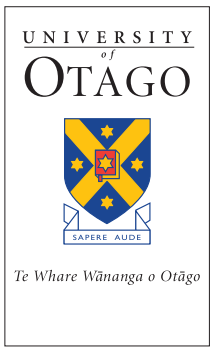

A Multi-Level Approach and Infrastructure for Agent-Oriented Software Development

Mariusz Nowostawski
Geoff Bush
Martin Purvis
Stephen Cranefield

The Information Science Discussion Paper Series

Number 2001/06
March 2001
ISSN 1177-455X

University of Otago

Department of Information Science

The Department of Information Science is one of six departments that make up the School of Business at the University of Otago. The department offers courses of study leading to a major in Information Science within the BCom, BA and BSc degrees. In addition to undergraduate teaching, the department is also strongly involved in postgraduate research programmes leading to MCom, MA, MSc and PhD degrees. Research projects in spatial information processing, connectionist-based information systems, software engineering and software development, information engineering and database, software metrics, distributed information systems, multimedia information systems and information systems security are particularly well supported.

The views expressed in this paper are not necessarily those of the department as a whole. The accuracy of the information presented in this paper is the sole responsibility of the authors.

Correspondence

This paper represents work to date and may not necessarily form the basis for the authors' final conclusions relating to this topic. It is likely, however, that the paper will appear in some form in a journal or in conference proceedings in the near future. The authors would be pleased to receive correspondence in connection with any of the issues raised in this paper, or for subsequent publication details. Please write directly to the authors at the address provided below. (Details of final journal/conference publication venues for these papers are also provided on the Department's publications web pages: http://www.otago.ac.nz/informationscience/pubs/). Any other correspondence concerning the Series should be sent to the DPS Coordinator.

Department of Information Science
University of Otago
P O Box 56
Dunedin
NEW ZEALAND

Fax: +64 3 479 8311
email: dps@infoscience.otago.ac.nz
www: http://www.otago.ac.nz/informationscience/

A Multi-Level Approach and Infrastructure for Agent-Oriented Software Development

Mariusz Nowostawski
Geoff Bush
Martin Purvis
Stephen Cranefield
Department of Information Science
University of Otago
PO Box 56, Dunedin, New Zealand
Phone: (64 3) 479 8318, Fax: (64 3) 479 8311
{mnowostawski,gbush,mpurvis,scranefield}@infoscience.otago.ac.nz

Keywords
agents, multi-agent system, multi-agent platform scalability

ABSTRACT
An architecture, and the accompanying infrastructural support, for agent-based software developement is described which supports the use of agent-oriented ideas at multiple levels of abstraction. At the lowest level are micro-agents, which are robust and efficient implementations of streamlined agents that can be used for many conventional programming tasks. Agents with more sophisticated functionality can be constructed by combining these micro-agents into more complicated agents. Consequently the system supports the consistent use of agent-based ideas throughout the software engineering process, since higher level agents may be hierarchically refined into more detailed agent implementations. We outline how micro-agents are implemented in Java and how they have been used to construct the Opal framework for the construction of more complex agents that are based on the FIPA specifications.

1. INTRODUCTION
It has been argued that the use of agents, particularly intelligent agents, can be useful for the modelling and construction of complex distributed information systems [9, 10]. Implicit in these arguments is the basic notion that the use of agents supports scalability; that as system complexity scales upward to highly distributed and dynamic environments, the use of agents will be essential for the successful operation and maintenance of these systems. Three important techniques have been identified for dealing with the complexity of large systems [2], as follows:

Decomposition – essentially the notion of "divide and conquer".

Abstraction – encapsulating and hiding unimportant details by defining modelling "chunks" that emphasise a few important details and suppress others. As a model is gradually refined, the abstractions associated with specific model components may be changed.

Organisation – the process of identifying and managing the interoperation of complex components.

It has been shown that it is plausible to consider agents to be effective entities for use in connection with all three of these techniques [11]. The consistent use of agents throughout the software development process in this manner is often termed agent-based software engineering.

For agent-based software engineering to hold in practice, it is necessary that there be a suitable agent-building infrastructure available for software engineers so that they can employ agent constructs in the various ways that they are envisioned. In particular to support decomposition, it is necessary to be able to use agents at various levels of modelling detail and refinement. In this way, a designer should be able to consider a system at any level of detail desired and think of that system in terms of agents (each of which could, in principle, be composed of smaller, internal agents). In addition, the use of agent entities should not impose an unsatisfactory performance penalty on the designer who elects to use them. This means that agent-based software development should support scalability if it allows agents to be used at various levels of abstraction and, at the same time, supports the efficient execution of agent interoperation.

At the present time, however, there does not appear to be any agent-building toolkits or infrastructural support that completely meets these two basic demands and enables agent-based software engineering in the ideal manner envisioned. The main publicly available agent-building toolkits discussed in the literature [1, 13, 14] are focussed on the construction of systems whose individual agents have a relatively coarse degree of granularity and which are not intended to be refined into smaller agents. Coarse grained agents typically require significant amounts of computing machinery. This may involve machinery to support interoperable and semantically rich string-based communication models [3], or to support high-level cognitive modelling, for example by using the BDI model [5]. While such powerful machinery may be valuable for certain situations, its use for building smaller components, such as graphical user inter-

face applications, may be irrelevant and impractical. MadKit [6, 7] is a fine-grained agent-building toolkit that includes some ideas similar to those expressed here, although to our understanding it has some architectural differences and has not been used as the basis for a coarse-grained agent-building infrastructure, such as the Otago agent framework, Opal.

Existing agent-building toolkit systems are quite complex in their own right. They are primarily built using object-oriented software technology, as opposed to agent-based technology. Consequently they do not have the notion of agents built-in to their machinery. The present paper discusses an approach for building complex software systems, whereby agents can be used at multiple levels of modelling and operational detail. With this approach, both fine-grained and coarse-grained agents can be employed where desired. In particular, one can use the approach to design an infrastructural toolkit that is to be used to support the general construction of agent-based systems. We discuss our work in this connection by describing the design and development of Opal, an agent-building framework that has been designed using this multi-level agent-based approach.

2. MICRO-AGENT KERNEL

The overall goal of our approach is to use the notion of agency to model and build systems at any level of abstraction. This is achieved by instantiating the idea of an agent at the lowest level of operation so that it is practically realisable for efficient code execution but still retains enough of the features of *agenthood* that it can still be considered to be an agent for modelling and design purposes. In order to facilitate the following discussion, we identify some terms that we will use to describe various aspects of our architecture:

Agent – a persistent entity deployed on a multi-agent system. This can be considered to be an actor that plays one or more particular roles in a society of agents.

Micro-agent – a particular type of agent that represents the lowest and most primitive level of agent instantiation

Role – a specification of a cohesive set of behaviours, functions or services in the multi-agent society. Roles may be played by one or more agents in an agent system. Each agent playing a role may take a different approach to providing the role's services.

Responsive Agent – an agent which does not control its own thread of execution, but simply reacts to the stimuli from the outside. Upon activation this agent can nevertheless perform deliberative computations, engage in social interactions, commit to or refuse to accept a particular goal given to it, or perform or refuse to perform a particular function assigned to it.

Autonomous Agent – an agent which controls its own thread of execution. It actively pursues and maintains its goals, stimulates other agents, including responsive agents, and may control and manipulate other agents (by playing the Group role).

Agent Group – a role that provides an environment in which other agents (sub-agents) exist. This role is used for registration and discovery in a society of agents. It provides a mechanism for agents to locate each other based on the role they are playing, a role-based "yellow pages" service for micro-agents. Agents can register with more than one agent playing the group role.

Agent Manager – a special role used for managing a society of agents. This role deploys, starts up, controls, and shuts down agents, usually in a context of a specific group.

Agent System – any persistent society of agents. An agent system could have multiple groups, and specific groups or agents could be introduced, deployed or redeployed, or killed at various times during the life of the system. An agent can be decomposed into smaller sub-agents that work together. When this happens, we can think of the original agent as having become an agent system.

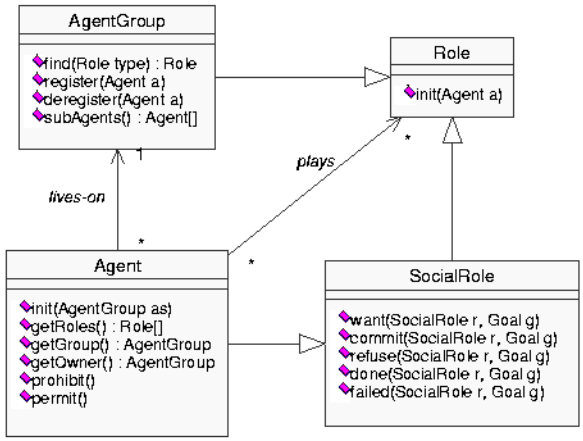

Figure 1: MicroAgent System Design

A UML diagram of the key entities in the micro-agent system is shown in Figure 1. There are two base elements in the micro-agent system, namely *agents* and *roles*. Agents represent actors in the system that can play one or more roles. Roles are interface specifications of a cohesive set of services that may be provided by one or more agents, and each agent may take a different approach to providing the role's services in order to implement that role. An agent group is a role that provides an environment in which other agents (sub-agents) exist. Because an agent group is a role, some agents can contain other agents. This can be used as a hierarchical decomposition method for cases where it is logical to design agents in terms of a set of sub-agents. All agents belong to at least one agent group (an owner) that they live in; however the top-level agent group does not have an owner, and this is effectively a recursion termination condition.

2.1 Micro-agents

Micro-agents exist at the lowest-level of agent-based abstraction in this architecture. In order to be efficient at this

fine-grained level they do not have all of the qualities often attributed to typical, more coarsely-grained agents.

Those agents that exist at the higher level of abstraction, such as those based on the Foundation for Intelligent Physical Agents (FIPA)[4] specifications, typically engage in agent communication using the declarative representation for their messages that is based on speech-act theory[15]. Micro-agents, on the other hand, employ a simpler form of agent communication and, in addition, have more limited flexibility when compared to higher level agents.

Coarser-grained agents, such as FIPA-based agents, may make reference to ontologies (which characterize the terms and relationships mentioned in their messages), and they may reason about such ontologies or even adopt new ontologies for new agent conversations. Micro-agents, on the other hand, do not have ontologies in that sense but can be thought of as having an implicit system-level ontology that cannot be changed or reasoned about.

Micro-agents, being the closest entities to the machine platform, must be implemented on a specially designed micro-kernel. For example, for the Opal system, which is implemented in Java, the micro-agents are implemented by extending defined packaging and framework constraints, and they communicate via method calls. As a consequence, the micro-agents behave in strictly-defined and predictable ways and do not carry out runtime reasoning. Additionally, some micro-agents are responsive agents and do not own their own thread or threads of control.

Agents may be composed of any number of other agents or micro-agents. Non-primitive micro-agents are composed only of micro-agents. The same agent-based modelling approaches apply in the same way to both coarse-grained agents and to micro-agents – the same design methodology, role-oriented and society-oriented techniques apply equally to coarse-grained agents as to micro-agents. Agent-oriented decomposition and role-based modelling are independent of the deployment scale. All the roles can potentially be played by micro-agents or coarser-grained agents with a similar result. However the key advantage of micro-agents is that for small scale systems they will radically out-perform coarse-grained agents at runtime.

2.2 Communication

For traditional communication in multi-agent, peer-to-peer asynchronous message passing with a formal agent communication language is used [3]. A message is embedded inside an envelope, which contains routing information, identification of the receiver and the sender, and the content of the message. The message content, which can be expressed in one of a number of possible content languages, makes references to terms formally defined in an ontology, and can be sent in the context of ongoing an interaction protocol [4] or conversation. A conversation in this context is a sequence of message exchanges which may span multiple interaction protocols and multiple agents.

In order to maintain the spirit of inter-agent semantic communication as expressed in speech-act theory, micro-agents have been designed so that they communicate using messages of the following types: directives, assertives, expressives, commissives, permissives and prohibitives [16]. This does not represent a true implementation of natural language communication, but instead uses names derived from the discipline. The intention is to enable the developer to employ a mental model of language-based agent communication when micro-agents are used.

The social aspects of the agent interactions are captured in the SocialRole (see Figure 1), which contains all the primitive type of communicative acts we have discussed above. The communicative acts (performatives) are implemented as simple method calls with a special argument list. The performative is represented as the method name itself, while the sender is identified by the first parameter and the message content is represented by the goal argument. Goals together with Roles can be designed in UML, which then maps directly to the implementation via classes and interfaces.

2.3 Implementation

The current implementation of the micro-agents and the kernel which supports them is written in the Java programming language. This elevates existing object-oriented design patterns up to a useful a agent-oriented abstraction level. However the Java programming language imposes some constraints of what can be done, and consequently some of the patterns that are desirable from the agent-based perspective cannot be implemented in a straightforward and efficient manner. Some of the most notable problems are:

- an agent dynamically playing different roles maps to a class implementing that can play several differing interfaces at runtime. This is not possible without the inefficient dynamic proxy mechanism introduced in Java 1.3
- an agent will often need to identify the sender of a message, but using Java it is impossible to identify the caller of a method without using an additional formal argument in the method
- Java reflection, needed to discover runtime agent capabilities, is inefficient

Despite these constraints, we believe that there are good reasons to implement an agent based system in the Java language, and we also believe that we have successfully implemented most of the features which lie at the heart of agent-oriented software engineering. The intention has been to provide a micro-agent and kernel framework which is as efficient as possible so that the micro-agent message operation involves little more overhead than a normal virtual method call in Java.

2.4 Hello World Example

To demonstrate the use of micro-agents from the Java developers perspective we have developed a simple "Hello World" example. The Hello World agent system consists of four agents, the User agent, HelloPrinter, Dumper, and DataCollector, which are organized into two groups, as shown in Figure 2. The User agent is a social agent which wants the Hello World data to be printed. HelloPrinter is a social

agent which can achieve the goal of printing `Hello World`. Dumper is a responsive agent which can dump data to a system output port, and DataCollector is a simple responsive agent which can provide data, and in this scenario it provides the static "Hello World" string.

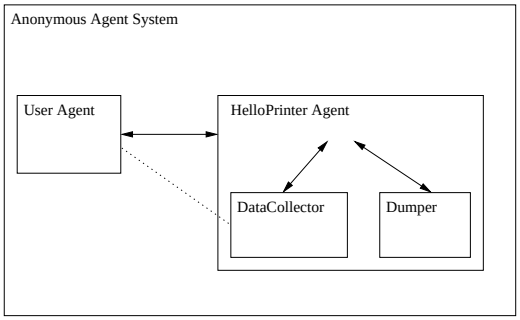

Figure 2: Hello World example

By separating these four different aspects of the system, different agents implementing the same roles can be plugged in without affecting the rest of the system. This will allow us to have the following collection of agents,

- Dumper agents, which dump to the screen, to a file, or to some GUI-based output
- several DataCollector agents, one collecting data from the user sitting in front of the console, others from a file, telephone or GUI application

These extensions could be added during runtime by plugging in different agents dynamically, without affecting the rest of the system.

Setting up the Hello World System in Java could look like this:

```
public static void main(String[] args){

  // the process of loading and creating new agents
  // performs registration and initialisation
  Agent hello = SystemAgentLoader.
                    loadAgent(new HelloPrinterImpl());
  Agent user = SystemAgentLoader.
                    loadAgent(new UserAgent());

  // add sub agents to hello agent
  Group group = hello.getGroup();
  AgentLoader loader = group.getAgentLoader();
  loader.loadAgent(new DumperAgent());
  loader.loadAgent(new DataCollectingAgent());

  // go!
  Goal g = HelloPrintedGoal.instance();
  hello.want(user, g);
}
```

Implementation	Time (ms)
Java	340
Micro-agents	440
JADE	140,000

Figure 3: 10,000 Hello World Iterations

```
// HelloPrinter Role Implementation
public class HelloPrinterImpl
  extends DefaultSocialRoleImpl
  implements HelloPrinter {

  public void want(Agent a, HelloPrintedGoal g){
    // machinery to achieve goal
  }
}
```

The code above shows an example of a top level main method which sets up agents in different groups. One group is the top level agent group containing the User agent and HelloPrinter agent. The second group is contained within the HelloPrinter agent—it controls and manipulates a DataCollector agent and a Dumper agent.

To demonstrate the merit of the micro agent approach three different Hello World implementations have been developed and timed. The first implementation was a simple call to the Java method `System.out.println("Hello World");` and the second the micro-agent implementation as described above. The third was a implementation using the the same agent-decomposition as the micro-agent example, but with the JADE agent toolkit [1]. The results of timing 10,000 Hello World requests are shown in Figure 3.

The JADE agents actually did not perform any message processing or parsing during the tests, and there was no data conversion performed for a given transport. It ran on single virtual machine and all message passing was done via a simple Java RMI mechanisms. This shows that coarse grained agents may in some cases be 300 times slower than our micro-agent implementation. That confirms that FIPA-like agents are not likely to be suitable for fined-grained, simple and efficient system components.

3. MICRO-AGENT APPLICATIONS

In this section we demonstrate how the micro-agent system can be used to build applications using an agent-oriented approach. An agent-based model of a robot is discussed later in this section. This is an example of an application where micro-agents are required because of the scale of the problem. In Section 4, the development of Opal using micro-agents is discussed. This is an example of how micro-agents might be used to develop a non-trivial system.

An initial agent-oriented model of a robot is shown in Figure 4. This model identifies the role of an robot, interacting with an environment and receiving instructions from a human operator. It would be possible to go from this model directly to an implementation of the robot as a coarse-grained agent. This implementation would be monolithic and it is likely that further design, possibly using object-oriented method-

ologies, would be required to provide a decomposition from this high-level model to implementation level components.

Figure 4: Role oriented decomposition A

Figure 5 shows a more detailed decomposition of the robot. Several autonomous, concurrently running and communicating sub-roles are identified, a sensory processor, task scheduler and meta-level processor. Three independent effectors are also identified, the arm and the left and right wheels. These components may themselves be further decomposed, the figure shows only the decomposition of the task scheduler. Three reactive components of the task scheduler are identified, a command analyser, action planner and action executor. An example scenario could be the operator asking the robot to move to a particular location. This directive would be processed by the command analyzer, then the planner would create a plan to reach the location, then the executor sends instructions to the wheels and arm to move to the location.

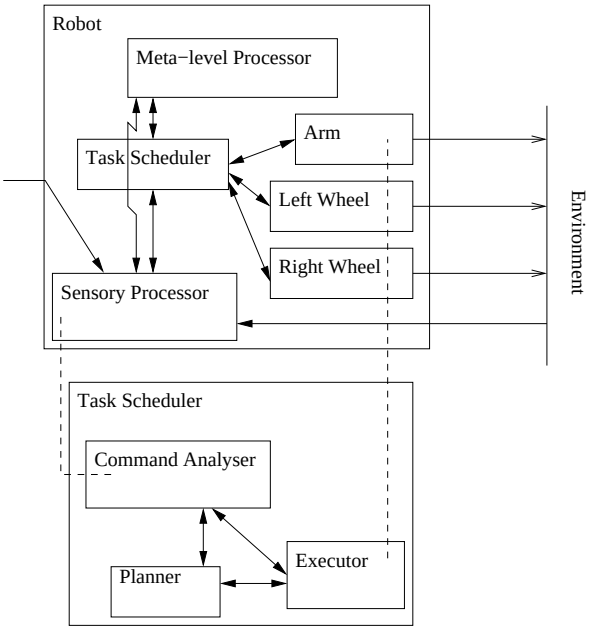

Figure 5: Role oriented decomposition B

The level of detail expressed in this model would not be practical to implement using traditional coarse-grained agents—as the hello world example has shown this would be too inefficent. Although the initial model for the coarse-grained implementation was agent-oriented further refinement of such a model using traditional agent development technologies would likely require alternative design methods to be used. On the other hand, using the micro-agent approach described in this paper a more-detailed level of design can be achieved with agent modelling employed all the way down to the implementation level.

4. OPAL AGENT FRAMEWORK

As discussed in section 2, micro-agents are presented without much of the coarse-grained machinery often associated with "intelligent" agents. While micro-agents have been found appropriate for closed systems, such as those discussed in the previous section, many interesting agent-related research areas involve open systems, where the agents are typically coarse-grained, heterogeneous entities that make use of different languages and ontologies.

The Foundation for Intelligent Physical Agents (FIPA) is developing open specifications for such coarse-grained agent systems [4]. A key part of this work has been the FIPA Abstract Agent Architecture (FAA), which is an abstract specification of the infrastructure necessary to provide a suitable platform on which such agents can exist.

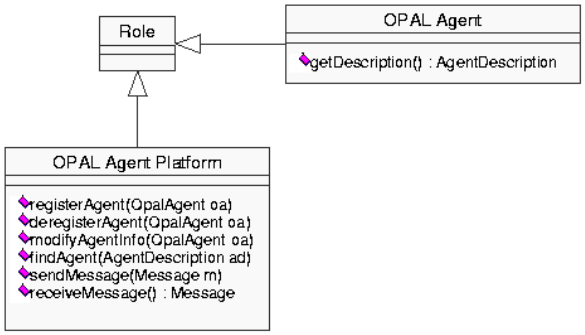

Figure 6: Micro-agent Roles used by Opal

This section discusses the Otago Agent Platform (Opal) which provides a correct concrete instantiation of the FAA, as well as other tools and utilities useful for the development of agent-based systems. In a sense, Opal can be considered to be a scaled-up version of the micro-agent system. There is considerable amount of infrastructure specified by FIPA for coarse grained agents that does not exist in the micro-agent system as we have described it. To provide this additional capabilities for FIPA-type agents, specialised micro-agents need to be introduced. The Opal system is therefore designed to be combination of these specialised micro-agents, that together provide for FIPA functionality.

4.1 Opal Architecture

An important concept of the FAA is the idea of an Agent Platform (AP), this provides environmental support and the basic services for the agents deployed on it and it also provides a directory service to agents outside the AP. The Opal AP is implemented as a micro-agent playing the Agent Platform Role (see Figure 6). The key services that the AP provides are inter-platform message transport via the Message Transport System (MTS), agent management and a white-pages directory via the Agent Management System (AMS), and yellow-pages directory services via the Directory Facilitator (DF). These three logical capability sets are implemented in the Opal AP as separate micro-agents.

The AP Role implementation does not itself perform the bulk of the processing required for the actions of the AP Role (see Figure 6), rather its tasks are delegated to the

three contained micro-agents. To register an agent, the MTS needs to know about the agent so it can receive messages for it, the AMS needs to add the agent to its white-pages directory and the DF needs to add the agent to its yellow-pages directory.

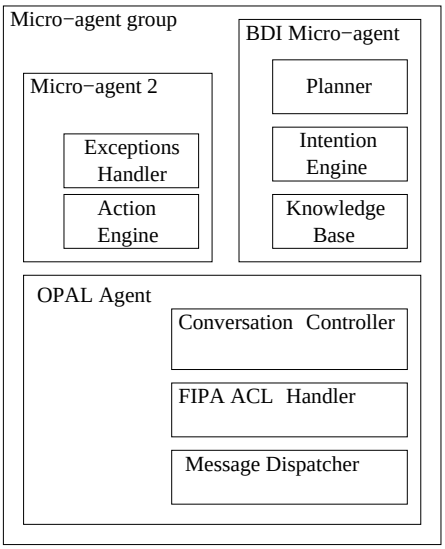

Figure 7: Opal's FIPA Platform and Opal Agents

It is convenient for developers to specify the receiver of a message using a simple name. For the platform to send the message the transport-level address, which might be a CORBA IOR, Java RMI address or even an email address, must first be found using the AMS, then the MTS is used to send the message. When performing an action the Agent Platform agent has the responsibility for ensuring that the correct sequence of sub-actions gets performed, but does not perform any of these sub-actions itself.

Aside from the micro-agent role representing the agent platform, an Opal system needs to contain the higher-level coarse-grained FIPA agents that exist on the Agent Platform. The micro-agent roles representing FIPA agents can contain a variety of sub-agent roles. An agent that contains no sub-agents is provided with only the ability to send and receive messages. Figure 7 shows a single FIPA agent that uses a conversation controller micro-agent role to keep track of conversations that the agent is involved in. The conversation controller requires that some micro-agent exists that is able to play the message dispatcher role. Another FIPA agent may require the Belief-Desire-Intention micro-agent role to enable it to be developed using the BDI model.

A complete Opal Agent System with associated FIPA-specified services is depicted in Figure 8. Individual Opal Agents of the type shown in Figure 7 can all access an Opal Platform Agent. The Opal Platform Agent contains individual micro-agents that implement the FIPA-specified services of the Message Transport System (MTS), the Directory Facilitator (DF), and the Agent Management System (AMS).

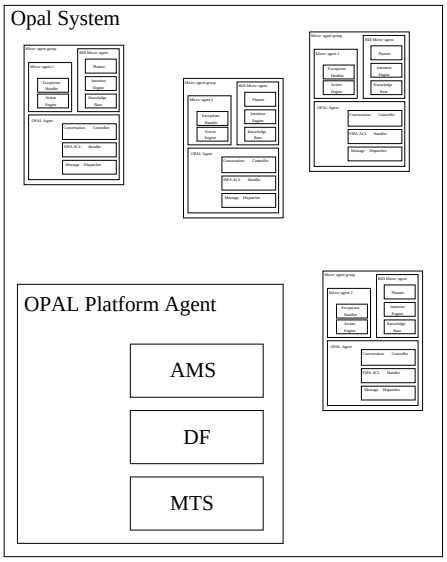

Figure 8: Example of Opal System

4.2 Message Transport and Dispatching

Transport services are the lowest level services provided usually by the Agent building toolkit or framework, to enable inter-agent communication and message passing. Most of the existing agent frameworks suffer however from different levels of incompatibilities because they all provide their own customized implementation of message transport and dispatching. Opal differs from its predecessors agent framework in this respect, by employing the an emerging industry standard for lower-level standard services, JAS.

JAS, which stands for Java Agent Services, is an effort to define an industry standard specification and API for the development of network agent and service architectures (for more details see www.java-agent.org)[1]. Opal employs a modular implementation approach to transport services not offered currently by any other agent framework. Transport services are pluggable and thus new transport implementation can be seamlessly integrated into the platform when needed. By default, Opal provides implementations of the two main transport protocols used by FIPA platforms, namely: FIPA JAS RMI-based and FIPA2000 IIOP-based transports, which can be plugged in and used in any JAS -complient platform.

4.3 Interaction Manager

Opal implements several standard FIPA interaction protocols, and there is a special entity which handles dispatching and switching the state of a conversation based on the type and properties of received and sent messages. We refer to this module as *interaction manager*, as distinguished from *conversation manager* which will be described in the following section.

[1]JAS does not have a formal specification yet, but there exists a preliminary proposed API set and reference implementation which Opal currently follows. Opal development efforts will continue to follow closely JAS efforts in order to maintain compatibility with it and ensure up-to-date an implementation

The basic abstract role is an Interaction Tracker, which is specialized by concrete roles from all predefined FIPA interaction protocols, like FIPARequestTracker, FIPAQueryTracker, etc. Other agents can use the concrete instantiations of those roles, which are played by the dynamically created agents. The actual InteractionManager is responsible for starting, monitoring and controlling all those instantiations.

The system is flexible enough to cope with different aspects during runtime, for example a specialized Interaction Manager could be used to monitor and reschedule priorities of the ongoing interactions, keep track of exceptions and delegate them to specialised units elsewhere in the system.

4.4 Conversation Manager

A conversation is an ongoing sequences of messages which can span multiple agents and multiple interaction protocols. There are many aspects which need to be addressed, including tracking individual interaction protocols and their violations, keeping track on the context and the subject, timing out late responses, allocating resources to more important tasks, etc. To manage the complexity of these conversations, a special infrastructural component must be modelled, implemented and deployed.

Our approach to this dynamic and complex task is to decompose the overall architecture into decoupled and relatively self-contained modules. This employs the same design philosophy that we have used in other parts of the system, whereby a "divide-and-conquer" approach is applied to the separation of concerns so that tasks are partitioned into small tractable modules. This enables us to benefit from all the positive features of agent-oriented decomposition and decoupling. All the elements are pluggable and there is no single machinery hardwired to any other one. In case of failure, it gives the flexibility and capability to switch to other working resources. In case of specialization and system evolution, it gives a natural separation of concerns: a change in one part of the system will introduce only minimal changes to other parts of the system.

We employ a three-layer architecture for modelling conversations, which expands on previous approaches that employs two abstractions of *protocols* and *conversations*, by adding an additional notion, that of *policy* (or strategy). The policy layer guides the participating agents during the course of a conversation and can be used to deal with conversational components that are directed to be *about* the current conversation in progress or that can serve to reroute the current conversation in a new direction. This new additional level helps to keep conversation-specific logic close to the conversation models, which improves encapsulation and helps conversation debugging and the verification process.

All three layers are modelled as specialized roles that are played by instantiated agents. Usually the Policy Manager will monitor all existing conversations and react to some meta-level heuristics. There are a great many enhancement possibilities, and much of the reasoning, for obvious reasons, is not actually implemented in Opal, but left to the specific domain and system developer to be implemented. The main purpose is to provide scalable and flexible infrastructure for developing complex strategies in the context of inter-agent communication.

4.5 Deliberation and Task Scheduling

A high-level part of the Opal architecture is a set of standard agents to perform scheduling and planning for other agents, however this element of the Opal framework is in the early experimental stages.

All micro-agents and ordinary agents are inherently goal-driven and role-oriented. The micro-agent platform built on top of micro-kernel provides set of standard agent services to perform hierarchical goal reasoning and planning, following Procedural Reasoning Systems [8, 12] traditions. This is currently in the design phase and will be described in future publication.

Developers can use all the lower level machinery to perform simple task and goal decomposition and program appropriate scenarios to be used as a role implementations. Further, the developer can provide some meta-level reasoning agents and capabilities which will typically span most of the existing system components.

The principle idea is not to cope with the big task in a single centralized place (like inside a big coarse-grained agent), but rather to reuse different bits and pieces distributed throughout the system. This is where the entire system can benefit from the micro-agent infrastructure. One of the main goals of Opal is to provide uniform modelling abstractions and operational techniques, which can be used for dealing with different scales and different granularity of components. A loosely coupled, but interconnected network of such components, an agent system, can in a flexible and robust way solve different complex systems.

5. DISCUSSION

This paper has presented an approach to agent-oriented software development that seeks to employ the notion of agent modelling at multiple levels of abstraction. With this approach, a high-level abstract representation of a system can be successively refined to lower and more detailed levels of implementation, so that agent-oriented concepts can be used throughout the software development process. In support of this approach, we have developed an agent-based infrastructure that uses base level agents, which we have called micro-agents, as the fundamental building blocks for the design and construction of agent-based systems. The micro-agent and supporting kernel implementations that we have developed in Java enable the software engineer to develop agent-based systems and components that are much more efficient than those developed by conventional coarse-grained agent technology.

Using the micro-agent-based infrastructure, we have built the Opal agent-based framework that offers support for the development higher-level agents that conform to the FIPA specifications. With Opal it will be possible to design FIPA-based agent systems and also employ agent-based components for virtually all aspects of a software system, including finer-grained components that are not normally implemented in terms of agent constructs for reasons of efficiency. It is our contention that the approach and infrastructure de-

scribe here supports scalable agent-based solutions, because one can employ agent-based concepts over a wider range of software engineering activities and still produce efficient software implementations.

We are continuing to add more services and functionality to the Opal agent framework and will make the source code for this infrastructure publicly available in the near future.

6. REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa. JADE - A FIPA-compliant agent framework. http://sharon.cselt.it/projects/jade, 2000.

[2] G. Booch. *Object Oriented Analysis and Design with Applications*. Addison Wesley, 1994.

[3] FIPA. FIPA Spec 2 - 1999. Agent Communication Language. Draft, Version 0.1, http://www.fipa.org/specifications/index.html, 16 April 1999.

[4] FIPA. Foundation For Intelligent Physical Agents (FIPA). FIPA 2000 specifications. http://www.fipa.org/specifications/index.html, 2000.

[5] M. Georgeff and A. S. Rao. A Profile of the Australian Artificial Insititute. *IEEE Expert*, pages 89–92, December 1996.

[6] O. Gutknecht and J. Ferber. Madkit: Organizing heterogeneity with groups in a platform for multiple multi-agent systems, December 1997. Technical Report 97188, LIRMM, 161, rue Ada - Montpellier - France.

[7] O. Gutknecht and J. Ferber. MadKit Agent Platform Architecture, May 2000. Technical Report R.R.LIRMM 000xx, 161, rue Ada - Montpellier - France.

[8] F. Ingrand and M. Georgeff. Procedural Reasoning System, User Guide. 1991. Australian Artificial Intelligence Institute, 1 Grattan Street, Carlton, Victoria 3053, Australia.

[9] N. R. Jennings. Agent-oriented software engineering. In *Proceedings of the 12th International Conference on Industrial and Engineering Applications of AI*, pages 4–10, 1999.

[10] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[11] N. R. Jennings and M. Wooldridge. Agent-oriented software engineering. In J. Bradshaw, editor, *Handbook of Agent Technology*. AAAI/MIT Press, 2000.

[12] J. Lee, M. J. Huber, P. G. Kenny, and E. H. Durfee. Um-PRS: An implementation of the procedural reasoning system for multirobot applications. In *Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS)*, pages 842–849, Houston, Texas, 1994.

[13] H. Nwana, D. Ndumu, L. Lee, and J. Collis. ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems. *Applied Artifical Intelligence Journal*, 13(1):129–186, 1999.

[14] S. Poslad, P. Buckle, and R. Hadingham. The FIPA-OS agent platform: Open source for open standards. In *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, pages 355–368, 2000.

[15] J. R. Searl. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press: Cambridge, England, 1969.

[16] M. P. Singh. Agent communication languages: Rethinking principles. In *IEEE Computer, 0018-9162*, pages 40–47. December 1998.