MAGENTA Technology: Multi-Agent Systems for Industrial Logistics

Jon Himoff[1] Petr Skobelev[1] Michael Wooldridge[2]

[1] MAGENTA Technology
33 Glasshouse Street
London W1B 5DG, UK
jon.himoff@magenta-technology.com
skobelev@magenta-technology.ru

[2] Dept of Computer Science
University of Liverpool
Liverpool L69 7ZF, UK
mjw@csc.liv.ac.uk

ABSTRACT

We introduce MAGENTA's commercial multi-agent systems technology, and illustrate its practical use by describing a field-tested application in the area of logistics/scheduling. MAGENTA technology provides two integrated toolsets for building industrial-strength systems: the *Ontology Management Toolkit* (which enables designers to capture the concepts and interrelationships between concepts in an application), and the *Virtual Marketplace Engine* (the platform supporting agent interaction in MAGENTA's technology); in addition, run-time visualisation and monitoring tools are provided for debugging systems. The application we describe is a field-tested scheduling/logistics system for Tankers International, which provides intelligent support in the scheduling of a 46-strong Very Large Crude Carrier (VLCC) fleet.

Categories and Subject Descriptors

I.2.11 [**Distributed artificial intelligence**]: multiagent systems; I.2.8 [**Problem Solving, Control Methods, and Search**]: scheduling

General Terms

design

Keywords

multi-agent systems, industrial applications, logistics, scheduling

1. INTRODUCTION

MAGENTA is a software development company that was started in 1999. MAGENTA has a head office is in the UK, and its software development base is in Samara, Russia. The company was established with the mission of commercialising and exploiting multi-agent systems technology, and to this end the company has developed a well-supported, Java-based ontology development toolkit and associated multi-agent toolkit. MAGENTA has successfully used this technology in several field-tested systems: the focus of most attention in applications to date has been on logistics – and in particular, scheduling – applications. Our aims in this paper are two-fold:

1. *To introduce MAGENTA's ontology and multi-agent systems technology.*

   Over the past six years, MAGENTA has developed commercial industrial-strength multi-agent systems technology, based on a Java enterprise base, which combines ideas from the semantic web (in particular, a rich framework for specifying and deploying *ontologies* for applications [6, 2]), with a multi-agent development framework that allows agents to interact in a *virtual marketplace*, which constitutes a toolset for building systems of cooperating agents, and a run-time platform for executing MAGENTA agents. Our first aim is therefore to give an introduction to MAGENTA's agent technology, and informally describe how the components can be used to build industrial multi-agent systems.

2. *To illustrate the technology by means of a logistics case study: the i-Scheduler system.*

   MAGENTA's agent technology is unusual in that it has been used to a number of deployed, commercial systems, particularly in the logistics/scheduling area. To illustrate the technology, we describe one such application: a field-tested scheduling/logistics system for Tankers International, which provides intelligent support in the scheduling of a 46-strong Very Large Crude Carrier (VLCC) fleet.

The remainder of this paper is structured as follows. We begin, in the following section, by giving a general overview of MAGENTA's agent technology, and the different software components that this technology provides. In section 3, we describe the *i*-Scheduler system, how it was implemented with MAGENTA's technology, and the key benefits that this system has brought. In section 4, we briefly point to related work and give some conclusions.

2. MAGENTA'S AGENT TECHNOLOGY

There are three main components to MAGENTA's agent technology (see Figure 1):

- Java 2 Enterprise Edition (J2EE) – the language foundation upon which MAGENTA's agent technology is built.

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

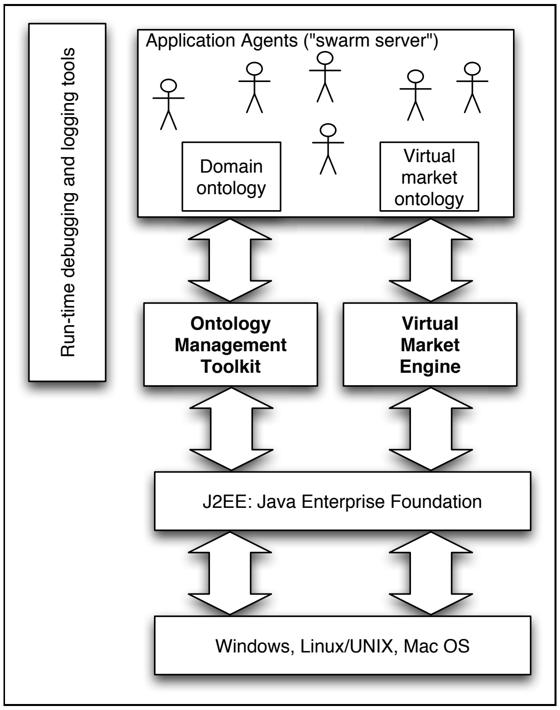

Figure 1: MAGENTA's agent technology: overview

- the *Ontology Management Toolkit* – the OMT enables a system designer to capture the concepts and the interrelationships between concepts in a problem domain – these will provide the *knowledge base* that agents will use when making decisions.

- the *Virtual Market Engine* – is the heart of MAGENTA's agent technology. It provides the tools to build and run agents, and enables them to make use of the knowledge base defined using the Ontology Management Toolkit. It also provides tools for run-time monitoring and debugging of a system, enabling a developer to picture and understand what agents are doing and why.

2.1 J2EE Foundation

MAGENTA's agent technology is built upon Sun Microsystem's Java 2 Enterprise Edition (J2EE) – Sun's enterprise Java solution. J2EE is the standard for developing multitier enterprise applications. By building upon J2EE, MAGENTA's agent technology provide a seamless route into the world of Enterprise Java Beans, Java Servlets, JavaServer Pages, XML, and web services, via the WS-I Basic Profile.

2.2 The Ontology Management Toolkit

Most software developers are by now well used to the idea of business object models. Crudely, a business object model is a collection of classes in an OO language such as Java, which captures the main components of a business – events, accounts, people, transactions, and of course the physical components of the business such as trucks, buildings, rooms, factories, and production lines. As much as it is about anything, successful OO development is about being able to develop good business object models – understanding what needs to be captured and how to capture it, and understanding what makes an elegant and useful model. However, in addition to business objects, MAGENTA's Ontology Management Toolkit allows developers to capture their business *knowledge*, by means of an *ontology*.

An *ontology* is a *formal description of a domain* [6]. "Formal" here simply means that the description is in a format that can be directly processed by computers, and the "domain" is the area of a business application. Thus, an ontology is a description of the concepts in an application, the relationships between these concepts and the attributes of these concepts, all represented in a form that can be processed by computer. One of the main goals in developing an ontology is to enable computer programs – and in particular, software agents – written by different organisations or individuals to agree on a common set of terms. By coming to such an agreement, businesses (and business units) can then effectively *share* business knowledge, which represents a key step towards successful interworking. Thus, when one agent uses the term "centimetre" while communicating to another agent, the other agent needs to share the same meaning of the term. By providing a common ontology, which both agents can access and make use of, we can provide such shared meaning.

MAGENTA's Ontology Management Toolkit enables a developer to define an ontology using a graphical user interface, by drawing a graph of the main concepts to be used, and making links between them to indicate relationships (see Figure 2). The GUI-based ontology editor was itself developed using J2EE, and is implemented as a server based web system. Developers can thus use the web-based front end to develop and deploy systems, making it easy to develop and share ontologies in development teams.

Ontologies are now increasingly used in business applications, with interest in them being driven in part by their use in the semantic web [6, 1]. As well as a number of ontology development tools, (see section 4), there are also now a number of competing ontology definition languages: the best known being OWL [2]. MAGENTA's ontology toolkit provides an ontology definition framework that will be immediately familiar to OWL ontology developers, but extends its capabilities in a number of ways. The rationale for these extensions is to provide access to the richer facilities of MAGENTA's agent toolkit; however, compatibility with OWL is possible, by means of an "OWL mode" option in the ontology developer. We briefly provide an overview of the key ideas supported by the ontology management toolkit.

The top-level idea in the ontology management toolkit is that of a *concept*. A concept is rather like a class in OO terms. Concepts correspond to entities ("nouns") in the system domain that are not active, choice making components. The basic ontology model thus consists of identifying and defining these concepts. Once the concepts in an ontology have been defined, the basic ontology model can then be refined by annotating concepts with their *attributes*. Attributes come in two types: *slots* and *relations*.

Slots correspond approximately to instance variables in OO terms. For example, a "person" concept might have an "age" slot, where the age is a positive integer.

Relations can be thought of as slots where the value of the slot is another concept. For example, we might define a Client concept and a Service concept, with the "purchases" relation between them (i.e., "a client purchases a service"). The attributes of a client might be name, address, and client code. The attributes of a service might be name, cost, and so on. We can document (for example) whether the relationship has any special properties, such a transitivity (e.g., "if Y is bigger than X and Z is bigger than Y, then Z is also bigger than X"). We can also define default properties and attributes of classes. Special relationships between concepts (such as the idea that one concept is a specialisation of another) are also possible.

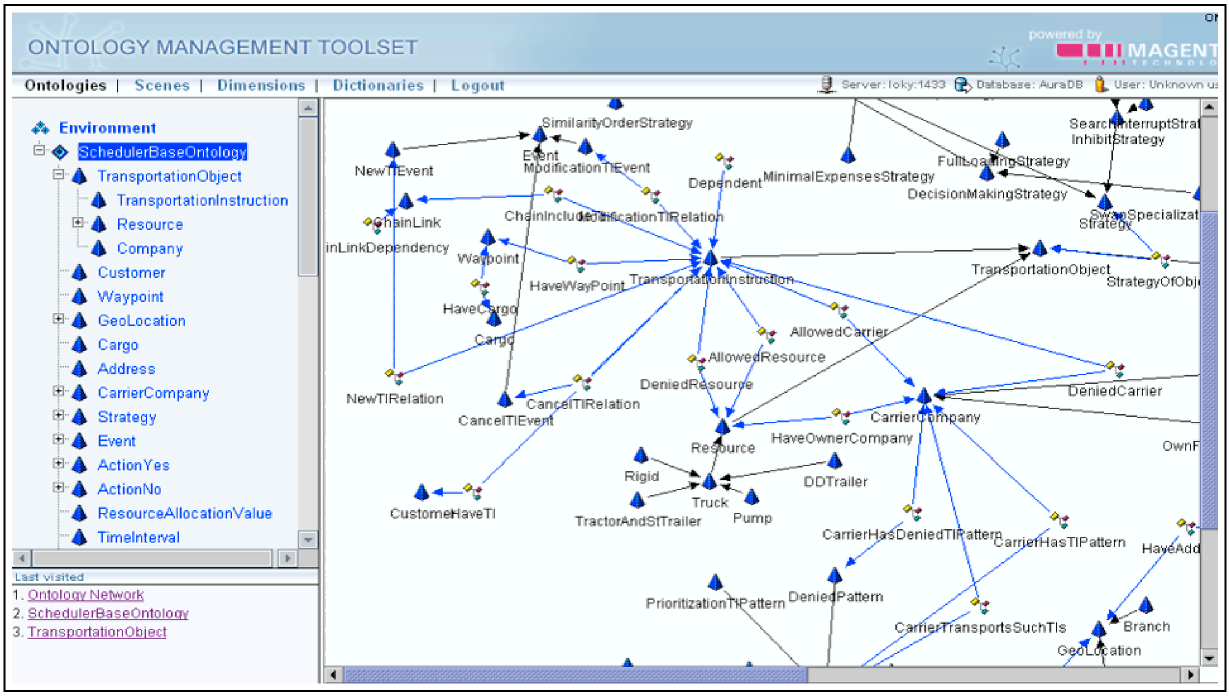

Figure 2: The Ontology Toolkit Interface, and a Fragment of the *i*-Scheduler Ontology

Attributes may be of several different types. All the obvious basic data types are provided (integers, Booleans, strings).

MAGENTA's ontology management toolkit also permits the definition of *computable* attributes: that is, we can state how an attribute is to be computed from other concepts and attributes. For example, we might define a "family" concept, which consists of a number of "people", where both "family" and "people" concepts have an "income" attribute, where this attribute in the "family" concept is defined to be the sum of the incomes in the respective family members. In other words, we can capture in our ontology the fact that the income of a family is the sum of the incomes of the family members. These computable attributes are directly implemented during the generation of Java code from the ontology, as we describe below. In much the same way, it is possible to describe computable *constraints* on classes, where these constraints represent predicates that instances in the ontology must satisfy. For example, a computable constraint might specify that the purchase amount for any item by some person must be less than or equal to the money available to that person. Again, these constraints are directly implemented when Java code is developed from an ontology.

Finally, the Ontology Management Toolkit permits a user to define *instances* of classes. For example, "Jon Himoff" and "Petr Skobelev" might be instances of a "person" class. Together, the classes, attributes, relations, constraints, and instances constitute a *knowledge base* [7]. In this way, by using the Ontology Management Toolkit, we capture not just the basic data properties of a domain ("a client has a name, address, and code"), but also the additional knowledge required to interpret and make sense of the domain.

Once a developer has created and refined an ontology using MAGENTA's ontology management toolkit, there are several options. First, an ontology is a valuable analysis tool in its own right – it serves as a detailed and unambiguous document of the domain at hand. But perhaps more usefully, using MAGENTA's ontology management toolkit, it is possible to generate code (Java class files and method stubs) at the press of a button. The code generation step automatically handles concepts, attributes, sub-class ("is-a") relationships, and other relations, including code to handle computable attributes. There is therefore a direct, automatic route from the definition of an ontology to its deployment as Java code in applications and systems. Using the ontology management toolkit, it is also possible to import and export ontologies in a variety of other formats, perhaps most importantly, standardised XML-based formats such as OWL [2].

There are many ontology development tools on the marketplace, which appear at first sight to provide similar functionality to MAGENTA's ontology management toolkit; we describe some in section 4. But of these, we are aware of none that provides the push-button transition to J2EE that is provided by MAGENTA's ontology management toolkit.

2.3 The Virtual Marketplace Engine

MAGENTA's Virtual Marketplace Engine is basically an operating system for agents. Just as a regular operating system provides an environment that supports programs, applications, and files, so the Virtual Marketplace Engine provides an environment in which agents can exist, communicate, and, most importantly, work together. Another way of thinking about a Virtual Marketplace is as a kind of *virtual organisation* or *electronic institution*.

To build a multi-agent system with the Virtual Market Engine, it is first necessary to define a *domain ontology*, as discussed above. The domain ontology captures knowledge about a domain – the entities (concepts) in it, how they are related together, and the properties that they have. Note that the domain ontology is entirely application-independent: once developed, it can be deployed and reused in a range of different applications.

The second step to building a multi-agent system involves creating a Virtual Market Ontology. This is the "application specific" part of an ontology, which relates directly to the application at hand. The Virtual Market Ontology provides a number of pre-defined

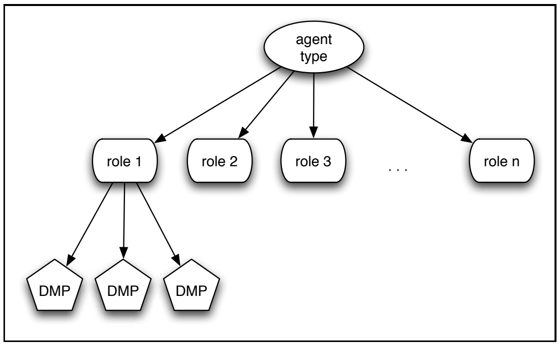

Figure 3: Agent types, roles, and decision making points.

concepts, which are important in developing the actual system:

- *role*;
- *agent type*; and
- *message type*.

A *role* in the Virtual Marketplace Ontology[1] represents a discrete, isolated chunk of activity that an agent can carry out (somewhat like roles in the Gaia methodology [9]). One of the most important aspects of roles is that they define how an agent can interact with other agents, via negotiation and cooperation. An agent type is a conglomeration of roles. An agent can play more than one role, and in order to play a role, an agent delegates the responsibility of acting to the role. A role itself is defined in terms of *decision making points*: as the name suggests, these represent the basic "reasoning" mechanism of an agent. Decision Making Points will make use of the Domain Ontology and Virtual Market Ontology. The overall structure is illustrated in Figure 3.

The final component of the MAGENTA agent framework is the notion of a *message*. We do not generally think of agents as invoking procedures or methods on one-another, in the same way that objects do. Although there are several existing message-based languages for agent communication (notably the KQML and FIPA languages [4]), these are somewhat controversial, and there is no real agreement that these languages are appropriate for all applications. For this reason, MAGENTA's agent framework provides the tools and templates to define a range of messaging frameworks. Message types are defined as part of the Virtual Market Ontology; the behaviour of agents on receipt of a message is defined via "message handlers". Compatibility with existing agent communication languages is supported by defining, for example, a "KQML Virtual Marketplace", or a "FIPA Virtual Marketplace".

In summary, to build a multi-agent system using MAGENTA's agent technology, we carry out the following steps.

1. Take an existing domain ontology off the shelf, or else define one from scratch, using the Ontology Management Toolkit.
2. Define a Virtual Marketplace, by constructing a Virtual Market Ontology consisting of:
   (a) The roles that agents play.
   (b) The agent types that represent aggregations of roles.
   (c) The message types that agents use to communicate with one-another.
   (d) The decision making points, which define how agents behave.

The multi-agent system is then created as J2EE Java code, which can be directly executed using MAGENTA's ontology management toolkit and agent run-time support frameworks. To assist the developer in debugging and understanding the behaviour of the system, run-time support tools are provided for logging and tracing activity; the *agent log* is one such tool (see Figure 4). It enables a user to track the messages and behaviour of agents in the system.

3. AN INDUSTRIAL LOGISTICS APPLICATION

In this section, we give an overview of one of the main applications that has been developed using MAGENTA's agent technology: the *i*-Scheduler system developed for Tankers UK Ltd, the UK operations centre for Tankers International (TI). Tankers UK[2] is a shipping company which operates one of the largest fleets of Very Large Crude Carrier oil tankers in the world – currently consisting of more than 40 ships. The oil transportation market, in which TI operates, is subject to frequent and unexpected fluctuations in transportation and other costs, which means that to maximise revenue, TI must be able to optimise its shipping schedules as events occur: these schedules determine what ships will carry which cargoes where and when.

Before the MAGENTA solution was developed, scheduling was done manually, by human operators. However, this approach suffered from several disadvantages. For example, it is hard for humans to take into account all the relevant factors in determining a "good" schedule. More precisely, the following difficulties arise when attempting to maintain an optimal schedule for such a large fleet in real-time:

- First, there are many *inter-dependencies* between different components of the schedule. That is, small changes in one part of a schedule can have repercussions in another apparently separate part of the schedule, and it is difficult for human operators to be able to correctly and completely predict these repercussions.
- Second, resources in the system are highly constrained: vessels have fixed parameters (capacity and types of cargo) that must be considered when developing a schedule. It is often hard for humans to accurately and completely recall these parameters. Moreover, a "good" schedule will take into account *preferences* relating to vessels and cargoes, where these preferences are soft constraints that will improve the overall quality of the solution in some way[3].
- Third, the overall planning and operations teams managing the fleet (as well as the fleet itself) will be spread across many different locations and time-zones.
- Fourth, optimal scheduling in a scenario such as this is trivially seen to be a computationally complex (NP-hard) process, and as a consequence, centralised solutions are generally not feasible.

[1]The usage here should not be confused with the term as it is often used in the context of an ontology.

[2]See `http://www.tankersinternational.com/`.

[3]This approach is dubbed *pro-active constraint stressing*.

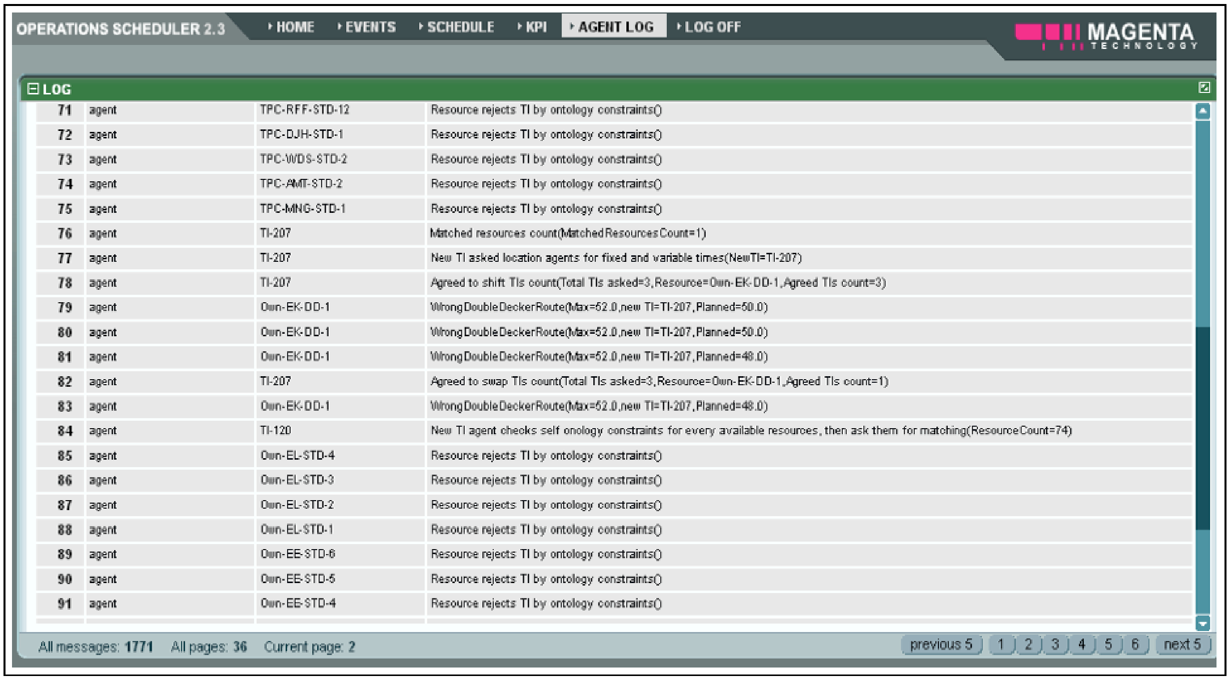

Figure 4: The Agent Log

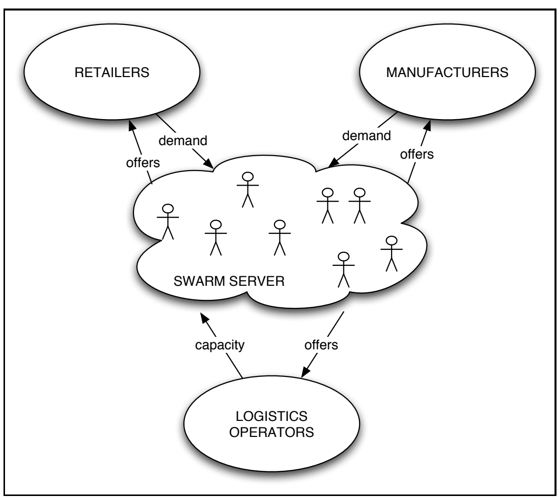

Figure 5: The *i*-Scheduler: An Industrial Logistics Application

- Fifth, scheduling in this scenario needs to be *comprehensible* to human schedulers. One difficulty with existing, centralised, rule-based scheduling solutions is that they deliver solutions that, while ostensibly "good", are in fact hard to rationalise, understand, and explain from a human point of view. This makes them hard to justify, explain, and modify by human schedulers – all essential characteristics of a useful solution from the TI point of view.

These factors tended to suggest a *multi-agent* solution, and MAGENTA's technology was deployed to develop such a system. This system is now in daily use at TI. We give an overview of the main features and design rationale for the system:

- First, the inter-dependencies between components being modelled in the system, and the constraints on these entities in the system, were rigorously and accurately documented using MAGENTA's ontology toolkit. (In fact, Figure 2 shows a fragment of the ontology that was actually developed and deployed in this application. This fragment illustrates that "Customer", "Cargo", "CarrierCompany", and "Event" are all examples of concepts in the *i*-Scheduler ontology.)
- Second, a multi-agent solution inherently supports a distributed solution: the Java/web based interfaces provided by MAGENTA's solution enable views to be easily distributed.
- Third, the multi-agent approach developed leads to an efficient solution to the problem. The approach adopted, which we describe below, is *sound* in that it guarantees any solution found will satisfy the constraints specified; but it is not *complete*, in the sense that it is conceivable that some possible solutions will be missed.
- Finally, the solution is derived in such a way that it tends to be comprehensible to humans. We will see this is more detail below.

In the *i*-Scheduler system as developed and deployed at TI, there are 63 concepts in the ontologies used to capture the domain, and five agent types: location, TI, resource, waypoint, and cargo. In the run-time system, there are typically of the order of 1000 agents in total. The overall goal of the system is to give support to human schedulers, who can use the system to iteratively explore possible solutions to scheduling problems, and derive schedules that maximise various different desirable criteria. For example, a typical, widely used strategy is to maximise what is known as "Time Charter Equivalence" (TCE), which roughly translates to income per day.

The basic run-cycle of the system involves the creation of *events* to the system, and of the agents in the system *reacting* to these events. An event would correspond to, for example, a new cargo needing shipping from one location to another. When such an event occurs, the agents respond to it by *negotiating an alternative solution*. The negotiation process is "benevolent", in the sense that the agents can always be relied upon to try to find a solution, under the proviso that this solution must satisfy the criteria as specified in, for example, the domain ontology.

The negotiation process involves two aspects. First, where a solution (involving carrying a cargo) is possible, those agents who can expedite it will always try to help. They do this by making proposals to the cargo agents, who will typically agree to the solution that maximises the criteria specified by the human scheduler overseeing the process. Where a cargo cannot directly be accommodated in this way, the agents will try to find an alternative negotiated settlement, by proposing to *swap* cargoes with each other where necessary. By swapping cargoes, with the goal of maximising the selected criteria, the agents effectively undertake a cooperative search of the possible solution space. Negotiating a deal via swapping takes place down to the 6th level; far more than could be considered by a human scheduler, and in practice typically enough to find a "satisficing" solution to the problem. The agents undertake the negotiations in a virtual marketplace implemented using the MAGENTA agent toolkit.

The system is designed so that solutions can be explored which involve different criteria for optimisation. These are presented as options to the human scheduler. The negotiation process, and in particular, the reasons why offers made by agents are accepted or rejected, provide a *rationale* for the proposals of the system, which can be easily understood by human schedulers.

Of course, as with an multi-agent solution, much of the codebase in the *i*-Scheduler is not related to the ''intelligent" components of the system, but dealing with more mundane – but none the less critical – components of the system, such as the user interface, report generation, interaction with legacy components and databases, and so on (cf. [3]). The *i*-Scheduler permits human schedulers to obtain many different reports on the system, giving multiple possible views of it. Figure 6 shows one typical report – a Gannt chart for scheduled voyages for a particular vessel, illustrating the different scheduled activities of that vessel.

The *i*-Scheduler system is currently in daily use at TI, where it assists human schedulers to develop schedules, respond quickly and optimally to unexpected events, and explore "what-if" scheduling scenarios. The perceived benefits of the system are manifold, including, for example:

- the precise documentation, using the domain ontology, of the knowledge (including constraints and preferences) of the components of TI's business;
- the fact that the system can execute while holding many different constraints in mind, guaranteeing that if any solution is put forward, it will satisfy these, thereby eliminating potentially costly and embarassing mistakes; and
- the fact that the solutions provided by the system are comprehensible to human schedulers, with automatically generated rationales that resemble those used by human schedulers.

4. RELATED WORK AND CONCLUSIONS

Although many platforms and architectures for multi-agent systems have been described over the past two decades [8], few of these have been used to build systems that have been deployed outside a research lab – and even fewer have been used to develop systems that have made it through to use in the field. In this sense, MAGENTA's agent technology is already unusual. One advantage of this approach is that MAGENTA's technology has been thoroughly tested and is well-supported: the rigours associated with deployment have meant that extensive effort has been devoted to ensuring that the codebase meets the standards required of commercial software. From the multi-agent systems research point of view, perhaps the most important unique feature of MAGENTA's technology is the fact that it provides an integrated set of tools for ontology *and* agent development, with a seamless, semi-automatic route to Java code generation. Although several well-known ontology editors are widely used in academia (notably Protégé [5]), these editors do not typically generate Java code, and integration with an agent toolkit must be performed manually. It is also worth noting that MAGENTA's technology does not support either of the "standard" agent communication languages: FIPA or KQML. But of course, this does not prevent MAGENTA agents from interacting with other agent systems via these languages: for example, FIPA can be supported internally in a MAGENTA system by defining a "FIPA virtual market", and may be supported for external interworking via existing Java-based FIPA-compliant messaging platforms.

In this paper, we have introduced MAGENTA's agent technology, which is comprised of two key components: the ontology management toolkit and the virtual marketplace. These two components, built on top of J2EE, provide a simple and seamless route into the development of enterprise Java-based agent systems. We have also described the *i*-Scheduler system, built using MAGENTA agent technology, which is currently in use at Tankers UK in the scheduling of a large fleet of "Very Large Crude Carrier" oil tankers. MAGENTA's technology continues to be developed, and this development goes hand-in-hand with the development of other commercial applications. The short term plan for the company is to focus on logistics applications, exploiting the experience gained in the *i*-Scheduler system, but also looking to other markets where the technology can be fruitfully deployed.

5. REFERENCES

[1] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. The MIT Press: Cambridge, MA, 2004.

[2] The OWL Coalition. OWL: The web ontology language, 2004. See `http://www.w3.org/2004/OWL/`.

[3] O. Etzioni and D. S. Weld. Intelligent agents on the internet: Fact, fiction, and forecast. *IEEE Expert*, 10(4):44–49, August 1995.

[4] FIPA. The foundation for intelligent physical agents, 2001. See `http://www.fipa.org/`.

[5] The Protégé Group. The Protégé ontology editor, 2004. See `http://protege.stanford.edu/`.

[6] T. R. Gruber. The role of common ontology in achieving sharable, reusable knowledge bases. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*. Morgan Kaufmann Publishers: San Mateo, CA, April 1991.

[7] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology, 2004. See `http://protege.stanford.edu/publications/`.

[8] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[9] M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (Agents 99)*, pages 69–76, Seattle, WA, May 1999.

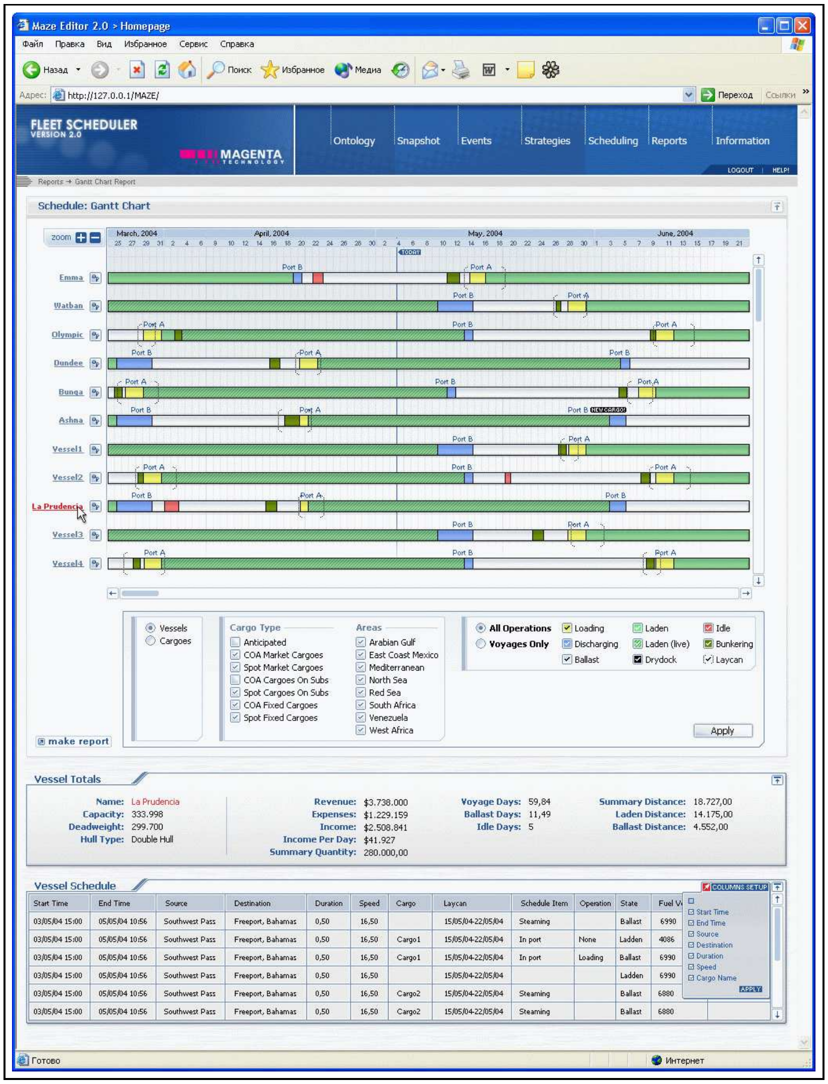

Figure 6: Gantt Chart for planned voyages with a schedule for a selected vessel: a typical report from *i*-Scheduler.