information processing and analysis are also intimately involved and must be supported by an effective EIS. One of the difficulties associated with previous approaches to information system development has been the forced separation of the data model and process model of a system [1]. This separation was at one time thought to be a virtue, when the benefits of well-structured database systems were first being recognized. As a result of this traditional approach, there is still the tendency to maintain this forced separation and build separate, closed environments – one for data access and the other with a fixed set of tools or algorithmic procedures. In many cases, however, this separation can be a hindrance, and the complexity of the current distributed and heterogeneous data landscape can require a more flexible approach so that data and process can be encapsulated into ('object' or 'agent') modules that have the degree of modularity that is appropriate for the given problem domain. The NZDIS architecture is designed to provide this flexibility.

In this paper, the overall architecture of the system is described in Section 2. Some of the key principles of the architecture and the rationale behind some of the design decisions is presented in Section 3. Section 4 is devoted to a discussion of how metainformation and ontologies are used in the architecture, and Section 5 presents conclusions and comparisons with other work in this area.

2 The System Architecture

Consider how a complex problem solving task might be performed manually by a group of human analysts, no one of whom possesses all the knowledge needed to solve the problem. A manager might specify the problem by issuing a brief, high-level statement of the problem to one of his or her chief analysts. This chief analyst doesn't know how to solve the problem by herself, but she has the appropriate experience to know whom to ask to help her solve the problem. First she might go to certain 'knowledge brokers' in various specialized domains, who are acquainted with either specific people in charge of information resources or domain experts who can help solve a particular type of problem. She can then contact these specialists directly. However before she does so, she may first have to plan in her head whom she should contact first, since the input for one special analyst may require the prior problem-solving work of another special analyst. She may also find that the various special analysts use technical terminology that don't match up, and so she may find it necessary to consult some 'translation brokers' who can help explain the terminology used in the various special

analyst reports and put them all on a common footing. Then once the chief analyst has the plan *and* the list of contacts, she will then be in the position to (a) contact each of the recommended domain experts in the right order and then (b) collect and collate the reports that she receives from the experts and produce the final report to the manager.

The NZDIS architecture attempts to borrow some notions from human problem-solving practice (such as that described above) and apply them to the computer information processing tasks for environmental information systems. It does so by employing an agent-based software architecture, where the agents represent encapsulated processing modules that correspond to some of the human agents that were described in the previous paragraph.

2.1 Agent-based software systems

Agent-based software systems consist of a collection of distributed processing modules, called 'agents', that cooperate by sending messages to each other. Agents offer a set of advertised services whose performance can be requested by users or other agents. We expect an agent to have at least the following properties:

- *state*: agents maintain a state that persists between occasions when they are accessed from the outside;
- *goals*: agents have internal goals that they attempt to meet; the goals are normally stored in a declarative fashion.
- *pro-activity*: agents can take the initiative in order to achieve their goals;
- *autonomy*: agents operate without direct intervention from the outside and have control over their actions and internal state;

The representation of agent state and goals in declarative form has advantages with respect to the logical construction of an agent and its extensibility and modification. Consequently it is also convenient for inter-agent communication messages to have a declarative form. Attempts to construct standard protocols for agent message-passing with declarative content, irrespective of the internal structure of the agent, are called agent communication languages (ACLs).

To achieve suitable flexibility, ACLs do not normally specify all possible actions to be taken. Instead, they merely specify the basic intention of a message, typically in the form of speech acts [2], and then include the message content. The basic notion behind speech acts is that spoken messages are usually more than simple statements of fact; they usually imply that some action should be taken upon receipt of the message. The content of the message can be understood by the participants in an agent conversation if they also share a common *ontology* [3], which is a separate, publicly available information model that has been constructed for the given problem domain and serves as a common dictionary for the agents. For environmental information systems, an ontology would include metadata about the various data sets that can be accessed by the system. From these considerations an ACL message might be expected to have

- the speech act performative (one of a set of known speech acts)
- the name of the ontology used in the message
- the sender and recipient of the message
- the message content

2.2 The NZDIS agent architecture

The NZDIS agent architecture has several types of specialized agents that provide various types of service within the system:

- **User Agent**. A user agent is the interface between a user and the other agents in the system. It provides the user interface of the system.
- **Ontology Agent**. An ontology agent provides answers to queries about ontologies and metadata.
- **Broker Agent**. A broker agent accepts registrations from other agents, which advertise their services with the broker agent. It will then use this information to respond to incoming queries from agents that wish to locate a certain type of service provider.
- **Resource Agent**. A resource agent provides an interface between the other agents and the specific command or query language associated a data resource or information processing module.
- **Query Processing Agent**. Query processing agents are part of a specialized subsystem that deal with requests for information from user agents and organize a task plan that will respond to theose requests.

A schematic illustration of the NZDIS architecture is shown in Figure 1. Each agent is depicted by

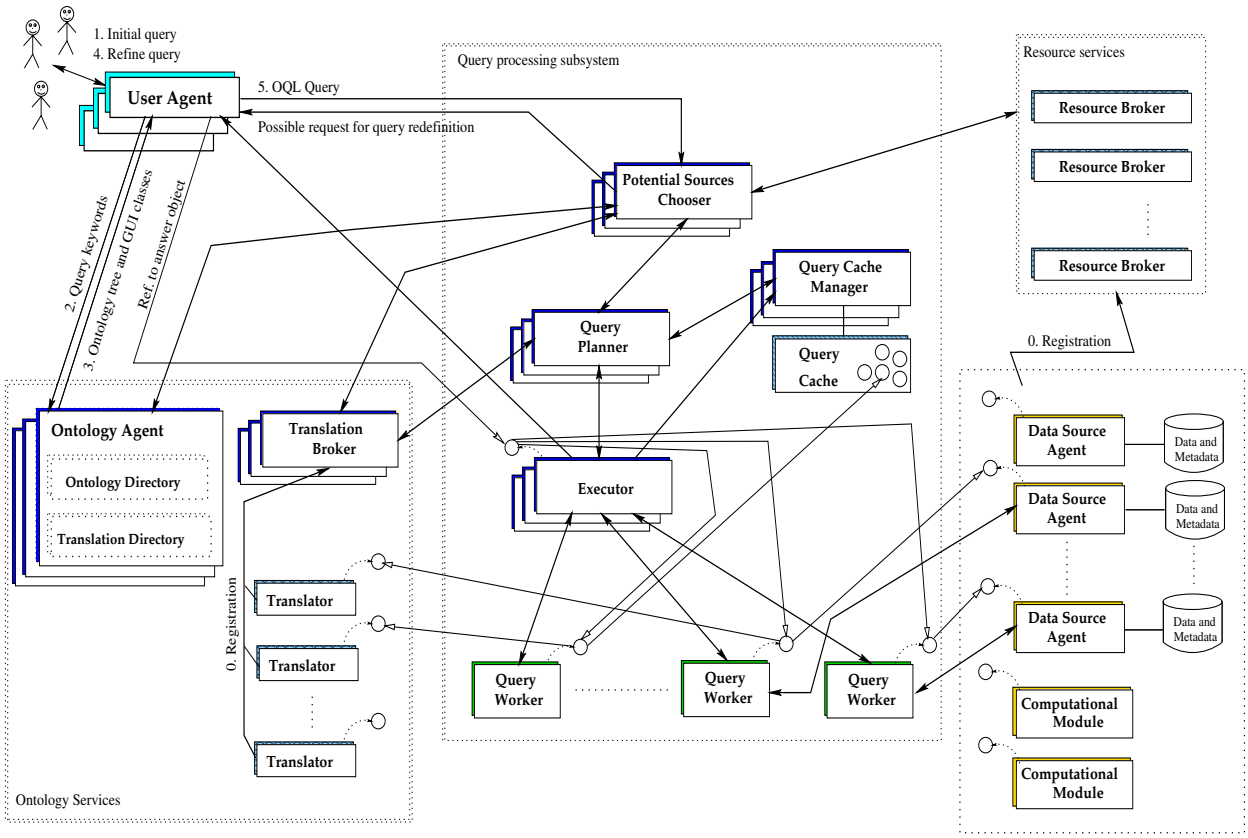

Figure 1. NZDIS agent-based architecture.

a rectangle. Links with black arrowheads indicate messages and links with white arrowheads indicate that an object reference is accessed.

The User Agent is the gateway to the rest of the system. It supports the formulation of queries (which may include requests for computations) by means of an appropriate graphical user interface and by means of a dialogue with the Ontology Agent.

The Ontology Agent manages the interaction between the user agent and individual ontologies, which can be organized in a directory hierarchy. It also maintains a translation directory to keep track of translations that are available between related attributes of separate ontologies.

The Translator Broker agent serves as a broker for individual Translator agents that can perform complex translations between attributes. In the course of information query processing involving multiple data sources, the need may arise to translate values from attribute A in one data source into another attribute B associated with another data source. One can envision three situations:

1. A and B have the exactly the same type and value metrics (no translation necessary).
2. A and B are of the same type, but rescaling is necessary. In this case it is not necessary to call a specialized Translator agent, and the translation can be embedded into the query during the action of the Query Planner (see Figure 1).
3. A more complication functional mapping is required to translate between A and B. In this case a Translator agent will need to be called to perform the required mapping between A and B.

The Translator Broker provides a front-end for these translations and resolves the type of translation that will be required.

On the right side of Figure 1 are shown the Resource Broker Agents along with the Data Source Agents and Computational Module agents. Each Data Source Agent manages the interaction between the system and a particular data source and can be implemented as a wrapper to the data source. Computational module agents provide a similar type of interface to the system for individual processing modules. Resource Broker Agents are a type of broker agent. On startup, a Data Source Agent or Computational module agebt will register with a Resource Broker Agent, identifying the data source that it oversees or computation it performs and the associated ontology. Since data sources may be spread across the Internet, it is appropriate, in order to avoid a single point of failure, for the Resource Broker Agents also to be distributed and to share information with each other.

The Query Processing Subsystem (QPS), shown in the centre of Figure 1, consists of a number of cooperating agents that collectively take a general query from the user and break it down into individual tasks. A query is processed as follows:

- Initially, a FIPA ACL message is sent by the User Agent to the Potential Sources Chooser agent.

- The Potential Sources Chooser agent performs the following:
  - Consults the Ontology Broker agent and Resource Broker agents to identify potential data sources that will need to be accessed to answer the query.
  - Consults the Translator Broker to see if the translation can be performed inline with the query, or if a Translator agent will be needed during plan execution. The Translator Broker agent may

also need to be consulted to check whether there are available Translator agents that can translate between related attributes for multiple data sources. If adequate sources are not available, a message may be sent back to the User agent requesting a reformulation of the query.
  - Ultimately sends the query to the Query Planner agent.

- The Query Planner agent generates a plan of execution by taking the original query and splitting it up into a structured set of subqueries, joins, etc. such that the subqueries can be directed to individual data sources. It first checks the Query Cache Manager agent to see if it is possible to reuse cached results from previous queries. It also consults the Translator Broker agent to see if a translation between related attributes can be performed inline at this stage, or if a Translator agent will be needed to perform the translation later on. The generated plan is then sent to an Executor agent to execute the plan.

- The Executor agent executes tasks that were specified in the generated plan. Individual Query Worker agents are generated to carry out tasks – each performs a query or computation operation with respect to a single data source (accessed via a Data Source agent) or Computational Module. Upon receiving results from a Query Worker agents, the Executor agent
  - generates an answer object and makes the reference to this object available to the User Agent by sending a message back to the User agent, and
  - send the reference to this object to the Query Cache Manager.

- The Query Cache Manager maintains references to objects representing cached queries. It also removes Query Worker Agents when they are no longer needed.

- Data Source agents and Computational Module agents provide an object-oriented query interface to their respective information sources. Note that each Data Source or Computational Module agent may accept queries in a different ontology.

3 Architectural Design Rationale

In this section we describe some of the design decisions and the rationale behind the NZDIS architecture. Our overall design philosophy is to design a flexible state-of-the-art agent architecture while at the same time employing widely-used and industrially tested approaches wherever possible. In particular, we are using standards from the Object Management Group (OMG), wherever possible, because of the widespread availability of software implementations of OMG standards from commercial vendors.

3.1 Agent Communication Language

There are two main ACLs that have been proposed and implemented in the distributed agent research community, KQML (Knowledge Query and Manipulation Language) [4] and FIPA (Foundation for Intelligent Physical Agents) ACL [5]. Both employ the basic notions of speech act theory mentioned above. KQML has been widely used, and there is more field experience with it; FIPA ACL is a more recent offering, but has received endorsements from industry, notably telecommunications organisations, and from major portions of the research community. It is our view that FIPA ACL

will ultimately emerge as the standard, and we have adopted it for our use. The content language of a FIPA ACL message is left unspecified. FIPA has proposed other elements associated with agent architectures to go along with their ACL specification, but these specifications are still under development.

Two key types of message that will be sent in our system are user requests for information sent to the Query Processing Subsystem and reformulated requests sent by the Query Processing Subsystem to Data Source Agents. Within the message content portion of a FIPA ACL message (left unspecified by FIPA), we will use the Object Query Language (OQL) [6] for these two types of messages. OQL is an SQL-like language that has been developed by the Object Database Management Group (ODMG) to have extensions that support the ODMG Object Model [6], which in turn is based on and compatible with the OMG Object Model.

3.2 CORBA

The Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG), which provides a platform for the interoperation of object-oriented systems in a distributed environment, is being used for the implementation of the transport layer of inter-agent communication. The semantics of the distributed agent architecture shown in Figure 1 are being implemented on top of this CORBA-based transport layer. Since CORBA is language neutral, it makes it possible to (a) implement some data source agent wrappers in languages compatible with a given associated legacy data source and (b) implement some elements of the system in languages that are optimised for performance.

In the NZDIS architecture the results of a query are not returned inside an ACL message – instead, the ACL "reply" message contains a CORBA reference to a result set server object that implements a standard interface allowing the result set to be returned in a variety of formats and protocols. This object is created dynamically by the Executor agent and may either already contain the query result, or it may produce the results only when asked do to do so by invoking other dynamically created objects that return the results of subqueries, ontology translations, joins and other operations. These objects are represented by small circles in Figure 1.

3.3 Ontologies

Ontologies are formally specified models that define the concepts used to describe a domain and the relationships that hold between them. We use ontologies to model the semantic structure of individual information sources, as well as to model general aspects of the problem domain that are independent of any particular information source. Several formalisms have been developed for expressing ontologies, notably KIF [7] , KL-ONE [8], and KL-ONE-inspired formalisms based on description logics [10]. We have opted to use the Unified Modeling Language (UML) [9], from the OMG to represent ontologies, because

- UML is intended to be the standard modelling language for object-oriented design. With some extensions, it can be used for general ontological models, which will make it easier to user additional object-oriented approaches, such as OQL for ontological queries.

- UML provides a standard graphical representation (unlike other ontology representation schemes, such as KIF and KL-ONE), and there are commercial tools that use it. Such a graphical representation is important to allow users to browse an ontology and discover concepts that can appear in their queries.

- UML has a large user-community, which will make it easier for new users of our system

- The Object Constraint Language (OCL) [11] allows for the expression of constraints that cannot be described using description logic.

Further discussion of UML and ontologies is given in Section 4.

4 Ontologies and Metainformation

We represent an ontology as a static model consisting of
▫ a UML class diagram to depict the classes in the domain and their relationships.
▫ a UML object diagram to show named instances of the classes
and use the associated OCL in connection with these diagrams. Although UML does not define a standard set of primitive types for attribute and operation declarations, OCL does, and we use these for ontology modelling with UML.

A sample ontology diagram in this format is shown in Figure 2. Note that for the purposes of ontology representation, all attributes are considered to have public visibility. Classes are linked by

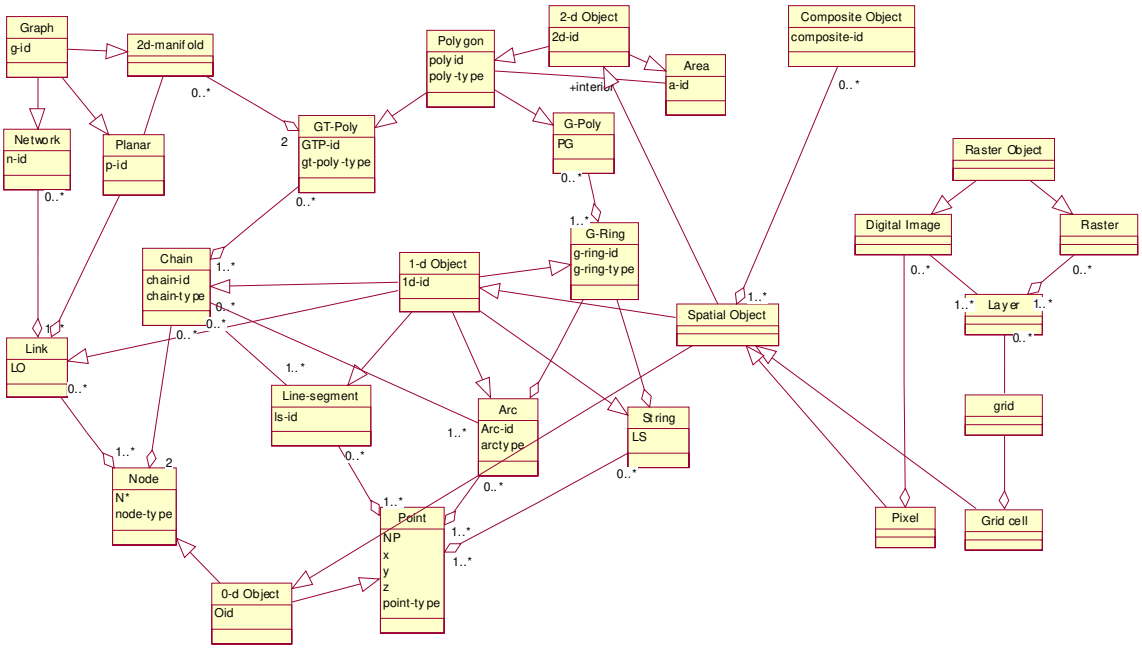

Figure 2. An ontology using UML of the conceptual model of spatial entities in the Spatial Data Transform Standard.

three types of relationship: generalisation (links with arrow heads), aggregation (links with diamonds), and association (straight links). Association and aggregation relationships can be annotated with role names and multiplicity indicators.

OCL enables expressions to be written that can constrain attribute values and possible instances of the relationships. To facilitate these constraint expressions, it features some predefined functions on collections of objects and some simple mapping and iterate functions. However no constraints are shown in Figure 2.

Of course, since an ontology is a formal model of a domain, it is important that the language used to describe it has formal semantics. UML and OCL lacks this at the moment, but this shortcoming is currently being addressed by a number of researchers [12,13,14,15, 16].

It is also important to consider the degree to which automated reasoning can be applied to ontological models. The UML-OCL combination that we are using contains both a highly structured model that *could* support automated reasoning (UML class and object models) and an expressive language (OCL) that would *not* be practical for general purpose reasoning. We believe, however, that the sort of reasoning mostly required for environmental information systems could be performed using the class and object diagrams alone. The constraints (expressed in OCL) could be regarded as extra detail specifying how systems that implement the ontology should behave. To accommodate OCL elements into an automated reasoning environment, it would be possible to define a set of standard OCL constraints that form a language, such as the types of slot constraints provided by description logics. However this is an area for further research.

A single ontology representation language is not necessarily convenient for modelling all domains. It may be useful to have several ontology representation languages available to the ontology designer. The Infosleuth project has an interesting approach to supporting multiple modelling languages [17]. A simple frame-based language is used to define specific ontology representation languages such as object models and entity-relationship diagrams. The actual ontologies are then expressed as instances of these languages. This is a three layer model, with the frame layer acting as a meta-metamodel, the definitions of the ontology representation languages being metamodels and the ontologies themselves being models.

A similar facility is offered by the OMG's Meta Object Facility (MOF) [18]. The MOF is the OMG's Meta-Object Facility standard. It defines a standard for CORBA-based services to manage meta-information in a distributed environment. The MOF defines a model (in fact a meta-meta model) that can be used to define modelling languages such as UML. It also defines interfaces that can be used to populate and query repositories of

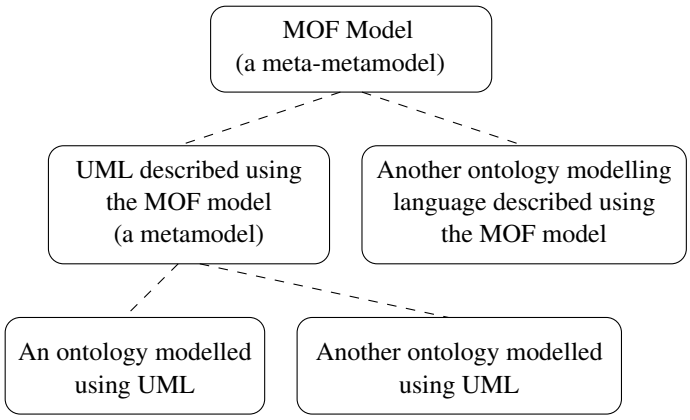

Figure 3. A MOF-based ontology repository.

models defined using various languages. We intend to use this framework to build an ontology server agent with similar capabilities to those of the Infosleuth project. Figure 3 shows the structure of a MOF-based ontology server. While UML provides a standard graphical interface for ontology models, an altogether different interface is needed for agents which may need to query or transfer the model. Several proposals for UML interchange formats have been proposed, and we are using XMI (XML Metadata Interchange), which is a stream-based interchange format for manipulating MOF-based meta objects and UML-based models and appears to be emerging as a standard.

5 Conclusions

We have presented an overview of the NZDIS architecture to be used for distributed environmental information systems. The system employs agent-based software interoperability so that the system can be expanded and enhanced while remaining online.

There are several related projects that have the goal of integrating distributed information resources: the Information Manifold [19], SIMS [20], Infosleuth[17], and OBSERVER [21]. The NZDIS architecture is distinguished from these programmes by the degree to which it offers a distributed agent-based architecture that is constructed on top of a framework using industry-tested, platform-independent object-oriented technology, such as CORBA, OCL, OQL, MOF, XMI, etc.

References

[1] T. Mowbray and R. Malveau, *CORBA Design Patterns,* John Wiley & Sones, Inc., NewYork, 1997.

[2] J. Searle, *Speech Acts*, Cambridge University Press, Cambridge, 1969.

[3] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications", *Knowledge Acquisition*, 5(2), 1993, pp. 199-220.

[4] T. Finin and R. Fritzson and D. Mckay and R. McEntire, "KQML}: An Information and Knowledge Exchange Protocol", in *Knowledge Building and Knowledge Sharing*, K. Fuchi and T. Yokoi (eds.) Ohmsha and IOS Press, 1994.

[5] http://www.fipa.org.

[6] R. G. G. Cattell, D. K. Barry, D. (eds), *The Object Database Standard : Odmg 2.0*, Morgan Kaufmann, 1997.

[7] ANSI X3T2 Ad Hoc Group on KIF, "Knowledge Interchange Format specification" (Working Draft), http://logic.stanford.edu/kif/specification.html, March 1995.

[8] R. J. Brachman and J. G. Schmolze, "An overview of the KL-ONE knowledge representation system", *Cognitive Science*, 9(2), 171-216, 1985.

[9] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA, 1999.

[10] B. Owsnicki-Klewe, "A General Characterisation of Term Description Languages, *Sorts and Types in*

Artificial Intelligence, K.-H. Blasius, U. Hedtstuck and C. Rollinger (eds.), Springer-Verlag LNAI, 418, 1990, pp. 183-189.

[11] J. B. Warmer and A. G. Kleppe, *The Object Constraint Language: Precise Modeling with UML*, Addison-Wesley, Reading, MA, 1998.

[12] R. Breu, R. Grosu, F. Huber, B. Rumpe, and W. Schwerin, "Towards a Precise Semantics for Object-Oriented Modeling Techniques", *Proceedings ECOOP'97 Workshop on Precise Semantics for Object-Oriented Modeling Techniques*, H. Kilov and B. Rumpe (eds.), Technische Universitat Munchen, TUM-I9725, 1997, pp. 53--59.

[13] A. Evans, R. France, Kevin Lano, and B. Rumpe, "Developing the {UML} as a Formal Modelling Notation", *Proceedings of UML'98 International Workshop France, June 3 - 4, 1998*, P. Muller and J. Bezivin, ESSAIM, Mulhouse, France, 1998, pp. 297--307.

[14] G. Overgaard, "A Formal Approach to Relationships in The Unified Modeling Language", *Proceedings PSMT'98 Workshop on Precise Semantics for Modeling Techniques*, M. Broy, D. Coleman, T. S. E. Maibaum, and B. Rumpe, Technische Universität München, TUM-I9803, 1998.

[15] M. Richters and M. Gogolla, "On Formalizing the UML Object Constraint Language OCL", *Proc. 17th Int. Conf. Conceptual Modeling (ER'98)*, T. Ling, and S. Ram, and M. Lee, Springer, Berlin, LNCS 1507, 1998.

[16] A. Hamie, J. Howse, and S. Kent, "Interpreting the Object Constraint Language", *Proceedings of the 5th Asia Pacific Software Engineering Conference (APSEC'98)*, IEEE Press, 1998.

[17] 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, D. Woelk, "InfoSleuth: agent-based semantic integration of information in open and dynamic environments", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, J. Peckham (ed.) June 1997, pp. 195-206.

[18] Distributed Systems Technology Centre, “Meta Object Facility Frequently Asked Questions, http://www.dstc.edu.au/Meta-Object-Facility/MOFAQ.html, 1998.

[19] A. Y. Levy, D.Srivastava and T. Kirk. “Data Model and Query Evaluation in Global Information Systems” *Journal of Intelligent Information Systems*, 5(2), September 1995.

[20] Y. Arens, C. A. Knoblock, and W. Shen, “Query Processing in the SIMS Information Mediator”, *Advanced Planning Technology*, Tate, A. (ed.), AAAI Press, Menlo Park, CA, 1996.

[21] 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”, *Proceedings of First IFCIS International Conference on Cooperative Information Systems (CoopIS’96),* June 1996.