# Platforms for Agent-Oriented Software Engineering

Mariusz Nowostawski Geoff Bush Martin Purvis Stephen Cranefield

*Department of Information Science, University of Otago*
*P.O. Box 56, Dunedin, New Zealand*
*{mnowostawski, gbush, mpurvis, scranefield}@infoscience.otago.ac.nz*

***Abstract***

*The use of modelling abstractions to map from items in the real-world to objects in the computational domain is useful both for the effective implementation of abstract problem solutions and for the management of software complexity. This paper discusses the new approach of agent-oriented software engineering (AOSE), which uses the notion of an autonomous agent as its fundamental modelling abstraction. For the AOSE approach to be fully exploited, software engineers must be able to gain leverage from an agent software architecture and framework, and there are several such frameworks now publicly available. At the present time, however, there is little information concerning the options that are available and what needs to be considered when choosing or developing an agent framework. We consider three different agent software architectures that are (or will be) publicly available and evaluate some of the design and architectural differences and trade-offs that are associated with them and their impact on agent-oriented software development. Our discussion examines these frameworks in the context of an example in the area of distributed information systems.*

## 1. Introduction

Expansions and performance improvements in hardware, data storage facilities, and telecommunications has led to ever more ambitious software engineering projects, which are approaching the point where they are so complex that they are unmanageable. Just accessing all the necessary data is made difficult by the fact that the various information stores are often distributed across various remote sites, stored on different platforms and in different formats, and organised according to differing organisational schemas and semantic models. Traditional software engineering development techniques based on functional analysis and data flow, or on object-oriented modelling, are in many cases not sufficient, in themselves, to capture needed dynamism and flexibility of some of the current development tasks that are undertaken. Researchers are now seeking new methods and approaches that can help software engineers grapple with some of these problems. One of the new approaches that has been proposed is agent-oriented software engineering (AOSE) [16].

The fundamental notion on which agent-oriented software engineering is based is that of the autonomous agent [13,20]. To see the advantages of this approach, consider what has to be done when a complex, real-world system is modelled. The model will comprise individual components that are part of a larger structure. For the model to be effective, we observe that

a) it is natural to conceptualise the relevant features, *i.e.* the behavioural components to be modelled, in terms of simple and familiar elements; and
b) the overall model structure and the number of individual elements *must* be kept simple enough so that the entire model can be easily understood, manipulated, and modified, if necessary.

For complex systems, requirement *b* means that the individual elements must represent rather complex modelling "chunks". It is appropriate to express these dynamic "chunks" in terms of complex entities from the world with which we are already familiar (due to requirement *a*), *i.e.* human agents. Although the anthropomorphic approach is decried in some elementary science textbooks as medieval, this is a natural way to *begin* building a model.

Agent modelling in software engineering is a relatively young area, and there are, as yet, no standard methodologies, development tools, or software architectures. There are, however, some software frameworks that are beginning to appear, and in this paper we describe three of them that use agent-oriented technology and which are interesting candidates to the software developer because they are (or are about to be) freely available and come with the source code. Here we

– describe the software frameworks and the architectures that they presume or prescribe, and
– discuss some of the implications and trade-offs for software engineering behind the differences between the respective framework architectures.

The three AOSE frameworks that we will discuss are

1. The ZEUS [5,6,19] system toolkit, an agent-based system developed at British Telecom Laboratories,
2. JADE [2] (Java Agent DEvelopment framework), an open-source agent-based software development project at the Telecom Italia Group Company, and
3. the agent-based infrastructure associated with the New Zealand Distributed Information Systems (NZDIS) project [22], which we will call the "NZDIS architecture".

## 2. Agent-oriented software engineering

The notions of *intelligent agents* and *agent-based systems* have emerged from artificial intelligence research [28] and, although there is still debate over what constitutes the precise notion of an agent (see [13,20] for discussion), there is now a movement towards applying these ideas to mainstream software engineering practice [14]. Those behind this movement assert that key techniques for managing complexity, such as decomposition, abstraction, and organisation [3] can be comfortably accommodated within the agent-based modelling paradigm. But, in addition as mentioned in the previous section, the agent-oriented approach supports the notion of personification, a commonly used real-world abstraction: it can be straightforward to personify software components, endowing them with human-like intentions and abilities. Consequently the promoters of AOSE argue that "agent-oriented approaches can significantly enhance our ability to model, design and build complex (distributed) software systems" [16].

Agent-oriented techniques can be employed during both the design phase, where agent-based techniques can be used to model the problem domain and the system design, and during the implementation phase, where agent-oriented development tools would be used. The use of the agent-based approach in both phases is a natural fit, but not required. Not to use them together, however, would fail to take advantage of the natural mapping from one phase to the other and would be like implementing an object-oriented design using a procedural programming language. Here we will make reference to agent-oriented design but will concentrate our discussion on agent-oriented development frameworks.

At the present stage of AOSE development, there are no commonly used agent-based programming languages, not to mention compilers or interpreters. In fact there is a significant range of opinion concerning what specific properties that an agent should have and consequently some difference of opinion concerning what should be the internal workings of an agent. We do expect an agent to have at least the following properties though:

- *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;

Moreover there is considerable support for an internal organisation of an agent that in some way accommodates the symbolic representation of the *beliefs*, *desires*, *intentions* (the BDI model) [10,23] of an agent.

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. It is then also convenient for inter-agent communication messages to have a declarative form, as well. As a consequence, efforts have been make to construct standard protocols (called "agent communication languages" or ACLs) for agent message-passing with declarative content, irrespective of the internal structure of the agent. Normally ACLs specify only the basic intention of a message (typically by making reference to one of a small set of standard "speech acts" [27], which identify an implied expected action associated with a message) and then include the message content as a separate element. Two well-known ACLs that follow this approach are KQML [18] and the Foundation for Intelligent Physical Agents (FIPA) ACL [12]. The content of an agent message can be understood by the participants in an agent conversation if they also share a common *ontology* [15], 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.

Because of the general diversity of opinion concerning the appropriate internal agent structures, the primary development tool support for AOSE currently takes the form of general agent-building toolkits (see the list of tools at [1]), which provide some of the architectural building blocks for the construction of software agents and support for one of the basic ACLs.

### 2.1 An example agent application: distributed information systems

The agent model can be used in connection with a number of widely different specific application areas, which can, for the given cases, entail rather distinctive agent architectures. Thus for some applications mobile agents or a simulation environment of numerous reactive agents may be appropriate. Here, we are interested in "collaborative agent systems" [20], and we examine some agent frameworks associated with this type of system in the context of an application from the area of distributed information systems.

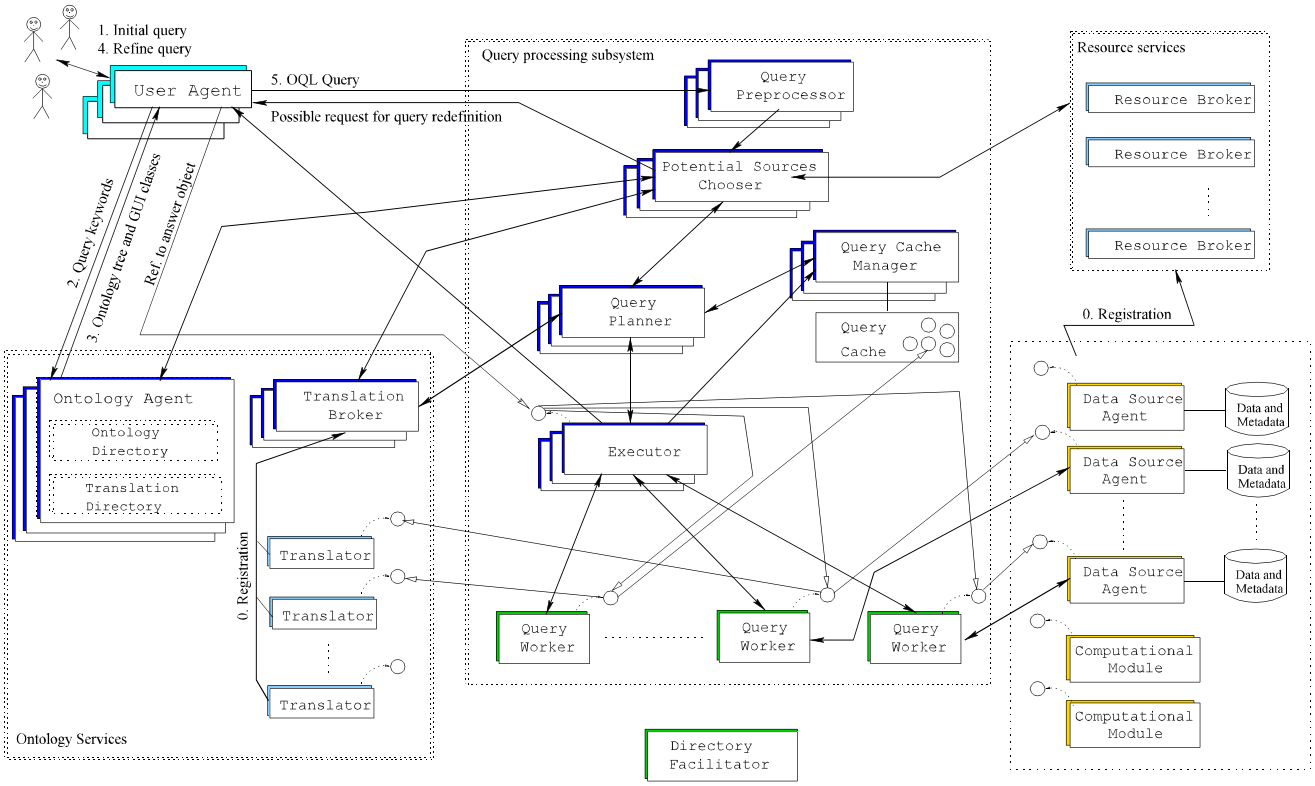


**Figure 1**. **An agent architecture for a distributed information system.**

For the application example, we envision an information system that seeks to provide a degree of integration in connection with accessing a set of heterogeneous, distributed information sources. The information sources may be either data sets, stored according to their specific formats and semantics, or computational modules. An example of the architecture of such a system is shown in Figure 1.

The architecture shown in Figure 1 has five basic components:

- one or more *user agents* which provide an integrated interface for a user to query the information sytem;
- *data source* (or *computational module*) wrapper agents that provide an "transducer" interface between between conventional software modules and the agent system;
- *resource broker* agents that offer services so that other agents can find distributed data sources
- the *query processing subsystem* – a collection of agents that (a) construct a suitable plan for accessing multiple agents in order to respond to a user query and then (b) execute the plan;
- *ontology agents* – service agents that provide information and translation mappings concerning various known ontologies.

In the diagram all of the directed black arrows indicate agent messages.

## 2.2 Agent-oriented architectural frameworks

We have selected two agent-oriented frameworks that have received prominent mention in the agent research community (ZEUS and JADE) and compare them with the framework that is under development by the authors (NZDIS). All three frameworks have been constructed using the Java programming language, come with full source code, and are available free-of-charge (and thus are not commercial products). First, we provide a brief overview of these frameworks.

**2.2.1 The ZEUS Agent-Building Toolkit**. The ZEUS Agent Building Tookit contains a substantial number of tools and utilities that are oriented towards the development of collaborative agent systems [20] – systems of 'specialist' agents that work together to solve a problem that is beyond the capabilities of any one agent. ZEUS is implemented as a collection of Java classes and can be roughly partitioned into three components:

- An agent component library, which forms the building-blocks of individual agents. The classes provided in this library include tools for knowledge storage and representation, agent communication, agent-coordination protocols, and a planning and scheduling system.

- Agent building software, which provides a collection of tools accessed by means of graphical user interfaces (GUIs). These tools are intended to simplify the process of building an agent-based system by guiding developers through the ZEUS agent development methodology [5].
- Agent society visualisation tools – a suite of tools that can be used at runtime to monitor and manage the behaviour of a ZEUS agent society.

The underlying computational paradigm of individual ZEUS agents is based of forward-chaining production systems, which are similar to expert systems that have been developed in the artificial intelligence community and which are based on a flat fact structure (a fact cannot be embedded inside another fact) and a Lisp-like syntax. The inference engine used to evaluate and apply the rules in the system is based on an implementation of the RETE algorithm [11]. Thus we can view ZEUS as essentially a distributed expert system that is based on fact structures and if-then rules (although ZEUS does add many agent-oriented extensions to this).

**2.2.2 JADE – Java Agent DEvelopment framework**. JADE is a development project that aims to provide a Java-based implementation of the agent platform that has been specified by the Foundation for Intelligent Physical Agents (FIPA), an international, non-profit organisation dedicated to promoting "the development of specifications of generic agent technologies that maximise interoperability within and across agent based applications." [12]. This is currently the most mature FIPA-compliant agent platform available and provides a good representation of what the FIPA specifications can offer.

JADE consists of sets of classes that implement an Agent Management System, a Directory Facilitator, and an Agent Communication Channel, all of which are mandatory components of any FIPA-compliant platform. As far as the software developer is concerned, the JADE software provides an object-oriented application programming interface (API). Thus the programmer does not really use any agent-oriented mechanism to interact with those standard components. This is in contrast to ZEUS, where the agent system developer interacts with the ZEUS component by means of the GUI-based tools supplied by the ZEUS system.

**2.2.3 The NZDIS Architecture.** The New Zealand Distributed Information Systems project is a research project conducted by the authors and others at the University of Otago to develop distributed information systems (DIS) software to support the integration of diverse, distributed information sources. The infrastructure of this software is agent-based and designed to conform with the FIPA specifications. A signal characteristic of the NZDIS Architecture is its use of standard object-oriented technology (essentially that specified by the Object Management Group [21]) as a technical basis. For example it makes novel use object-oriented ontologies for specifying the content of agent messages [7,8,9] and employs the OMG's Common Object Request Broker Architecture (CORBA) to provide the foundation for agent communication. This facilitates the use CORBA for the transfer of large datasets in distributed information system applications.

### 2.3 Mapping abstract design to implementation

A key part of the software engineering process is the development of an abstract design and then the mapping of the design into software objects, and we have already indicated above that the agent modelling paradigm should facilitate this process. However none of the three agent platforms investigated here provide significant support in this area. The ZEUS Agent Toolkit documentation does somewhat address this area , though, and suggests the use of a role-model [17] design process. They offer some template role models in the documentation and an example of how to do the implementation. But ZEUS is not predicated on role modelling, and there is no specific support for roles in the framework. The template roles offered by ZEUS are not directly applicable to the distributed information systems application discussed in this paper.

## 3 Agent Communication

The principle agent communication specifications (KQML and FIPA ACL) separate a message into two layers: an inner layer and an outer layer. The outer layer is concerned with transporting and decoding the message, while the inner layer contains the message content itself. For both KQML and FIPA ACL the use of string-based message content raises questions or concerns with respect to efficiency when large or complex objects are to be passed between agents.

### 3.1 ZEUS agent communication

ZEUS messages currently use the string-based KQML agent communication language, and messages are transferred between agents using string-based TCP/IP socket connections. This allows direct peer-to-peer communication, based on an agent's host name and TCP/IP port; these details are obtained from an Agent Name Sever, a default agent in every ZEUS system whose address is well-known. Sockets are a relatively low-level form of communication – ZEUS does not leverage the higher level facilities offered by such tools as CORBA, but rather implements these itself where needed. The content of ZEUS messages is a `fact', represented using a Lisp-like s-expression encoding. Such facts must first be defined in an ontology (Section 4).

### 3.2 JADE agent communication.

JADE uses the FIPA ACL for agent communication, which specifies that message-content for inter-platform communication must be string-based, but that intre-platform communication can employ method calls. Thus for the JADE system agent-builder, messages are simply object-oriented; the system will transparently translate messages to a FIPA-compliant string representation when necessary (*i.e.* when inter-platform message exchange is involved). For message exchange on a local platform, local method call is performed to skip parsing to and from a string representation and Java serialisation is used to pass objects directly between agents. From the developers point of view this approach is convenient. There is, however, no enforcement by the framework to ensure string-based message content for inter-platform communication, which can lead to the violation of the FIPA protocol for agent communication. For intra-platform communication it is permissible to pass custom Java objects, but the same operation will fail when the recipient is located on a remote platform. Thus this is not a fully seamless model, and in our opinion the mechanism should either be more strict (for FIPA conformance) or should be more flexible so that it can cope with arbitrary Java objects on all levels of abstractions. If arbitrary Java objects are allowed and they all are seamlessly translated to string-based representations, then conformance to the ontology would be the only necessary consideration for interoperability.

### 3.3 NZDIS agent communication

The NZDIS architecture also follows the FIPA ACL specification and is quite similar to JADE. Agent communication is achieved by means of string-based discrete messages transported via the CORBA IIOP protocol. Message content is encoded by means of XML. Internally, messages are represented as objects, and simple object manipulation can be performed on the message object. Figure 2 shows how a FIPA-type message can be represented as an object. Again marshalling and unmarshalling procedures are performed for converting between messages and objects.

The NZDIS group is also investigating extending their agent communication protocol beyond the current FIPA specification (but still keeping the "spirit" of FIPA communication) by encoding object information, modelled according to some ontology expressed in the Unified Modelling Language [26] (see discussion below), as an XML document together with an XML schema [7,8,9].

```
(<communicative-act>
  :sender <sender>
  :receiver <receiver>
  :ontology <ontology>
  :language <language>
  :content <content>
  :conversation-id <conv-id>
 etc.
)
```

**(a) FIPA ACL string format**

```
public class Message {
 private String performative;
 private Address sender;
 private Address receiver;
 private String ontology;
 private String language;
 private String content;
 private String conv_id;
//etc
}
```

**(b) Object-oriented abstraction**

**Figure 2.** FIPA ACL messages implemented as objects.

## 4 Ontologies

For communication, agents need to share a common understanding for concepts (terms, objects, relations, etc.) represented in their messages. FIPA ACL specifies a structured form for the fundamental parts of an agent message, but the language for the message content (the "inner message layer" mentioned above) is left up to the implementation. The representation and use of ontologies is recognised an important aspect of agent communication, but it is still a volatile area under exploration and likely to change as new developments appear.

### 4.1 ZEUS ontologies

The domain concepts used in the message content of ZEUS agent communication are modelled as facts – entities used in the rule-base system upon which ZEUS' internal architecture is grounded. Prior to building a ZEUS system all significant concepts within the domain must be defined in an ontology; there is a single system-wide ontology, to which every agent in the system has access, defining all the concepts (facts) that will be used in the system. A concept should be included in the ontology if meaningful discourse between agents can not occur without the agents being aware of the concept. The generation of ontologies is supported by the ZEUS ontology editor, which provides a graphical tool for specifying the domain concepts in the system. The result of the graphical specification process is an *ad hoc* structured

text representation of the ontology (ZEUS toolkit developers have indicated that they intend to use XML for this in the future). This text-based ontology is later used to generate a Lisp-like s-expression representation of the fact, and this format is subsequently used in agent discourse (see Figure 3.

```
BEGIN_FACT_ITEM
 :name Query
 :parent NZDISFact
 BEGIN_ATTRIBUTE_LIST
  BEGIN_ATTRIBUTE_ITEM
   :name query-text
   :type String
   :restriction ""
   :default ""
  END_ATTRIBUTE_ITEM
  BEGIN_ATTRIBUTE_ITEM
   :name language
   :type String
   :restriction ""
   :default "OQL"
  END_ATTRIBUTE_ITEM
 END_ATTRIBUTE_LIST
END_FACT_ITEM
```

**(a) Example of ontology representation in ZEUS.**

```
(query (query-text "Select ... From
... Where ...")
       (language "OQL"))
```

**(b) Example of ZEUS message content.**

**Figure 3. Representation of ontologies and message content in ZEUS.**

## 4.2 JADE ontologies

JADE offers an object-oriented view of ontologies: ontologies in JADE are Java objects and classes. This gives the agent system developer a convenient object-oriented API for the construction and manipulation of ontologies. JADE distinguishes three major elements of the ontology: *Concepts*, *Predicates*, and *Actions*. Concepts represent "objects" (not necessarily just in the object-oriented sense) of discourse; Predicates represent relations between Concepts and constants. Actions represents a function or method call, which might be performed on request.

JADE makes use of some object-oriented techniques via its dependency on and usage of Java interfaces for useful OO idioms, such as polymorphism and multiple inheritance by means of interfaces can be exploited for the Java representations of ontological objects.

In its most basic interpretation an ontology is a set of entities that compose the domain of discourse, and in JADE, each such entity can be represented as an application-dependent Java class. By providing an object-oriented API for ontology construction, JADE tries to make manipulation of symbolic reasoning easier for the application developer. JADE itself does not make any extensions to the original ontology paradigm, as represented for example in OKBC and Ontolingua [4]. This means that the traditional predicate- and logic-based representation is wrapped by an object-oriented Java-based representation, leading to a paradigm mismatch, such that the inference engine then needs to be implemented for symbolic, predicate-based reasoning.

## 4.3 NZDIS ontologies

Ontologies represent the both the domain of discourse and the knowledge about a specific domain in a machine-readable (as well as human-readable) format. Traditionally there are two major paradigms used – the predicate-logic-based approach and the procedural reasoning approach. The former is based on a declarative (logic-based) specification of the relations between entities of the discourse, while the latter is more focussed on the functional description of "how-to", and useful for procedural reasoning. Ontologies are put to use in order to provide interoperability between different agent systems, which have not necessarily been built by the same team or at the same time. By sharing a common understanding of the domain of discourse, along with their objects and relations, separately developed collections of agents can communicate with one another using an ACL without having to know the details of the "API" of each agent targeted for a message (assuming low-level interoperability issues, such as transport, have been resolved).

Since object-oriented technology has many features which are valuable for dealing with large knowledge repositories, such as encapsulation, modularity, well-understood modelling abstractions, and polymorphism, it is increasingly used for the modelling and implementation of information systems. To represent ontological information in terms of standard object-oriented modelling constructs reduces the "impedance mismatch" that otherwise exists when traditional logic-based schemes are used for ontology representation in connection with such systems. Consequently the NZDIS approach is to use UML as the ontology modelling language [7,8].

We observe that it would also be desirable for an agent platform that is built using an OO such as CORBA or Java to provide an object-oriented interface to its Message Transport Service (MTS). Ideally such an MTS would be extensible so that interfaces corresponding to ontologies in UML could be used to construct IDL or Java representations of object diagrams conforming to those ontologies. It would also be useful if the MTS interface included options to

support the transmission of objects either by value or reference. At the present time, the object-oriented modelling tool Rational Rose [25] (and possibly others) provides proprietary mappings from UML models to IDL and Java. Although it would be better to use a standard mapping, for the short term this may be the simplest approach for building an MTS interface that directly supports the construction of object-oriented message content.

## 5 Agent Conversations

For effective agent information exchange simple request-response (client-server) communication is insufficient, and the developer must implement protocols that support more complex conversations. The only formal specification in this area is that of FIPA [12]. The internal (to the agent) state of a conversation is normally represented by a finite state automaton, but Petri nets have also been considered [24].

Some standard communication protocols are provided with the ZEUS toolkit, for example the contract-net protocol. The agent system builder selects one of the standard protocols at design time, and then the given protocol is automatically invoked by the ZEUS Planner at runtime. The specification of new protocols not already stored in the ZEUS system is a non-trivial task with the ZEUS system.

JADE supports most of the predefined FIPA protocols by providing abstract classes that can be subclassed by the developer with behaviour for the transitions in a finite state automaton. The developer must decide the behaviour at design time, because it cannot be changed during runtime. For simple cases where agent participation in a protocol can be specified at design time, this approach is satisfactory. For more dynamic cases, where it is appropriate to establish the response at runtime, it is necessary to employ a specialised subclass as a wrapper to another class which chooses the behaviour dynamically.

Currently the NZDIS architecture leaves the implementation of agent conversations up to the application level. For complex conversations involving several agents, this approach is the most flexible. However, from the perspective of simple patterns, such as fipa-request and fipa-contract protocols, the JADE approach is preferable. In the future the NZDIS architecture will also support those protocols and represent the state of agent conversations by employing coloured Petri nets.

## 6 Summary and conclusions

The ZEUS, JADE, and NZDIS architectures all provide sets of Java classes to support the construction of multi-agent systems that communicate by means of the speech act paradigm. All could be used for the construction of a distributed information system shown in Figure 1.

The ZEUS toolkit supports agent communication using KQML and sockets. It has the best user documentation and GUI-based tools, which greatly facilitate the construction of an agent-based system for the non-programmer. The ZEUS team has gone the farthest in terms of developing specific tools and accessories that support and simplify agent system construction. However, ZEUS developers are rarely concerned with the specifics of agent communication protocols, however, because the ZEUS toolkit provides GUI tools for the developer to construct messages. This appears to represent a fundamental design decision on the part of the ZEUS development team: even though the ZEUS source code is provided, comments have been stripped out and consequently the details of the ZEUS API are shielded from the agent system builder. ZEUS users are, instead, encouraged to access the ZEUS toolkit purely through the GUI-based construction tools, and the communication protocols are automatically invoked by the ZEUS Planner at runtime.

This approach has both advantages and disadvantages. On the positive side, one can say that the ZEUS GUI-based tools are straightforward to use and accelerate the learning curve for new users, especially those who are unfamiliar with agent concepts. The documentation, too, is well presented and offers tutorials and examples. The disadvantage with this approach, however, is that along with this ease-of-use comes a loss of flexibility. Agent-based systems, such as the DIS example considered here, are often complicated and require a certain amount of tuning and customisation to get them right. Since the system builder is discouraged from accessing the Java-based API, he or she is forced to do things in a constrained manner that is offered by the ZEUS control panels. Our assessment is that the full construction of the DIS example shown in Figure 1, which would require the use of sockets for inter-platform communication, would be time-consuming using the ZEUS toolkit.

JADE and NZDIS are similar, in that they both supply sets of Java classes that use FIPA ACL for agent communication. JADE is more oriented to a purely Java-based solution, with even ontologies represented as Java classes and all agents are assumed to be instances of Java classes. This has implications for debugging, message sniffing, and starting and shutting down individual agents. Since everything is in Java, all agents can be run from a single Java virtual machine or via Java Remote Method Invocation. Consequently JADE provides the system builder with a GUI-based Remote Management Agent, which is responsible for starting, controlling, and stopping agents on the platform.

The NZDIS architecture is implemented in Java, but makes no fundamental assumptions about Java. It could have been implemented in C++. It does make assumptions that generic object-oriented technology (as specified by

OMG) is used in the infrastructure. Thus CORBA is employed as the transport layer for agent communication, and UML is used to represent ontological information. At the present time the NZDIS software does not have GUI-based tools that are as convenient as those of ZEUS or JADE, but these are under development. On the other hand, the NZDIS software may be well-suited for the flexible incorporation of ontology manipulation and management tools that will be need for future agent-based systems.

## 7 Acknowledgements

The work in this paper and the NZDIS software is funded by the New Zealand government's Public Good Science Fund. The NZDIS development team consists of Geoff Bush, Dan Carter, Bryce McKinlay, Mariusz Nowostawski, Roy Ward, Stephen Cranefield, and Martin Purvis.

## 8 References

[1] AgentBuilder Web site. Agent construction tools. Available at http://www.agentbuilder.com/AgentTools/index.html.

[2] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE - A FIPA-compliant agent framework. http://sharon.cselt.it/projects/jade, 2000.

[3] Grady Booch. Object Oriented Analysis and Design with Applications. Addison Wesley, 1994.

[4] Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, James P. Rice. Open Knowledge Base Connectivity 2.0.31 –Proposed–, April, 1998. http://ontolingua.stanford.edu/okbc/

[5] Jaron Collins and Divine Ndumu. Zeus methodology documentation. http://www.labs.bt.com/projects/agents/index.htm.

[6] J C Collis, D T Ndumu, H S Nwana, and L C Lee. The ZEUS agent building toolkit. BT Technology Journal, 16(3), July 1998.

[7] 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. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-23/cranefield-ijcai99-iii.pdf.

[8] S. Cranefield and M. Purvis. Extending agent messaging to enable OO information exchange. In R. Trappl, editor, Proceedings of the 2nd International Symposium `From Agent Theory to Agent Implementation' (AT2AI-2) at the 5th European Meeting on Cybernetics and Systems Research (EMCSR 2000), pages 573--578, Vienna, April 2000. Austrian Society for Cybernetic Studies. Published under the title `Cybernetics and Systems 2000'.

[9] Stephen Cranefield, Martin Purvis, and Mariusz Nowostawski. Is it an ontology, a meta-model or an abstract syntax? Modelling FIPA agent communication. In Proceedings of the Workshop on Applications of Ontologies and Problem Solving Methods, 14th European Conference on Artificial Intelligence, pages 16.1-16.4, 2000.

[10] Jacques Ferber. Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence. Addison-Wesley, 1999.

[11] Charles L. Forgy. Rete: A fast algorithm for the many pattern many object pattern match problem. Artificial Intelligence 19 (1982), 17-37.

[12] Foundation For Intelligent Physical Agents (FIPA) web site. Located at http://www.fipa.org/.

[13] Stan Franklin and Art Graesser. Is it an agent, or just a program? : A taxonomy for autonomous agents. In Jorg P. Muller, Michael J. Wooldridge, and Nicholas R. Jennings, editors, Intelligent Agents III. P}roceedings of the Third International Workshop on Agent Theories, Architectures and Languages, volume 1193 of Lecture Notes in Computer Science. Springer, 1996.

[14] Michael R. Genesereth and Steven P. Ketchpel. Software agents. Communications of the ACM, 37(7):48--53, July 1994.

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

[16] Nicholas R. Jennings and Michael Wooldridge. Agent-oriented software engineering. In J. Bradshaw, editor, Handbook of Agent Technology. AAAI/MIT Press, 2000.

[17] E. A. Kendall. Agent roles and role models: New abstractions for multiagent system analysis and design. In International Workshop on Intelligent Agents in Information and Process Management, Germany, September 1998.

[18] Yannis Labrou and Tim Finin, A Proposal for a new KQML Specification. TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, February 1997.

[19] Hyacinth Nwana, Divine Ndumu, Lyndon Lee, and Jaron Collis. ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems. Applied Artifical Intelligence Journal, 13(1):129-186, 1999.

[20] Hyacinth S Nwana. Software agents: An overview. Knowledge Engineering Review, 11(3):1-40, September 1996.

[21] Object Management Group (OMG) web site. Located at www.omg.org.

[22] Martin Purvis, Stephen Cranefield, Geoff Bush, Daniel Carter, Bryce McKinlay, Mariusz Nowostawski, and Roy Ward. The NZDIS Project: an Agent-based Distributed Information Systems Architecture. In R.H. Sprague Jr., editor, CDROM Proceedings of the Hawaii International Conference on System Sciences (HICSS-33). IEEE Computer Society Press, 2000.

[23] Anand S. Rao and Michael P. Georgeff. BDI Agents: From Theory to Practice. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, USA, June 1995.

[24] Wolfgang Reisig. A Primer in Petri Net Design. Springer-Verlag, 1992. based on a German edition.

[25] Rational web site. Located at www.rational.com.

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

[27] John R. Searle. Speech Acts. Cambridge University Press, Cambridge, 1969.

[28] Michael Wooldridge. Agent-Based Software Engineering. IEE Proc Software Engineering, 144:26--37, 1997.