

# **ZEUS: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems**

**Hyacinth S. Nwana, Divine T. Ndumu & Lyndon C. Lee**

Intelligent Systems Research,  
Advanced Research & Technologies Department  
BT Laboratories, MLB1, PP12,  
Martlesham Heath  
Ipswich, Suffolk IP5 3RE  
England

{hyacinth, ndumudt, lyndon}@info.bt.co.uk

## **Abstract**

There is an emerging consensus on the need to develop methodologies and tool-kits for building distributed multi-agent systems. This paper presents ZEUS, an advanced development tool-kit for constructing collaborative agent applications. ZEUS is a culmination of a careful synthesis of established agent technologies with the addition of some new ones, to provide an integrated environment for the rapid software engineering of collaborative agent applications. ZEUS defines a multi-agent system design methodology, supports the methodology with an environment for capturing user specification of agents, and automatically generates the executable source code of the user-defined agents. We also report on preliminary informal evaluation of ZEUS on three domains.

## **Keywords**

Agent architectures, distributed multi-agent systems, constructing collaborative agent applications.

## **I. Introduction**

This paper describes ZEUS, an advanced tool-kit for building distributed multi-agent applications. The tool-kit facilitates the construction of *collaborative* agent applications, arguably one of the more complex of agent types because they can have rich, deliberative internal models and operate in open and time-constrained environments (Nwana 1996).

The reasons for the development of ZEUS fall into two broad categories. Firstly, there is an emerging consensus amongst agent researchers of the need to develop methodologies and tool-kits for building distributed agent systems, moving away from point solutions to general architectures, frameworks and tool-kits (Ndumu & Nwana 1996). Secondly, a tool-kit like ZEUS facilitates the *engineering* of agent applications; it speeds up development time and encourages code reuse and standardisation of agent technology.

The breakdown of the rest of the paper is as follows. Section 2 presents the philosophy and a clearer specification of ZEUS. Section 3 describes briefly a ZEUS agent and outlines our agent design methodology, and Section 4 describes the ZEUS tool-kit. Section 5 summarises the ZEUS visualisation and debugging suite of tools while Section 6 reports on on-going evaluation of ZEUS. Section 7 covers related other work and Section 8 concludes with a brief discussion of some lessons we have learnt from constructing and (preliminarily) evaluating ZEUS.

## 2. The ZEUS Philosophy and Specification

ZEUS arose out of a need to provide a *generic*, *customisable*, and *scaleable* industrial-strength collaborative agent building tool-kit. We required that the tool-kit encapsulate the following principles:

- Firstly, it should clearly delineate between *domain-level* problem-solving abilities and *agent-level* functionality. The latter covers issues of communication, co-operation, co-ordination, task execution and monitoring, exception handling, etc. while the former allows for the acquisition of domain-specific knowledge. Thus, the tool-kit should provide agent developers with all the agent-level functionality; such that they need only provide the code implementing the domain-specific problem-solving abilities of the agents they define.
- Secondly, the tool-kit should employ the direct manipulation metaphor whenever feasible. This HCI metaphor is epitomised by the visual programming paradigm and the ‘pick-and-choose’ metaphor. That is, application developers should be able to select from varied menus, the functionality and modalities required of their agents.
- Thirdly, the tool-kit should utilise ‘standardised’ technology wherever feasible as is exemplified by our employment of KQML (Finin & Labrou, 1997) as our agent communication language.
- Fourthly, the tool-kit should support an open design to ensure it is easily extensible.

On the broad goals we set of ourselves for ZEUS, we required a tool-kit which provides a multi-agent systems developer an environment which supports the following:

- the *configuring* (definition) of a number of different agents of varying functionality and behaviour;
- the *organisation* of the agents in whatever manner using the relationships *sub-ordinate*, *superior*, *peer* and *co-worker*. Superior and sub-ordinate links define authority relationships between agents. Co-workers are agents in the same organisation who have no authority relation between them, and peers are agents belonging to different organisations;
- the *imbuing* of each agent with selected ZEUS-supplied agent-level communicative and co-ordination mechanisms. Different co-ordination protocols which the designer can choose from are provided by ZEUS including master-slave, contract net, a variety of auction strategies, negotiation for time (where the agent requests for more time for some goal), cut-price negotiation (where the agent offers to reduce the cost it is charging in exchange for more time for some goal), etc. We also allow for new user-supplied protocols as long as they conform to our *universal co-ordination protocol*;
- the *supplying* of each agent with the necessary domain-specific problem-solving code; and
- the *automatic generation* of the required executables for the agents.

In addition, we believed the tool-kit should support predefined agents such as name-server agents (white pages), facilitator agents (yellow pages) and a visualiser. A key goal of the visualiser was for analysis and debugging of distributed agent systems, as it is well known that this is a notoriously complex issue. Our visualiser is reported in Ndumu *et al.* (1997).

In summary, the specific goals for the ZEUS project included:

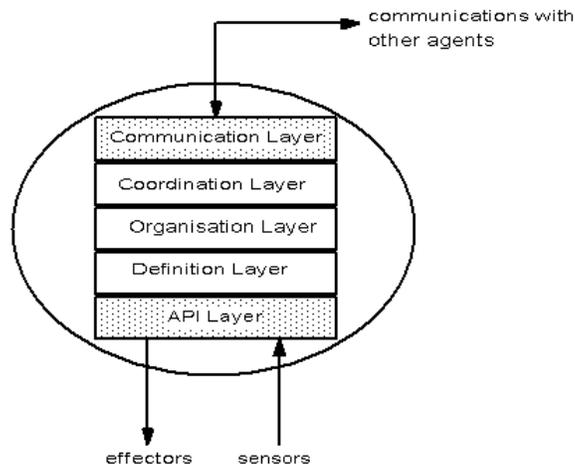
- defining and developing a generic ZEUS agent,

- developing a clear methodology for building distributed multi-agent systems and the ZEUS tool-kit to support/instantiate it,
- developing a suite of visualisation and debugging tools for the ZEUS tool-kit.

The next section describes a generic ZEUS agent and briefly outlines the ZEUS agent design methodology. Section 4 presents the ZEUS tool-kit and Section 5 describes visualisation and debugging support in ZEUS.

### 3 A Generic Zeus Agent and The Zeus Agent Design Methodology

At the highest level of abstraction, a ZEUS agent is composed of three layers: a definition layer, an organisation layer and a coordination layer (Fig. 1).

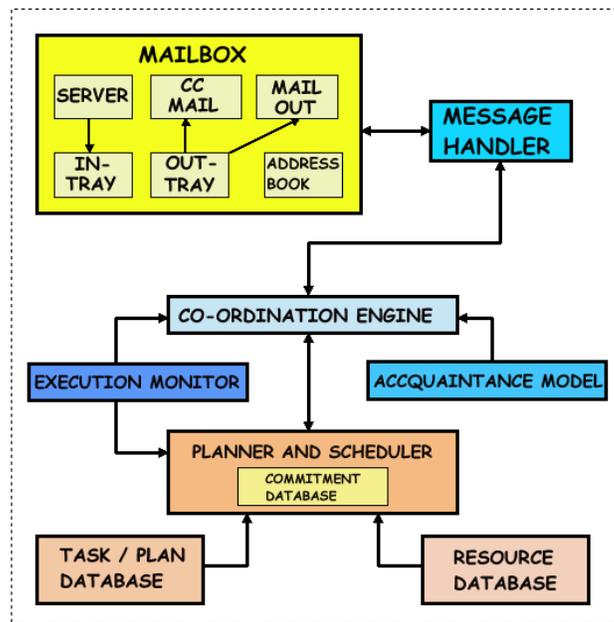


**Fig. 1: An abstract context diagram of a generic agent**

At the definition layer, the agent is viewed in terms of its reasoning (and learning) abilities, its goals, resources, skills, beliefs, preferences, etc. At the organisation layer it is viewed in terms of its relationships with other agents, e.g. what agencies<sup>1</sup> it belongs to, what roles it plays in these agencies, what other agents it is aware of, what abilities it knows those other agents possess, etc. At the coordination layer the agent is viewed as a social entity, i.e. in terms of the coordination and negotiation techniques it possesses. Regarding the other two layers, the communication layer handles the low-level details involved in inter-agent communication, while the application programmer's interface (API) layer links the agent to the physical realisations of its resources and skills.

Fig. 2 depicts the architecture of a Zeus agent, which is not too dissimilar from others in the literature. The coordination layer of Fig. 1 is captured within the coordination engine of Fig. 2; the organisation layer roughly translates to the acquaintance model while the definition layer refers to the entire agent template. The key difference between ZEUS and other agent environments is how ZEUS supports the rapid engineering of agent systems with the internal complexity of Fig. 2.

<sup>1</sup> An agency is a group of related agents, i.e. they share a common attribute, for example, they may belong to the same company. Agencies may be virtual or real. A virtual agency is a group of agents who share some sort of cooperation agreement.



**Fig. 2: The architecture of a ZEUS agent**

### 3.1 Definition of a ZEUS Agent

In order to illuminate the definition of a ZEUS agent in our methodology, we briefly describe a simple implemented scenario. The domain is supply chain provisioning in which agents collaborate to manufacture and/or provision goods making up a service. In the example, we have five principal agents: *C*, a computer manufacturer, which has two subordinates, *M* and *U*. *M* produces monitors, and knows *C* as its superior and *U* as its co-worker. *U* produces central processing units (CPUs), and similarly knows *C* as its superior and *M* as its co-worker. Both *M* and *U* share two subordinates (*X* and *Y*). *C* knows of another agent *P* as a peer who produces printers. *P* has a subordinate *T* which produces printer ink and toner cartridges. In the example, the production of a computer requires the base unit (CPU and monitor) as well as a keyboard and an appropriate printer. In the environment are four additional support agents: an agent name server (ANS), which provides a white-pages facility for agent address look-up; a facilitator agent (PC\_Broker), which provides a yellow-pages facility through which agents find other agents capable of performing a task; a database proxy agent (DB), whose sole function is to store and retrieve messages from proprietary databases on demand from other agents, and a visualiser agent (Visual). These agents all communicate in the common agent communication language, KQML in our case.

We mainly concentrate on the definition of agent *C* in the above scenario in order to present the processes involved in the definition of a ZEUS agent. Recall the three layer model: agent definition, organisation and coordination. We begin with the agent definition step.

Firstly, for agent *C*, the designer in *defining* it has to answer the following questions:

- What tasks can *C* perform (i.e. what goals can it achieve)?
- How many domain tasks can it perform concurrently?
- How far ahead in time does *C* normally plan its activities?
- What are the initial facts (resources) available to it?
- What proportion of double-booking of its resources does *C* allow?

The metaphor underpinning this set of questions is that of a *job-shop scheduling manager*. *C* has a number of resources (e.g. keyboards in our scenario). The realisation of a goal by *C*

may consume some resources, such as the assembling of a PC consumes a keyboard, a printer, a CPU and a monitor. C’s role is to produce as many computers as possible, in such a manner that it minimises its idle time and maximises its profit. In order to do this, it has to schedule customer requests on the basis of available resources, free assembly lines and the cost and time of performing each task. Typical customer requests are of the form “produce a computer  $x$  by time  $y$  at cost  $z$ ”. To aid its scheduling, C maintains a *commitment database* (see Fig. 2).

Another useful feature which ZEUS agents possess is their ability to ‘double-book’ their time slots in the expectation of unreliable customers. Thus designers are also asked the degree of double-booking they require (the default is 10% in ZEUS). We borrowed this feature from airline reservation systems.

For the *agent organisation* step, the designer of C answers the following questions:

- What other agents in the community does C know (these are *beliefs*)?
- What primary structural relationships does C know it has with each agent it knows (*beliefs*)?
- What items (if any) does C know that other agents in the community can produce, and at what average cost and time (*beliefs*)?

From agent C’s perspective, Table 1 summarises one possible view of its organisational knowledge as required by the preceding three questions.

Base Agent	Relation	Target Agent	<i>Beliefs</i> C possesses
C	sub-ordinate	M	(:item Monitor :time 4 :cost 200)
	sub-ordinate	U	(:item CPU :time 3 :cost 300)
	peer	P	(:item: Printer :time 5 :cost 250)
...	...	...	...

**Table 1: An instance of the organisational knowledge of agent C**

The next step is the *coordination* step. Here, the designer selects from a ZEUS supplied list zero or more negotiation/coordination strategies which C can invoke. C clearly has subordinates and so ideally should be able to invoke a *master-slave* protocol. It also needs to deal with peer agents like P, so again ideally needs to be imbued with the contract-net protocol so that it can announce bids, obtain replies to its call for bids and make decisions. It may also be imbued with the other strategies mentioned in Section 2 such as the auction protocols.

The next step is to define the *tasks* which C can perform, e.g. it can assemble computers. For this, the designer performs the following:

- Determine the average cost and duration of the task.
- Identify all the items (resources) required before the task can be performed.
- Identify all the items produced once the task is performed.
- Determine and note all the constraints between the consumed and the produced items.
- Determine if the task can be performed by directly executing a domain function (primitive task) or whether it is an abstract specification of a network of tasks (i.e. a summary task).
- If the task is primitive, then provide the following two functions: one to execute to perform the task, and the other to compute the actual cost once the task has been

performed. For summary tasks, provide a description of them in terms of their component tasks.

The metaphor here is of a task in the job-shop scheduling scenario discussed earlier. The last step is where designer provides the domain-specific code for the domain.

To round up this section, we return to Fig. 2 and describe its components, and relate them to the above discussion:

- The Mailbox handles communications between the agent and other external agents.
- The Message Handler processes messages incoming from the mailbox, dispatching them to other modules of the agent.
- The coordination engine and reasoning system takes decisions concerning the goals the agent should be pursuing, how they should be pursued, when to abandon them, etc., and coordinates the agent's overall activities. It also has a database of pre-built coordination strategies like contract nets, auctions and master-slave.
- The Acquaintance Model describes the agent's beliefs about the capabilities of other agents in the society, and its relationships to them (see Table 1).
- The Planner and Scheduler plans and schedules the tasks the agent is controlling, monitoring and managing based on decisions taken by the coordination engine and reasoning system, and the resources and tasks available to be controlled, monitored and/or managed by the agent.
- The Resource Database contains logical descriptions of resources available to the agent. It also provides an interface between the database and external systems such that the database can query external systems about the availability of resources, etc.
- The Task Database provides logical descriptions of tasks available for control, monitoring and/or management by the agent.
- The Execution Monitor starts, stops and monitors external systems scheduled for execution or termination by the planner and scheduler. It also informs the Coordination Engine of successful and exceptional terminating conditions of the tasks it is monitoring.

We have provided in this section a description of ZEUS agents, and also provided a flavour of how a ZEUS agent is actually defined. The broad steps carried out in defining a ZEUS agent forms the core of our agent design methodology. Naturally, there are much other details which we omit here (for a more complete specification of the methodology see Ndumu *et al.* (1997)).

#### **4 The ZEUS Tool-Kit**

The ZEUS tool-kit essentially supports and instantiates the methodology outlined in the previous section. ZEUS consists of a host of (visual) editors which an agent developer uses to specify the information required to define a ZEUS agent. The current suite of editors include the following:

- An Ontology Editor: for defining the ontology of the domain.
- A Task Description Editor: a visual editor for describing primitive tasks that the agent can perform.
- A Summary Task Editor: for describing summary tasks which are composed of one or more sub-tasks and ordered in some fashion.
- An Organisation Editor: a visual tool for defining the organisational relationships between agents and also beliefs that agents have about others in the system.

- An Agent Definition Editor: for describing agents logically, their abilities and initial resources.
- A Coordination Editor: for allowing the user select co-ordination protocols s/he wants to imbue an agent with or for describing new ones.
- A Fact/Variable Editor: for describing specific instances of facts and variables, using templates provided by the ontology editor.
- A Constraint Editor: for describing the constraints between(i) the preconditions and effects of tasks, (ii) one or more preconditions of a task, and (iii) the effects of a preceding task and the preconditions of a succeeding task in a summary task description.
- A Code Generation Editor from which the executable source code for selected agents are generated.

All the above editors essentially facilitate the identification and description of a set of agents, selecting agent functionality and inputting task and domain-related data. Hence, the output of the ZEUS tool-kit is then a logical description of a set of agents and a set of tasks to be carried out in a domain, together with executable code for each agent and *stubs* for executable code for each task. ZEUS therefore embodies a system of methods (i.e. methodology) plus environment.

In order to generate executables, in addition to the class of editors above, ZEUS also comes with the following predefined libraries:

- A library of co-ordination graphs which describe ZEUS-defined co-ordination algorithms/protocols.
- Relationships data library which define ZEUS-defined organisational relationships.
- An API library for housekeeping operations such as executing tasks and evaluating cost functions.
- An agent shell library which provides the agent template which dictates the agent structure shown in Fig. 2 including KQML communications, planning & scheduling algorithms, mailbox library, etc.
- Lastly, a set of standard supporting agents like agent name servers (ANS) (white pages), facilitators (yellow pages), and a visualiser.

Therefore, in order to generate the code for a specific application system, the code generator essentially *inherits* from the libraries and *integrates* the data from the various editors. ZEUS also provides a set of *scripts* for integrating the generated agents with the standard ones such as name-servers and visualisers into a coherent system. ZEUS allows for as many visualisers, name-servers and facilitators as is required by the user.

The entire ZEUS tool-kit has been implemented in the Java programming language and runs on Solaris, Unix and Windows NT platforms. Java was chosen for the following reasons. It is object-oriented, multi-threaded (each agent consists of several threads), portable across operating systems, web/HTML-friendly and it provides a rich set of APIs with its packages which are continuously being added to. ZEUS consists of about 60K lines of Java including the visualiser code.

## 5 Visualisation and Debugging in ZEUS

An area which the ZEUS tool-kit takes seriously, which we believe multi-agent system researchers have so far neglected is that of the *analysis*, *visualisation* and *debugging* of multi-agent systems. It is vital that a tool-kit of the like of ZEUS provides such facilities to the agent designer. Generally, visualising overall system behaviour in such systems with

distributed control, data and process is a notoriously difficult task. Each agent in the system has only a *local* view of the organisation, and the burden is on the user to integrate into a coherent whole the large amounts of scope-limited information provided by individual agents. Further, because of the complexity of multi-agent interaction and behaviour, effective visualisation assumes more importance than in single-agent systems. It is necessary in order to reduce the information overload on users, and thus allow them to

- confirm, understand, control and/or analyse the behaviour of the system, and to
- debug the system.

The debugging problem in multi-agent systems cannot be over-stated, not least because even when individual agents in an organisation are “correct” the overall system behaviour might be less than desired.

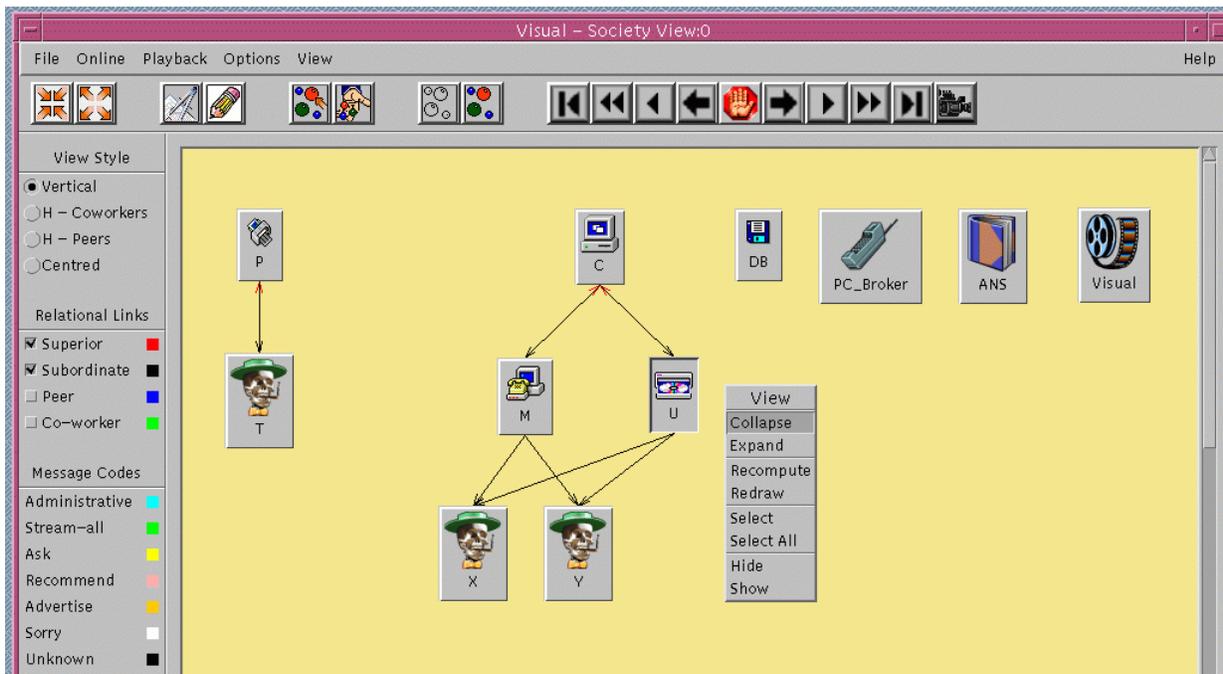
In this section, we briefly describe ZEUS’ visualisation system. It system comprises a suite of tools, with each tool providing a view of the distributed multi-agent application being visualised. Each tool interrogates the agents in the application, collates the returned information and presents this information to a user in an appropriate manner. This in essence shifts the burden of inference from the user to the visualiser. Note that the visualiser is another agent which just requests local information from agents and tries to build a global view. Some agents may not reply because they are suspended or dead, and in such a scenario, the global view presented by the visualiser will be incomplete. The current tool-set includes

- A *society tool* that shows the message interchange between agents in a society.
- A *report tool* that graphs the society-wide decomposition of tasks and the execution states of the various subtasks,
- A *micro tool* for monitoring the internal states of agents.
- A *control tool* for remotely modifying the internal states of individual agents. This tool is really used for the administrative management of the agents in the tool-kit e.g. killing of or suspending agents, adding, modifying or deleting goals, resources, tasks, coordination strategies, etc.
- A *statistics tool* for collating individual agent and society-wide statistics.
- A *video tool* (within the society tool, report and statistics tools) which can be used, video-style, to record, replay, rewind, forward, fast-rewind and fast-forward sessions from a database.

The video tool allows the society, report and statistics tools, in addition to functioning in *online* mode (i.e. visualising the ‘live’ interactions in a multi-agent set-up), to also support *off-line*, video-style replay of previously saved multi-agent interaction sessions. This multi-perspective visualisation approach gives users the flexibility to select which tool to use, when to use a particular tool, and what to visualise.

We briefly illustrate ZEUS’ visualisation tools suite by returning to the scenario we described earlier. To reduce the textual descriptions of these tools to a minimum, we provide several screendumps of the visualiser tools on this scenario. For details on the visualiser suite of tools, see Ndumu *et al.* (1997).

Fig. 3 shows the agents from the view of the Society Tool which clearly implements the scenario as described. Note again that the information shown was sent by all the different agents, and the visualiser just collated and presented this global picture of the society.



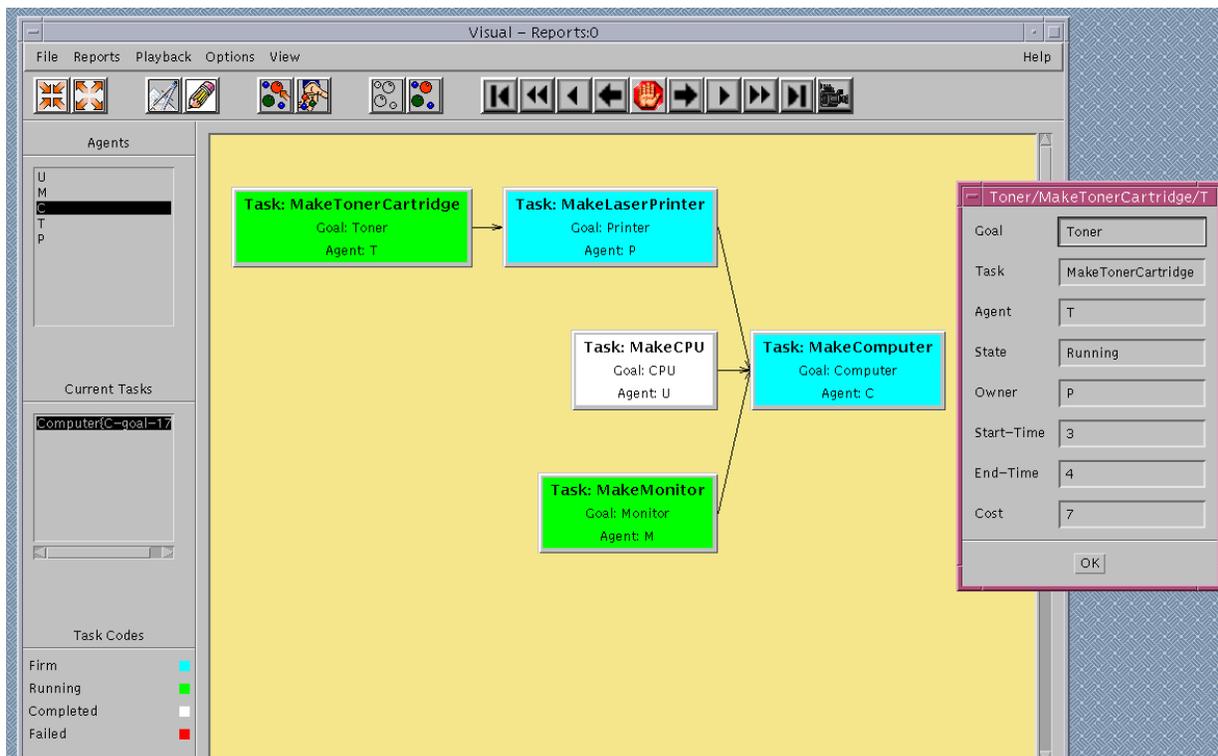
**Fig. 3:** The society tool showing the vertical layout of the agents in our scenario. On the left of the window are the layout and view controls. The toolbar buttons on the right control video-style replay of messages, while those on the left and the popup menu on the right control the position and visibility of the agent icons.

Fig. 4 shows the Report Tool on the same scenario. The Report tool provides a global view of the problem-solving in a society of agents; this is also a useful debugging and analysis tool. It allows a user to select a set of agents and request that they report to it the status of all their tasks/goals. Since each agent only has a local and incomplete view of the problem-solving effort of the society, this tool requests and collates the local views to provide a more complete picture. The user can select an agent of interest and a task owned<sup>2</sup> by that agent. For the selection of agent and task, the tool generates a GANTT chart showing the decomposition of the task, the allocation of its constituent subparts to different agents in the community, and when each agent is scheduled to perform its part of the task. As Fig. 4 shows, the task/goal of making a computer owned by agent C has been split up into three sub-tasks and awarded to agents P, U and M.

Fig. 5 shows the Micro-Tool view of agent C, which essentially maps onto the generic ZEUS agent template depicted in Fig. 2. The figure shows the internals of agent C as it is running, depicting the messages being received and sent out by the agent, a summary of the actions taken in response to incoming messages, a graphical depiction of the co-ordination process of different goals by the agent, a diary detailing the tasks the agent has committed itself to performing, and the current status of those tasks (i.e. waiting, running, completed or failed). It also shows a list of the resources available to the agent including those allocated to the different goals/tasks it has *committed* itself to, and a summary of the results of monitoring executing tasks or tasks scheduled to start execution.

These tools are invaluable in the analysis and debugging of applications, because they together *encourage debugging via corroboration*. By this we mean, if two or more different tools looking at the society from different viewpoints point to some component (e.g. some

<sup>2</sup> An agent *owns* a task if it is scheduled to perform the task at the root of the task decomposition hierarchy for the goal.



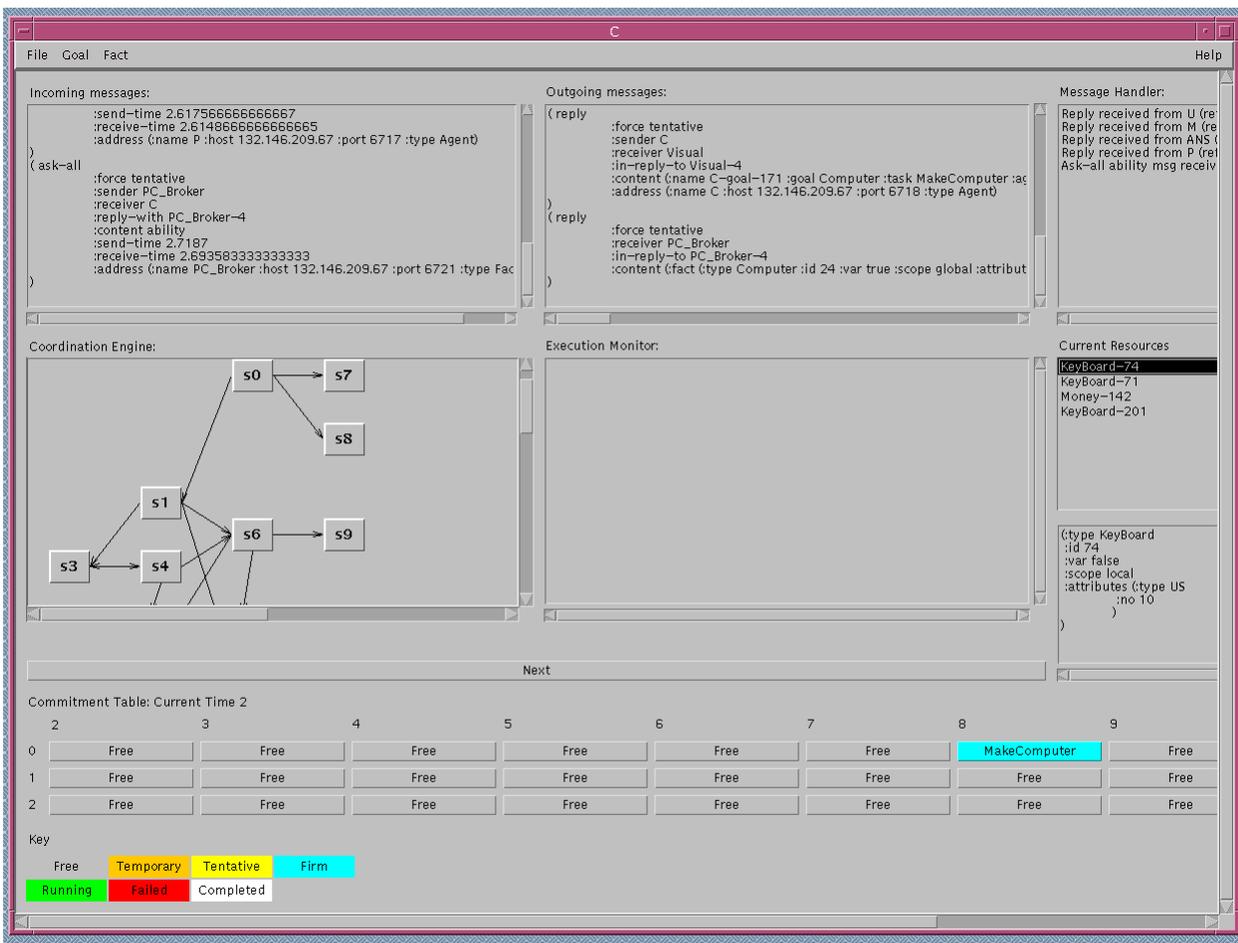
**Fig. 4: GANTT chart of the MakeComputer task. The tasks MakeTonerCartridge and MakeMonitor are running, MakeCPU has been completed, while MakeLaserPrinter and MakeComputer are waiting. The dialogue box on the right shows details of the MakeTonerCartridge task.**

agent) as being ‘buggy’, the user is more convinced that the bug emanates from that particular component.

## 6 Evaluation of ZEUS

The key goal of ZEUS was to evolve a methodology for engineering industrial-strength collaborative agent applications, and to construct a tool-kit based on the methodology. While no formal evaluation of the ZEUS system has been performed, we have developed three multi-agent applications with ZEUS to test the system against its functional requirements. The three applications include:

- a travel management application involving 17 agents, in which a travel manager agent negotiates with hotel, airline, car, train and taxi reservation agents to generate an itinerary for a trans-atlantic trip for a user. This was a *3 man-week* exercise, and was the first demonstrator to be built using ZEUS. The project time is in contrast to that of an earlier system, MII (Titmuss *et al.*, 1997) of much less complexity, which took 4 man-years to complete. This comparison is not entirely fair since the 4 man-years included the design time, the time taken to research the prototype, decide on implementation environments, implement an initial rough prototype, etc. However, it suffices to make our point that we are making very significant time gains in building even more complex demonstrators using ZEUS, than from scratch. Fig. 6 shows the personal travel assistant (PTA) application as depicted by one of the visualisation tools.
- a telecommunications network management application with 27 agents. Here agents controlling network links negotiate to provision virtual private networks for users. For this application the report tool was very easily customised to support a different display format for task decomposition graphs. This application took about *2-man weeks* to complete.



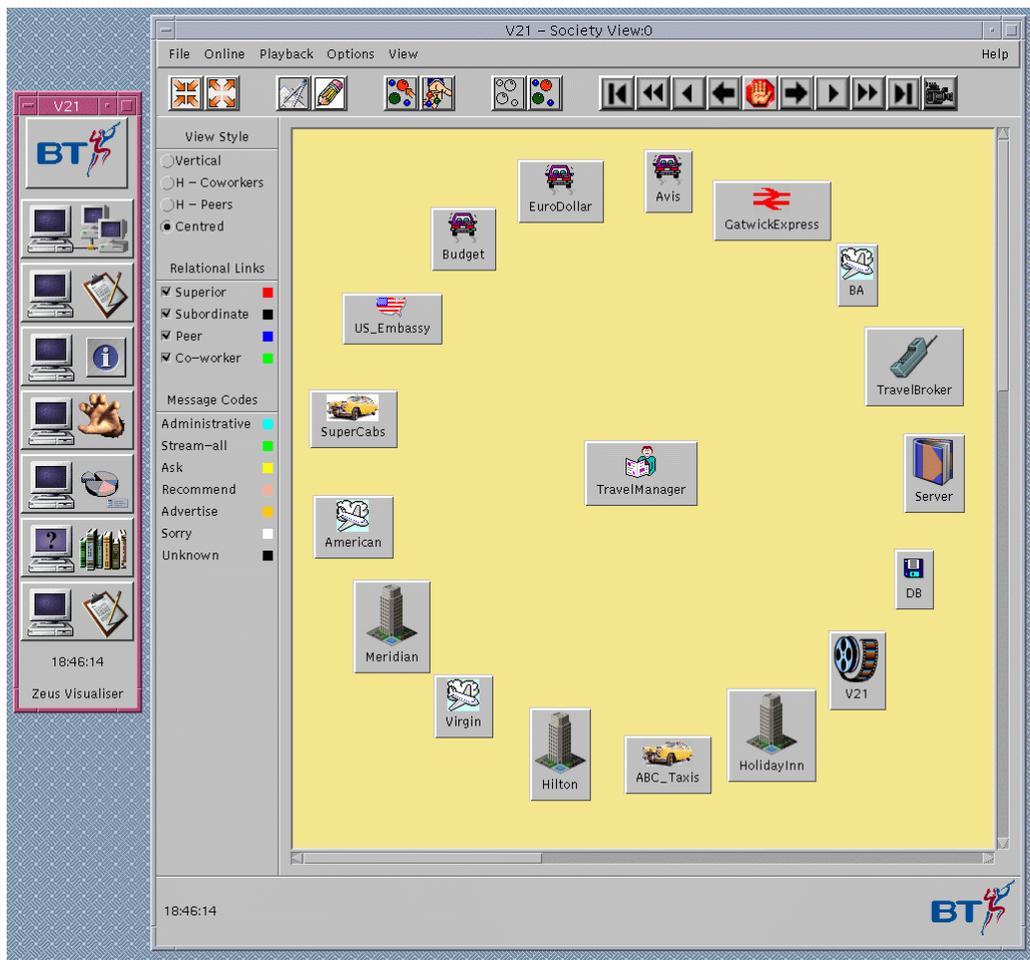
**Fig. 5: A micro tool view of agent C.** From left to right, we have views of the incoming and outgoing mail boxes, the message handler summary, the coordination graph, the execution monitor summary and a list of the items in the agent's resource database. Spanning from left to right at the bottom of the picture is the diary of the agent's current commitments. Currently, it is firmly committed to execute the MakeComputer task at time 8.

- an electronic commerce application where agents buy and sell goods and services using different negotiation strategies. This application involved over 100 agents simultaneously performing many different tasks. This application took about *2 man-months* to complete. Again, this was because during the construction of this demonstrator, we were developing/coding in tandem the auction protocols on which the demonstrator is based.

The key message of this section is that the ZEUS tool-kit concept has been 'proven' by using it to construct several non-trivial, functional and distributed multi-agent demonstrators.

ZEUS will be evaluated more thoroughly on more industrial-strength applications in the next 12 months when several other groups within our organisation use it on other domains. However, our experiences of constructing the demonstrator applications so far suggests that ZEUS *appears* to have met most of its objectives:

- It is scalable, customisable and generic: we have scaled it up to a hundred agents with the agents managing about 2500 goals successfully. It has been used on diverse application domains suggesting it quite generic, and it has allowed for easy customisation in these three domains.



**Fig. 6: The Personal Travel Assistance Demonstrator - in this demonstrator the central agent (the travel manager) plans a trip for the user by negotiating with airline agents, hotel agents and other agents. Once agreement has been reached, the travel manager generates the trip itinerary, emails to its user and telephones the user to notify him that the itinerary has been generated.**

- Judging from the demonstrators we have built, ZEUS clearly speeds up development of distributed multi-agent systems. Although we have not formally measured it, the speed-ups with the demonstrators we have built range *roughly* from 50% to 90%.
- ZEUS clearly encourages code re-use, standardisation and a more structured agent development process. New systems do not have to be built from scratch with ZEUS.

A more rigorous but informal evaluation of ZEUS is about to begin with more people using it. Doubtless, many problems are going to be revealed.

## 7 Related Other Work

There are several other attempts at building agents building environments for agents of differing types and complexities. The ADEPT Project (O'Brien & Wiegand, 1997) produced an agent architecture (or framework) for business process. The RETSINA architecture (Sycara *et al.*, 1996) is a goal-oriented distributed collection of software agents that cooperate asynchronously to perform information retrieval and integration for supporting various decision-making tasks. Work at the University of Toronto on an agent building shell (Barbuceanu & Fox, 1996) is also related to ZEUS. This work describes an agent building

shell “that provides several reusable layers of languages and services for building agent systems: coordination and communication languages, description logic based knowledge management, cooperative information distribution, organisation modelling and conflict management” (p. 235). This work is still much in progress and has not yet resulted in a practical tool-kit embodiment as ZEUS or a coherent architecture like RETSINA, with much effort having been expended on theoretical issues such as description logics, non-monotonic truth maintenance in agent-based systems, and on the extension of KQML to derive the language COOL. The Toronto work clearly identifies and begins addressing some important issues which are identified in Ndumu & Nwana (1997). We borrowed from some of this work, for example, the conversation rules in their COOL language takes a rather top-down approach to managing conversations between agents, while our universal coordination protocol is a more bottom-up design.

There are other attempts at building *test-beds* for multi-agent systems which are relevant to ZEUS, but these are aimed primarily to support the implementation of ideas so that they can be assessed in a meaningful context. Test-beds include DVMT (Lesser *et al.*, 1987), MACE (Gasser *et al.*, 1987) and ARCHON (Wittig, 1992). On the other hand, ZEUS provides a *development environment* as has been described.

Perhaps the closest work to ZEUS is the dMARS system developed at the Australian Artificial Intelligence Institute (AII), which is the most recent C++ implementation of the Procedural Reasoning System (PRS) originally developed in 1987 (Georgeff & Lansky, 1987). Like PRS, dMARS has its conceptual underpinnings in the belief-desire-intention (BDI) model (Bratman *et al.*, 1988), and it provides an environment for developing distributed multi-agent systems. A dMARS application consists of a number of agents residing in one or more processes on one or more host machines. Each dMARS agent conforms to a BDI architecture in a similar way to ZEUS agents conforming to our schema depicted in Fig. 2. In many ways, a ZEUS agent encapsulates the BDI model. Beliefs in ZEUS translate to *facts/beliefs*, desires to *goals*, and intentions to *commitments*. A difference with dMARS is that the BDI model is operationalised via an explicit set of plans. In ZEUS, plans are embodied in the set of tasks we define for realising goals within an agent, and our coordination engine (see Fig. 2) subsumes the functionality of the BDI interpreter of a dMARS agent. ZEUS also adopted dMARS’ principle of visual programming, but ZEUS generates real source code. We also believe the ZEUS methodology is more prescriptive and comprehensive than the BDI approach.

A key point of note is that the dMARS architecture has been deployed in many industrial applications, ranging from air traffic control, space shuttle fault diagnosis to business process management. However, we believe ZEUS subsumes much of dMARS functionality and provides a more *complete* tool-kit for multi-agent system development. For example, we provide many ZEUS-defined coordination mechanisms, a richer agent communications infrastructure and a better visualisation/debugging suite of tools. ZEUS also comes with defaults agents like facilitators, name servers and visualisers. In the next section, we note where we believe we have advanced the current state-of-the-art. The concept of global databases and shared plans in dMARS, in our opinion, remain questionable vis-a-vis real autonomy for agents, even though the developer does not have to use them. We understand that AII is re-implementing dMARS in Java (like ZEUS) and adding some of the functionality currently in ZEUS.

## 8 Conclusions

A project of the complexity as building a generic and customisable tool-kit for constructing collaborative agents truly forces its developers to confront and synthesise many disparate bodies of work. We have found this very rewarding and we certainly encourage such research which synthesises existing work, but which also offers much scope for more meaningful ‘invention of new wheels’. In carrying out this project, we have borrowed unashamedly from a wide and diverse literature including agent communication languages, distributed object technologies, co-ordination/cooperation/negotiation literature, rational agent theories, visual programming, planning & scheduling, visualisation, methodological issues, specification of ontologies, automatic code generation, HCI design, etc. In the same vein, we believe we have pushed the state-of-the-art in these broad areas:

- The derivation of a clear methodology for constructing multi-agent systems;
- The production of a generic, customisable and extensible tool-kit which facilitates and speeds up the development of complex agent applications, and which automatically generates code. We are still to prove that it can support industrial-strength applications like dMARS;
- The visualisation/debugging of complex distributed agent-based systems;
- The derivation and implementation of a universal co-ordination protocol which supports a host of other protocols in the co-ordination literature. This enables ZEUS to be able to support a true marketplace where different agents are using one or more different coordination strategies concurrently as is exemplified in our electronic commerce demonstration.
- We have also recently begun introducing learning primitives into ZEUS.

There are several other advancements we have made in ZEUS but they are hidden in the details of various algorithms we have developed. Nevertheless, there are still many more research and development challenges yet to be addressed and some of these are covered in detail in Ndumu & Nwana (1997).

In conclusion, this paper has described briefly the rationale, philosophy, underlying methodology, design, implementation, evaluation and visualisation of an advanced tool-kit for constructing distributed agent applications. We hope it provides a valuable contribution to this new and exciting area of research.

## Acknowledgements

Jaron Collis, Barry Crabtree, Stuart Soltysiak and Nader Azarmi all contributed to useful discussions which shaped the ideas embodied in the ZEUS tool-kit. This work was funded by BT Laboratories.

## References

- Barbuceanu, M. & Fox, M.S. (1996), The Architecture of an Agent Building Shell”, in Wooldridge, M., Muller, J. & Tambe, M. (eds.), *Intelligent Agents II* **1037**, Berlin: Springer-Verlag, 235-250.
- Bratman, M.E., Isreal, D.J. & Pollack, M.E. (1988), Plans and Resource-Bounded Practical Reasoning, *Computational Intelligence* **4**, 349-355.
- Finin, T. & Labrou, Y. (1997), KQML as an agent communication language, in Bradshaw, J.M. (ed.) *Software agents*, Cambridge, Mass.: MIT Press, 291–316.

- Gasser, L., Braganza, C. & Herman, N. (1987), MACE: A Flexible Test-bed for Distributed AI Research”, In Huhns, M. (ed.), *Distributed Artificial Intelligence: Research Notes in Artificial Intelligence*, 119-152.
- Genesereth, M.R & Ketchpel, S.P. (1994), Software Agents *Commun. ACM* **37**(7), 48–53.
- Georgeff, M.P. & Lansky, A.L. (1987), Reactive Reasoning and Planning, *Proceedings of AAAI87*, Seattle, Washington, 677-682.
- Lesser, V. R., Corkill, D.D. & Durfee, E.H. (1987), An Update on the Distributed Vehicle Monitoring Test-bed, *Computer Science Technical Report 87-111*, University of Massachusetts.
- Ndumu, D.T. & Nwana, H.S (1997), Research and Development challenges for agent-based systems, *IEE Proceedings on Software Engineering* **144** (1), 2-10.
- Ndumu, D.T., Nwana, H.S. & Lee, L.C. (1997), Advanced Visualisation of Distributed Multi-Agent Systems, *Submitted for Publication Consideration to the CHI'98 Conference*
- Ndumu, D.T., Collis, J.C., Nwana, H.S. & Lee, L.C. (1997) The ZEUS Methodology for Building Distributed Agent-Based Applications, Internal Report, ISR, BT Laboratories, Ipswich IP5 3RE, UK.
- Nwana, H.S. Software Agents: An Overview, *The Knowledge Engineering Review* **11**, (3) (1996), 205–244.
- Nwana, H.S. & Ndumu, D.T. (1996), An Introduction to Agent Technology, in Nwana, H.S. & Azarmi, N. (eds.) *Software Agents and Soft Computing: Concepts and Applications*, Lecture Notes in Artificial Intelligence Series, **1198**, Berlin: Springer-Verlag, 3-26.
- O'Brien, P.D. & Wiegand, M.E. (1997), Agents of Change in Business Process Management, in Nwana, H.S. & Azarmi, N. (eds.) *Software Agents and Soft Computing: Concepts and Applications*, Lecture Notes in Artificial Intelligence Series, **1198**, Berlin: Springer-Verlag, 132-145.
- Sycara, K, Pannu, A., Williamson, M. & Zeng, D. (1996), Distributed Intelligent Agents, *IEEE Expert* **11**(6), 36-46.
- Titmuss, R., Crabtree, I.B. & Winter, C.S. (1997), Agents, Mobility & Multimedia Information, in Nwana, H.S. & Azarmi, N. (eds.) *Software Agents and Soft Computing: Concepts and Applications*, Lecture Notes in Artificial Intelligence Series, **1198**, Berlin: Springer-Verlag, 146-159.
- Wittig, T. (1992) (ed.), *ARCHON: An Architecture for Multi-Agent Systems*, London: Ellis Horwood.