The NZDIS Project: an Agent-Based Distributed Information Systems Architecture

Martin Purvis, Stephen Cranefield, Geoff Bush, Dan Carter,
Bryce McKinlay, Mariusz Nowostawski and Roy Ward
Department of Information Science
University of Otago, Dunedin, New Zealand
{*mpurvis, scranefield*}*@infoscience.otago.ac.nz*

Abstract

This paper describes an architecture for building distributed information systems from existing information resources, based on software agent and distributed object technologies. An agent-based architecture is used, with messages exchanged via the FIPA agent communication language (ACL).

Novel features of this system include the use of standards from the object-oriented programming community such as the Common Object Request Broker Architecture (CORBA) (as a communications infrastructure), the Unified Modeling Language (used as an ontology representation language), the Object Data Management Group's Object Query Language (used for queries) and the Object Management Group's Meta Object Facility (used as the basis for an ontology repository agent). 'Stringified' object references may be returned as the content of ACL messages, allowing query results to be made available via a variety of CORBA interfaces.

1. Introduction

The rapidly increasing extent and availability of electronically stored information the world over has outstripped all efforts to access and make effective use of it, and it is recognised by many governments and organisations that improved technology in this area is of strategic importance. The task is made difficult not so much by the size but by the considerable heterogeneity of the available information sources: data are stored according to widely differing storage formats, media types, and organised according to differing semantics. The challenge is to provide suitable means to integrate such disparate information in a dynamic, open, and distributed environment. Survival in the increasingly competitive global economic climate may well depend on how organisations respond to this challenge. This paper describes the initial design work of the New Zealand Distributed Information Systems[1] (NZDIS) research project, which seeks to develop new tools and techniques that address this problem

The approach taken by the NZDIS project is to harness the potential to be derived from combining various information sources by employing a distributed collection of collaborating software agents. Certainly at the abstract modelling level, the notion of using cooperating agents has several attractive features:

- Using a collection of problem-solvers makes it easier to employ divide-and-conquer strategies in order to solve complex, distributed problems. Each agent only needs to possess the capabilities and resources to solve an individual, local problem.
- The mental image of autonomous, human-like agents facilitates the mapping of real-world problems into a computational domain.
- The idea of several agents cooperating to solve a problem that none could solve individually is a powerful metaphor for thinking about various ways that individual elements can be combined to solve complex problems.

In the last several years there have been a number of research projects pursuing this promising direction [1–5].

In the following sections, we present several perspectives of the NZDIS agent-based software architecture. Section 2 provides a brief overview of agent-based software interoperability, and Section 3 describes why we are adopting some standard technologies as part of the architecture. The overall NZDIS system architecture is presented in Section 4, and

1. This project is funded by the New Zealand government's Public Good Science Fund

the individual (internal) agent architecture is described in Section 5. Section 6 covers issues relating to query processing in distributed information systems. The final section offers some concluding remarks.

2. Multi-agent systems

The notion of agent-based software interoperability is based on the idea of a loosely-coupled collection of agents that can cooperate to achieve a common goal. Each individual agent is presumed to be a specialist for a particular task, and the expectation is that, just as is in the sphere of human engineering, complex projects can be undertaken by a collection of agents, no one of which has the capability of performing all the required tasks of the project. In addition, if the system has an open agent architecture, then individual agents can be replaced by improved models, thereby enabling the system to improve gradually, grow in scope, and generally adapt to changing circumstances. For such an approach to work so that the agents work together effectively, all agents, including those newly introduced to the system, must not only possess a common understanding of possible messages and message types, but also must have an understanding of the kinds of dialogues that can take place between two agents or among groups of agents. For this reason there has been considerable interest in establishing common standards for agent communication, interaction, and knowledge representation.

Two proposals for standard agent communication languages, Knowledge Query and Manipulation Language (KQML) [6] and the Foundation for Intelligent Physical Agents Agent Communication Language (FIPA ACL) [7], have been based on speech acts [8], a concept from linguistics theory that associates human speech with certain communication types (‘performatives’), such as requests, assertions, promises, etc. Each agent message written in such a language identifies the performative associated with the message content.

Communication, however, typically consists in more than the sending of an isolated message — it usually involves the exchange of several messages that take place within the context of a dialogue. These dialogues frequently follow commonly occurring patterns or ‘conversation policies’, and communication can be enhanced if the two participants are explicitly aware of the particular pattern in which they are engaged. Work on conversation policy development is an active research topic [3,4,9,10], but conversation policy standards have yet to emerge.

In order to understand the range of possible messages that can be received, however, an agent must also have, in addition to a common means of characterising performatives and conversation policies, a model of the application domain with which the agent is associated. Such a model, called an ‘ontology’, characterises the relationships and constraints associated with possible entities in the given domain. Most work on ontology representation has so far been based on first-order logic or knowledge representation languages descended from KL-ONE [11], but our approach follows a different line and will be discussed below.

In general the NZDIS approach to the problem of enhancing open-agent systems is to look for (suitably powerful) representation and implementation schemes that have already achieved a wide degree of acceptance in the professional computing arena and that can be effectively incorporated into the agent-based software interoperability agenda. The next section discusses some of the specific methods that we have adopted.

3. NZDIS and object-oriented standards

The variety of data collections in New Zealand available for integration efforts covers an assortment of types, and includes many maps and geographically-oriented data sets that have been assembled by means of automated data acquisition techniques. A large proportion of these collections are not organised according to standard database structuring, but instead are simply available as flat files. In order to provide software technology that improves access to these various collections, we have chosen to build the NZDIS system using industry standards from the object-oriented programming community. This usage is in addition to the use of those standards from FIPA associated with multi-agent system development. The use of object-oriented standards enables the NZDIS project to take advantage of existing commercial implementations of the standards and enhances the stability and maintenance prospects for significant components of the system. The adopted object-oriented ‘standards’ include

- the Object Management Group’s (OMG [12]) Common Object Request Broker Architecture (CORBA) as a communications infrastructure,
- the OMG’s Unified Modeling Language (UML) for representing ontologies (for describing models of both the user-level domain and models of data sources),
- the Object Data Management Group’s (ODMG) Object Query Language (OQL) for expressing queries,
- and the OMG’s Meta Object Facility (MOF) for storing ontologies and models of ontology modelling languages.

3.1. CORBA as a communications infrastructure

The Common Object Request Broker Architecture (CORBA) is an industry standard developed by the Object Management Group for the provision of object-oriented interfaces between systems located on separate platforms. The CORBA standard defines:

- the architecture for the Object Request Broker (ORB)—if ORBs exist on each of two separated machines, then objects on the two machines can be accessed transparently;
- the Interface Definition Language for defining platform-independent object interfaces;
- a set of services associated with remote object access.

There are now commercial implementations of the CORBA standard available on nearly all computer platforms and for most of the widely-used programming languages. Thus CORBA provides a mechanism for linking distributed objects over the Internet, as long as the interfaces can be described precisely. By using CORBA as the transport layer for a distributed multi-agent system, it is possible to combine the high-level agent model with robust commercial implementations of distributed communication.

3.2. UML as an ontology representation language

The most common formalisms used to represent ontologies are the Knowledge Interchange Format (KIF) [13] and KL-ONE style knowledge representation languages [11].

KIF provides a Lisp-like syntax for expressing sentences of first order predicate logic and also provides extensions for representing definitions and meta-knowledge. KIF is a highly expressive but low-level language for representing ontologies; however, the Stanford University Knowledge Sharing Laboratory's ontology editing tool, Ontolingua [14], allows users to build KIF ontologies at a higher level of description by importing predefined ontologies defining concepts such as sets, standard units, time and simple geometrical functions.

Much of the research on ontology design and use is performed by researchers using knowledge representation tools descended from KL-ONE [11]. KL-ONE was the basis for much work in the field of knowledge representation. It implemented "structural inheritance networks": networks containing descriptions of named concepts with generalisation/specialisation links between them. Descendants of KL-ONE include Loom [15] and a family of logics called *description logics* or *terminological logics*[2] [16, 17].

Knowledge representation (KR) systems such as Loom are large and complex systems with a steep learning curve and are little known outside AI laboratories. Instead of using such technology, the authors are investigating the rapidly growing and more mainstream arena of object-oriented technology to construct a distributed information retrieval and processing system. Currently there is no counterpart for the deductive capabilities of KR systems in current object-oriented technology; however, for distributed information systems these capabilities are not necessarily needed. Many of the benefits of KR systems occur during the process of designing an ontology. This support is undoubtedly useful, but in the object-oriented world there is also much support available for the design of models, with mature and commonly used languages, methodologies and tools available.

The other function of KR systems — to store highly structured data and answer queries about it — is not an issue in distributed information systems. The point of distributed information systems technology is to allow disparate databases and other information sources to be integrated. Nothing can or should be assumed about the underlying databases and information storage systems. In particular, it cannot be assumed that the information sources will be implemented using KR systems.

The ontology representation formalism used in the NZDIS project is a subset of the OMG's Unified Modeling Language (UML) [18], together with its associated Object Constraint Language (OCL) [19, 20]. Benefits of using UML and OCL include the following:

- UML has a very large and rapidly expanding user community. Users of distributed information system infrastructures will be more likely to be familiar with this notation than KIF or description logics. This issue should not be overlooked for its importance in gaining acceptance of distributed information systems technology amongst new end-user communities.
- Unlike description logic formalisms, there is a standard graphical representation for models expressed in UML. Such a graphical representation is important to allow users of distributed information systems to browse an ontology and discover concepts that can appear in their queries. In contrast, a description logic has a linear syntax but no standard graphical representation. Although UML currently has no standard linear syntax, the OMG is in the process of adopting XMI (XML Model Interchange) as a standard for stream-based model interchange [21].
- The Object Constraint Language (OCL) is powerful and allows the expression of constraints that cannot be described using description logic. (Of course, there is

2. In a description logic, concepts can be introduced by simply naming them and specifying where they fit in the generalisation/specialisation hierarchy. Concepts may be specialised by operations such as *value restriction*, where the possible values of some concept *role* (effectively an attribute) are restricted to be instances of a certain class, and *number restriction*, where the operators `atleast` and `atmost` are used to restrict the possible number of values that a given role may have.

a trade-off between the expressive power of a language and the computational complexity of reasoning about it.)

UML defines several types of diagram that can be used to model the static and dynamic behaviour of a system. We have chosen to model an ontology as a static model consisting of a class diagram to depict the classes in the domain and their relationships, and an object diagram to show particular named instances of those classes (see [22] for more details).

3.3. The Object Query Language

As we have chosen the object-oriented modelling language UML to represent ontologies, it follows that queries involving concepts in these ontologies would be best represented using an object-oriented query language. The ODMG's Object Query Language (OQL) [23] is an industry standard for expressing queries over an object-oriented data model and we have therefore chosen it as the query language for our system.

3.4. A MOF-based ontology repository

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 addressing this issue by supporting multiple modelling languages [1]. 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) [24–26] The MOF defines a standard for CORBA-based services to manage meta-information in a distributed environment. It defines a model (in fact a meta-meta model) that can be used to describe modelling languages such as UML. It also defines interfaces that can be used to populate and query repositories of 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.

4. System architecture

A schematic representation of the NZDIS system architecture is shown in Figure 1. In this figure, solid rectangles represent agents and circles represent CORBA objects. Arrows with solid heads depict agent messages expressed in FIPA ACL and hollow-headed arrows represent references to remote CORBA objects. The architecture also includes a standard FIPA Directory Facilitator (not shown in the figure) which each agent is required to register with when it is incorporated into the system.

For example, suppose there are two separate data collections available:

- one collection containing questionnaire responses pertaining to asthma incidence and including geographical location in NZ Map Grid coordinates of respondents (described by the AsthmaOnt ontology)
- and another collection containing climatological information with respect to geographical location (described by the ClimateOnt ontology).

A user may be interested in collecting records for all the people who suffer from asthma and who live in areas with average humidity greater than 70%. This query is entered via a dialogue with the User Agent which interacts with the Ontology Agent in order to get references to relevant ontologies. A query message is then sent to the Query Preprocessor (using FIPA ACL) where the message content is the following OQL query:

```
select p
from AsthmaOnt:person as p,
     ClimateOnt:Environment as e
where e.humidity > 70
  and p.Response.hasAsthma = true
  and e.northing = p.northing
  and e.easting = p.easting
```

The Query Preprocessor constructs an initial representation of the parsed query and passes this to the Potential Sources Chooser agent, which identifies possible data sources that have been registered with a Resource Broker agent. References to the potential data sources are then passed to the Query Planner, which generates an appropriate plan for processing the query. The potential data sources may have data that do not match precisely the form requested in the query but which are related to the requested form by means of a known transformation. In that case a Translator agent which can perform this transformation may already have been registered with the Translation Broker. The Query Planner checks with the Translation Broker to determine whether any Translator agents are to be used in connection with the query processing. (In the simple query given above, it is assumed that no translations between ontologies are needed).

When the plan for query processing has been constructed, it is passed to the Executor. The Executor generates one or more Query Workers, which operate in their

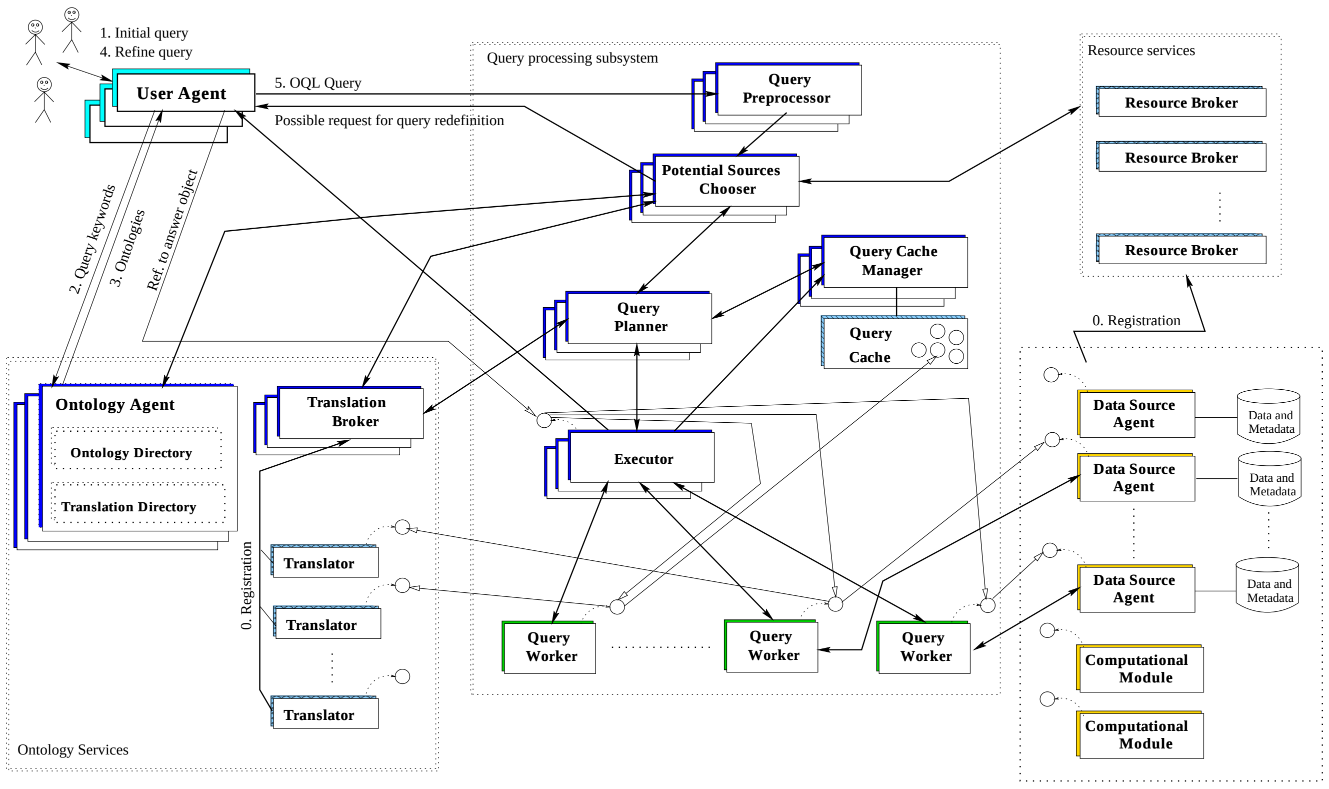

Figure 1. The system architecture

own threads and interact with Data Source Agents that are responsible for a given data source. Typically, Data Source agents serve as wrappers for existing databases. For the example given above, one Data Source Agent is associated with the environmental data and another Data Source Agent is associated with the survey-questionnaire data.

When the Query Worker agents obtain CORBA object references to individual query responses requested of individual Data Source Agents, these references are passed back (as the content of FIPA ACL messages) to the Executor, which merges the information in a form appropriate to the original query and passes a reference to this merged information back to the User Agent. The User Agent can check this reference to see what methods the answer object has available for transferring or processing the data. References to the information obtained by the Query Worker agents are also maintained by the Query Cache Manager agent for possible reuse in subsequent queries.

In some circumstances an information system query could require further computation to be performed on available data. In that case a Query Worker agent can make use of the services performed by a Computational Module agent (bottom right corner, Figure 1), rather than a Data Source Agent. A Computational Module agent provides a wrapper for a module that performs some specialist computation, such as statistical or connectionist analysis.

Note that Figure 1 shows that the system architecture can accommodate multiple instances of many of the agent types, such as the various broker agents. These multiple instances are all registered with the System Facilitator and may be located anywhere across the distributed environment.

5. The internal agent architecture

Individual agents in the NZDIS system have an architectural organisation whose common elements are depicted schematically in Figure 2. This figure describes an abstract, high-level architecture which may vary somewhat with individual agents within the system. The Agent Executive is in control of the agent. A message handler operates one or more input and output queues of the agent and makes them available to the Agent Executive. The Belief States component contains the agent's current state in a declarative form. Agents may also contain a record of the capabilities of the agent (individual agents may optionally also be equipped with special operational capabilities, which is a potential component shown at the bottom of Figure 2). Separate Conversation components also exist, each of which maintain the state of one of the agent's current conversation dialogues in accordance with a conversation policy appropriate to the

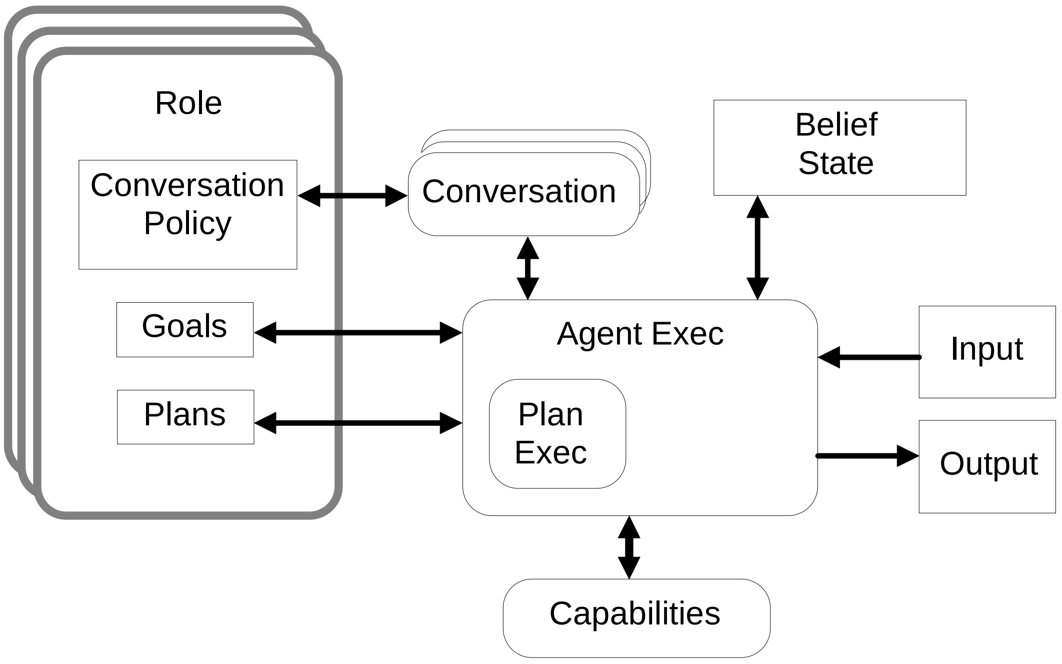

Figure 2. The internal agent architecture

agent's role in that conversation.

When an agent receives a message, it will execute a 'plan' in order to respond to the message. For some simple agents, the plan can be hard-coded into the agent, but in general, the plan can be considered to be a script that will be interpreted by the Plan Executive component of the Agent. On the left side of Figure 2 are three components that are collectively designated to be a 'role' for the agent. A role comprises a conversation policy, a set of goals to be fulfilled, and the plans required to achieve those goals. For example when an agent registers with a broker agent, it performs the role 'registrant'.

An agent can have more than one role, but at the present time these roles are fixed in NZDIS agents. However, we are investigating the utility of allowing agents to adopt new roles at runtime, and so the notion of a role is being maintained as a potentially modifiable architecture element. This scheme will allow for agents to accept messages that cause them to install new goals or plans for a given role, or even install new roles for the agent.

6. Query processing

The query processing subsystem is responsible for accepting OQL queries from the user agent, creating and optimising a query plan involving multiple data source agents, and controlling the execution of that plan to generate the resulting data set. It must also translate terms from the user's domain ontologies into the ontologies representing the data source agents, and then translate the resulting data back into the original ontologies. This is a complex process and is currently the least developed part of our architecture. The current query processing subsystem has limited functionality designed to support our current prototype and is not intended as a final solution.

This section briefly overviews the query processing techniques used in related research, by classifying these systems along a number of key architectural dimensions, and discusses how well these techniques fit with our own needs. It should be noted that the systems discussed cannot be directly compared as competing solutions to the same problem, as they have differing aims, some being intended as fully-functional federated database systems, some as flexible and easily extensible information retrieval systems, and some as components within a single database implementation.

Ontology representation and translation A fundamental design decision in any distributed information retrieval system is the formalism used to describe the individual data sources (the *export schemas* or *data source ontologies*) and the user-level domain model(s) (*global schemas* or *domain ontologies*). Generally, individual data sources represent their knowledge using different ontologies, which means that a mechanism must be provided for translating subqueries from the domain ontology (or ontologies) into the appropriate data source ontology, and then in turn, translating the resulting dataset back into the ontologies used in the query.

Ontology languages used in existing projects include AI representation languages such as Loom (SIMS [27]) and description logic (OBSERVER [28]), a Datalog-like language (Infomaster [2]) and the Object Data Management Group's Object Definition Language (ODL) (DISCO [29], MIND [30, 31]). The Infosleuth project [1] uses a metamodelling approach to allow new ontology representation languages to be defined and used to model data sources.

One approach to the ontology translation problem (or *schema integration*) problem is to initially define a global schema integrating the schemas of all data sources, and to define views mapping from various sets of data source schemas to the global schema [32]. The decomposition of queries into subqueries on individual data sources is essentially hardwired into the system. The MIND system takes this approach, using ODL to define the global schema as a view over data source schemas. This has the disadvantage of requiring the domain model and query processing system to be regenerated whenever a data source is changed or added. The DISCO system includes a number of mechanisms that alleviate but do not eliminate this problem.

An alternative approach, used in SIMS, assumes the prior existence of a domain model and symbolically represents mappings from data source models to the domain model. This information is then used dynamically by the query planner to decompose queries. If a data source is added or modified, the models and mappings of other data sources are unaffected, thus trading off increased complexity of query planning for greater ease of system extension. The limited expressiveness of Loom concept definitions (compared to object-oriented modelling techniques) simplifies the types of mappings required and helps to make this approach viable.

The OBSERVER system (based on description logic) uses a different approach. Rather than assuming the existence of a completely defined mapping from data source models to a domain model, OBSERVER includes a component called the Interontology Relationships Manager (IRM)—a repository for relationships between pairs of terms in different ontologies. Relationships supported are synonym, hyponym, hypernym, overlap, disjoint and covering. These relationships are used during query planning and transformer functions are also provided to convert query results from from one ontology to another.

One additional approach, taken by Infosleuth, is to leave ontology translation to be the responsibility of the data source wrappers [33]. In Infosleuth, each information agent advertises which part of the domain ontology it supports and it is the responsibility of the wrapper to translate between that ontology subset and the schema of the underlying data source.

Our choice of UML as an ontology representation language provides us a richer domain model than the projects discussed above, making a general approach to translation based on fully defined mappings an unrealistic goal. Instead we plan to adapt the idea of OBSERVER's Interontology Relationships Manager to allow multiple ontology translation agents to be registered with a broker, each having the expertise for translating particular pairs of ontologies. This framework will also allow us to support user queries containing terms from more than one domain ontology.

Query representation language The choice of a query language is an important determinant of the techniques used for query processing. Distributed information systems using OQL include MIND and DISCO. Alternative query languages used by other systems include programs in a Datalog-like language (Infomaster), description logic expressions (OBSERVER) and conjuncts of atoms referring to concepts and relationships defined in an AI knowledge representation language (SIMS). The query processing techniques used by these systems are not directly applicable to our architecture, although they have identified the important components and problems to be addressed in such a system.

As queries must be decomposed into subqueries over single data sources and then optimised, it is important to choose an appropriate intermediate format that supports these processes and into which the initial OQL query can be parsed. Because OQL queries commonly include path expressions (denoting one or more navigations from one object to another via references) and may include nested queries, query optimisation is an important consideration and many query algebras have been proposed to represent object oriented queries, most based on extensions to the nested relational algebra (see, e.g. [34] for a discussion). The ability of OQL queries to represent sets, bags or lists adds additional problems and has led to the development of representations such as the monoid comprehension calculus [34, 35] that provide a canonical representation for OQL queries and a uniform treatment of operations that apply to all three collection types.

Internal plan representation While most work represents a query plan as a tree over some appropriate query algebra, with the flow of data represented by the links from nodes to their parents, the SIMS project takes the alternative approach of using a general-purpose AI nonlinear planner to generate partially ordered plans comprising operators representing actions such as joins and data transfer. This is a powerful technique but the implicit representation of data flow (compared to algebraic expressions) may make it difficult to apply some of the algebraic optimisations commonly used for object-oriented query processing.

Currently we have adopted an algebraic representation of query plans but we are investigating the alternative approach taken by SIMS.

Query optimisation Techniques discussed in the literature for optimising queries are motivated by various aims.

Work on object-oriented query optimisation addresses the problem of finding the optimal algebraic form of a query (e.g. by eliminating nested subqueries and combining joins, function applications and grouping operations into a single operator) [34]. Cost functions are generally used to make a final translation into a physical algebra representing the operations that can be performed by a database. A common technique for this process is by using rewrite rules on algebraic structures with extra cost information attached to nodes [35–37]. Various search strategies have been used, including special-purpose algorithms and combinations of branch and bound search with random techniques such as random walk, iterative improvement and simulated annealing [38]. SIMS also uses rewriting techniques to optimise its partially ordered query plans [39].

When queries need to be split across multiple databases there are two additional considerations in query optimisation. First, the query processor may have information about the capabilities of the individual component databases that can be used to optimise the subqueries sent to them. For example, DISCO requires database wrappers to provide information about the algebraic query expressions they support (this is in the form of a grammar) and optional cost information about the algebra operators supported. Multidatabase systems assume that the component databases are autonomous and do not share such information, but techniques have been developed to estimate the costs of queries to local databases based on measurements from previous queries [31].

The second issue in optimising distributed query plans is estimating and reducing the network costs of transferring intermediate result sets between hosts. In distributed database systems, semi-joins are commonly used to minimise the data transferred across the network in order to join data from different databases [40]. A semi-join is executed by projecting one relation over the common join attribute, shipping the projection to the site of the other relation and then performing a join of the projection with that relation. This eliminates tuples that cannot be in the result. The final join is then computed at one site using the reduced data set. Various heuristics have been developed for finding an efficient schedule of semi-join operations for general queries (an NP-hard problem).

Another possible goal for query optimisation is to ensure the result set is accurate within some user-specified limit. This is necessary where answers can be obtained from multiple data sources with different degrees of quality, or when information may be lost or altered when translating between ontologies [28].

As the potential data sources to be linked into NZDIS include large data sets of environmental information, minimising network costs will be a crucial goal for our system. Our optimisation efforts will therefore focus on this issue.

7. Conclusions

The NZDIS architecture is designed to provide an open, agent-based environment for the integration of disparate sources of information. The intended use of the system is expected to lie between two ends of a spectrum of possible data gathering applications: at one end are tightly-integrated database systems, where existing distributed database system techniques could be used, and at the other end are information sources distributed so widely that only Web-based information discovery systems are practically feasible. Since many of the information sources to be integrated in the New Zealand context are expected to consist of flat or unstructured data files, the system does not presume to be a distributed database system and does not perform optimisations based on such an assumption. Instead, the NZDIS system offers the infrastructural components for integrating heterogeneous data sources with known, but differing, collections of data and metadata.

The system is designed to use existing, commercially-tested object-oriented technology wherever possible and so be accessible to a wide range of potential adopters in New Zealand.

References

[1] 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. Infosleuth: agent-based semantic integration of information in open and dynamic environments. In Joan Peckham, editor, *Proceedings of the ACM SIGMOD international conference on management of data*, SIGMOD Record 26(2), pages 195–206, June 1997.

[2] O. M. Duschka and M. R. Genesereth. Query planning in Infomaster. In *Proceedings of the 12th Annual ACM Symposium on Applied Computing (SAC'97)*, 1997. http://logic.stanford.edu/people/duschka/papers/Infomaster.ps.

[3] H. Nwana, D. Ndumu, L. Lee, and J. Collis. ZEUS: A toolkit for building distributed multi-agent systems. *Applied Artifical Intelligence*, 13(1):129–186, 1999.

[4] R. S. Cost, T. Finin, Y. Labrou, X. Luan, Y. Peng, I. Soboroff, J. Mayfield, and A. Boughannam. Jackal: A Java-based tool for agent development. In J. Baxter and B. Logan, editors, *Software Tools for Developing Agents: Papers from the 1998 AAAI Workshop*. Technical Report WS-98-10, AAAI Press, 1998.

[5] C. A. Knoblock and J. L. Ambite. Agents for information gathering. In J. Bradshaw, editor, *Software Agents*. AAAI/MIT Press, 1997.

[6] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, July 1994.

[7] FIPA 97 specification documents. http://www.fipa.org/spec/, 1997.

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

[9] M. H. Nodine and A. Unruh. Constructing robust conversation policies in dynamic agent communities. Technical Report MCC-INSL-020-99, Microelectronics and Computer Technology Corporation, 1999.

[10] M. Greaves, H. Holmback, and J. M. Bradshaw. What is a conversation policy? In M. Greaves and J. M. Bradshaw, editors, *Proceedings of the Autonomous Agents '99 Workshop on Specifying and Implementing Conversation Policies*, 1999.

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

[12] Object Management Group. OMG homepage. http://www.omg.org/, 1998.

[13] National Committee for Information Technology Standards Technical Committee T2 (Information Interchange and Interpretation). Draft proposed American national standard for Knowledge Interchange Format. http://logic.stanford.edu/kif/dpans.html, 1998.

[14] Adam Farquhar, Richard Fikes, and James Rice. The Ontolingua Server: a tool for collaborative ontology construction. In *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96)*, 1996.

[15] Information Sciences Institute. Loom project home page. http://www.isi.edu/isd/LOOM/LOOM-HOME.html, 1998.

[16] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge Representation and Reasoning*, Studies in Logic, Language and Information, pages 193–238. CLSI Publications, 1996.

[17] Bernd Owsnicki-Klewe. A general characterisation of term description languages. In K.-H. Bläsius, U. Hedtstück, and C. Rollinger, editors, *Sorts and Types in Artificial Intelligence*, Lecture Notes in Artificial Intelligence 418, pages 183–189. Springer-Verlag, 1990.

[18] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[19] Object Management Group. Object Constraint Language specification. ftp://ftp.omg.org/pub/docs/ad/97-08-08.pdf, September 1997.

[20] Jos B. Warmer and Anneke G. Kleppe. *The Object Constraint Language: Precise Modeling With UML*. Addison-Wesley, 1998.

[21] Distributed Systems Technology Centre. XMI spec recommended. News item on Meta-Object Facility Information Web Page, http://www.dstc.edu.au/Meta-Object-Facility/, January 1999.

[22] S. Cranefield and M. Purvis. UML as an ontology modelling language. In *Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.

[23] R.G.G. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade, editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, 1997.

[24] Object Management Group. MOF specification. http://www.omg.org/techprocess/meetings/schedule/Technology_Adoptions.html#tbl_MOF_Specification, 1997.

[25] Stephen Crawley, Simon McBride, and Kerry Raymond. Meta-Object Facility tutorial (draft). http://www.dstc.edu.au/Meta-Object-Facility/Tutorial.html, 1997.

[26] Distributed Systems Technology Centre. Meta Object Facility frequently asked questions. http://www.dstc.edu.au/Meta-Object-Facility/MOFAQ.html, 1998.

[27] Y. Arens, C. A. Knoblock, and W.-M. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6(2/3):99–130, 1996.

[28] E. Mena, A. Illarramendi, V. Kashyap, and A. Sheth. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Distributed and Parallel Databases*, 1999. (to appear), http://siul02.si.ehu.es/~jirgbdat/PUBLICATIONS/dapd99.ps.gz.

[29] A. Tomasic, L. Raschid, and P. Valduriez. Scaling access to heterogeneous data sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering*, 10(1), 1998.

[30] A. Dogac, C. Dengi, E. Kilic, G. Ozhan, F. Ozcan, S. Nural, C. Evrendilek, U. Halici, B. Arpinar, P. Koksal, N. Kesim, and S. Mancuhan. METU interoperable database system. *ACM SIGMOD Record*, 24(3), September 1995.

[31] F. Ozcan, S. Nural, P. Koksal, C. Evrendilek, and A. Dogac. Dynamic query optimization on a distributed object management platform. In *Proceedings of the 5th International Conference on Information and Knowledge Management (CIKM'96)*, 1996. ftp://ftp.srdc.metu.edu.tr/pub/mind/papers/cikm_96.ps.Z.

[32] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[33] B. Perry, M. Taylor, and A. Unruh. Information aggregation and agent interaction patterns in infosleuth. Technical Report MCC-INSL-104-98, Microelectronics and Computer Technology Corporation, 1998. http://www.mcc.com/projects/infosleuth/publications/TR98/INSL-104-98.pdf.

[34] T. Grust, J. Kroeger, D. Gluche, A. Heuer, and M. Scholl. Query evaluation in CROQUE: Calculus and algebra coincide. In C. Small, P. Douglas, R. Johnson, P. King, and N. Martin, editors, *Proceedings of the 15th British National Conference on Databases (BNCOD15)*, Lecture Notes in Computer Science 1271, pages 84–100. Springer, 1997. http://wwwdb.informatik.uni-rostock.de/ jo/bncod15.ps.gz.

[35] L. Fegaras. An experimental optimizer for OQL. Technical Report TR-CSE-97-007, University of Texas at Arlington, 1997. http://lambda.uta.edu/oqlopt.ps.gz.

[36] J. A. Blakeley, W. J. McKenna, and G. Graefe. Experiences building the open OODB query optimizer. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 287–296. ACM Press, 1993. ftp://ftp.cs.pdx.edu/pub/faculty/graefe/papers/OpenOODB.ps.

[37] M. Cherniak and S. B. Zdonik. Rule languages and internal algebras for rule-based optimizers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1996.

[38] J. Kröger, R. Illner, S. Rost, and A. Heuer. Query rewriting and search in CROQUE. Preprint CS-15-98, Computer Science Department, University of Rostock, 1998. http://wwwdb.informatik.uni-rostock.de/ jo/CS-15-98.html.

[39] J. L. Ambite and C. A. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*, 1997.

[40] J. M. Morrissey, S. Bandyopadhyay, and W. T. Bealor. A heuristic for minimizing total cost in disributed query processing. *Journal of Computing and Information*, 1(2):736–758, 1995. Special Issue: Proceedings of the 7th International Conference of Computing and Information (ICCI'95).