

# Agent Factory: An Environment for Constructing and Visualising Agent Communities<sup>1</sup>

G.M.P.O'Hare, R. Collier, J. Conlon and S. Abbas

PRISM Laboratory, Department of Computer Science, University College Dublin (UCD),  
Belfield, Dublin 4, Ireland

Department of Computation, UMIST, PO BOX No. 88, Manchester, M60 12D, UK

**Abstract.** This paper provides an overview Agent Factory, a rapid prototyping environment for Multi-Agent Systems. It examines the agent models and software tools used in this fabrication process. In particular, the paper concentrates upon the Agent Factory Visualiser (AFV) tool – a tool that supports the visualisation of agent communities through the Internet by way of a VRML (Virtual Reality Modeling Language) tool. The functionality of this tool is then animated through a simple virtual robotic demonstrator.

## 1. Introduction

This paper presents *Agent Factory* (AF) a software environment that facilitates the rapid design and delivery of Multi-Agent Systems together with the subsequent experimentation and investigation of agent community behaviour. Within the context of this paper we concentrate on the support offered for the visualisation of agent community behaviour. In section 2 we locate Agent Factory within a broader research landscape. Section 3 describes the Agent Factory environment and briefly outlines the support offered for both the fabrication of agents and visualising their associated behaviour. Section 4 exercises Agent Factory capabilities through a virtual robot case study, while Section 5 reflects briefly on the research presented.

## 2. Belief Desire Intention (BDI)

Much research work has been commissioned on Multi-Agent Systems (MAS) and Distributed Artificial Intelligence (DAI) [8]. Specifically competing agent

---

<sup>1</sup> This research was funded in part by Forbairt Basic Research Grant No.SC/96/621 COMEDY (COMmitment Evolution and DYnamics) and The National Software Directorate under the Programme for Advanced Technology IMPACT (Initiative for Mobile Programmable Agent-based Computing Technology)

architectures have been proposed in the literature. One approach that has gained wide acceptance is to represent the properties of an agent using mental attitudes such as *belief*, *desire*, and *intention*. In this terminology, an agent can be identified as having; a set of beliefs about its environment and about itself; a set of desires which are computational states which it wants to maintain, and a set of intentions which are computational states which the agent is trying to achieve. Multi-agent architectures that are based on these concepts are referred to as BDI-architectures (Belief-Desire-Intention) [10], and have recently been the subject of much theoretical research. Proponents of the BDI approach argue that understanding the dynamics of these mental attitudes and their intimate interdependancies, is crucial in achieving rational agent behaviour.

### 3. Agent Factory

#### 3.1 What is Agent Factory?

In essence Agent Factory is a distributed environment for the rapid prototyping of intelligent agents. More complete descriptions of Agent Factory are presented elsewhere in the literature [9], [7]. Agent Factory has been specified using the Vienna Development Method (VDM) and realised using ObjectShare's implementation of Smalltalk-80 – the VisualWorks integrated development environment.

It is a member of the class of systems that embraces the BDI philosophy [10], [RG92]. The system offers an integrated toolset that supports the developer in the instantiation of generic *agent structures* that are subsequently utilized by a pre-packaged agent interpreter that delivers the BDI machinery. Other system tools support interface customisation and agent community visualisation (section 3.3). In creating an agent community three system components must be interwoven these are *agents*, a *world* and a *scheduler*. Details of each of the components are provided in section 3.2. Initially though we look at the high level architecture.

#### 3.2 Schematic Functional Architecture

Figure 1 outlines the architecture of the Agent Factory (AF) System. Core to AF is a distributed *Component Allocation Table (CAT)* which contains details of all the AF environments (AFEs), and all the components that have been instantiated in those environments. Communication between the CAT and AFEs and any components defined within those AFEs is handled in one of two ways. Firstly intra-AFE communication is addressed via local message-passing, while for inter-AFE communication TCP/IP sockets are commissioned. Thus, AFEs may be located on

any machine that is accessible via the Internet and Agent Factory can thus be described as a truly distributed environment with agents being apportioned to different processors.

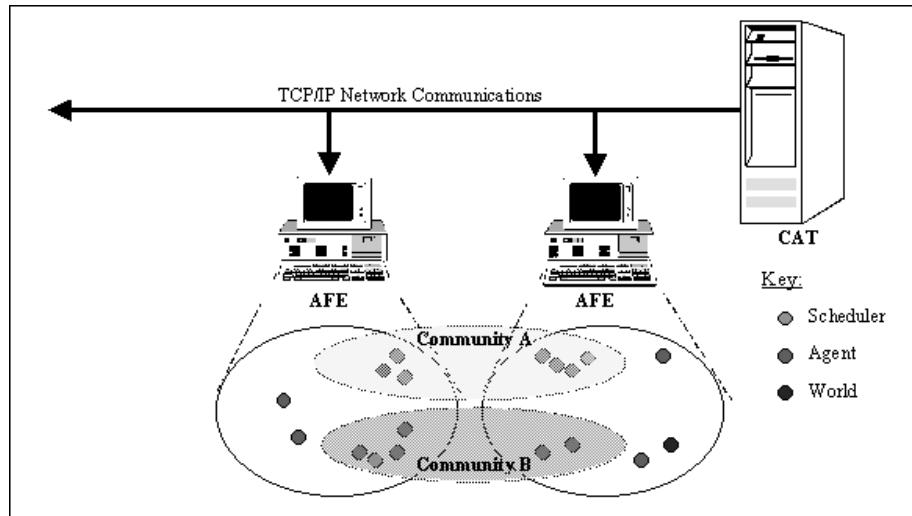


Fig. 1. The AgentFactory System

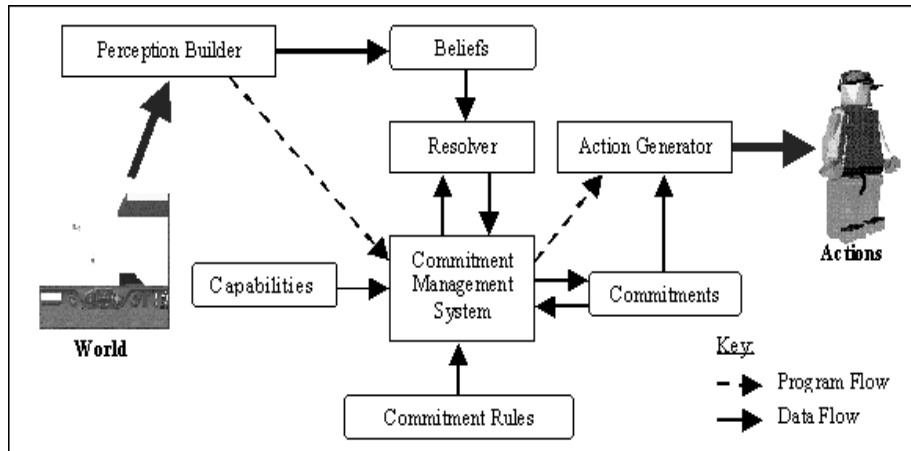
**The Agent Model.** One of the key Agent Factory components is a generic agent model. Agents within Agent Factory are classed as both *social* [6] and *intentional*. The generic agent model has three constituent components, namely a mental state, a communication mechanism (based upon Speech Act Theory [11]) and a model of an agent's acquaintances.

The mental state used by Agent Factory is a *Belief-Commitment-Capability* mental state [13]. This mental state is augmented with a set of *Commitment Rules* that defines specific conditions for the adoption of particular Commitments. Currently, Agent Factory implements a *Single-Minded Commitment Strategy* [10].

**The Agent Interpreter.** The agent interpreter controls the interplay between the mental attitudes identified in the previous section, namely: *Beliefs*, *Commitments*, and *Capabilities*. Figure 2 identifies four key sub-systems:

- **Perception Builder:** This process monitors the agents' environment and accordingly adopts and updates beliefs in accordance with these perceptions.
- **Resolution Theorem Prover:** The resolution theorem prover is used to query the agent's beliefs. Beliefs may be either ground terms or sentences that define how other ground terms may be derived. The resolver uses SLD-Resolution to resolve queries, generating new beliefs under deductive closure.

- **Commitment Management System (CMS).** The CMS manages and monitors commitment adoption and revision. The techniques used within this system are beyond the scope of this paper.
- **Action Generator.** The action generator brings about the sequence of operations necessary to realise an agent's commitments.



**Fig. 2.** The Schematic Agent Factory Architecture

**The Scheduler.** The scheduler is responsible for controlling when, and in what order, the agents under its control should be executed. The scheduler provides a common time frame (using a discrete linear time model) for all agents under its care. Again, a generic scheduler design is provided, allowing the developer to place their own scheduling models upon the community. Component reuse is facilitated via hierarchies of pre-existing schedulers that impose differing scheduling regimes and the designer can merely select that which is appropriate for their particular application.

**The World Model.** All AF agents are *situated* within a world. The world model provides a representation and/or a model of the particular physical/virtual environments which agents may inhabit. Where necessary, the world can act as an interface between the agent and its environment, enabling non-Agent Factory based systems to be integrated into the Agent Factory structure. Currently, a 3-dimensional world model has been developed, enabling agents to be situated within and subsequently navigate around a virtual environment. This physical world metaphor represents but one possible world model.

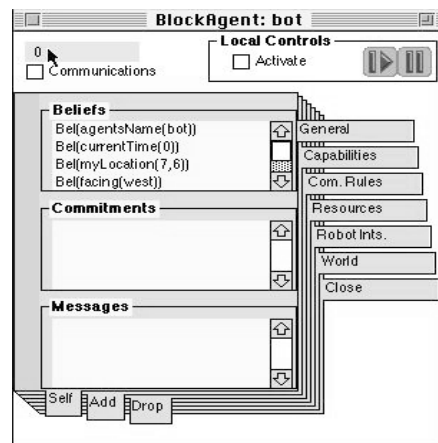
### 3.3 The Agent Factory Toolset

Agent Factory provides a set of tools to assist developers in the rapid prototyping of agents, worlds and schedulers. Below, three tools are introduced: the Agent Viewer

Tool (AVT), the Interface Customisation Tool (ICT), and the Agent Factory Visualiser (AFV). The toolset includes other tools offering a diversity of functionality these include *inter-alia* an *Agent Design Tool (ADT)*, a *network-testing tool*, and an *agent communication tool*. The toolset is wrapped in an integrated support environment providing a supportive medium for the developer at all stages of the development process.

**Agent Instantiation/Viewer Tool.** Figure 3 shows the AVT. This tool allows the user to systematically instantiate agent models, and subsequently view their evolution dynamically thus providing a medium by which agent behaviour can be accessed and indeed accounted for.

This tool adopts a notebook metaphor, with a set of interfaces selectable via tabs to the right of the notebook. Each of these interfaces provides a particular *view* of the agent mental state, for example, the interface in figure 3 allows the developer to select the world the agent is attached to. At the bottom, a list of acquaintances is provided. Upon selection these tabs these unveil, the agents' beliefs pertaining to that acquaintance. This tool and indeed the metaphor supports *information hiding* and allows the designer to view only those aspects that they are interested in at a given instance. Furthermore it provides an *abstraction* mechanism where designers are presented with the mental state in a more palatable form.



**Fig. 3.** Agent Viewer Tool

**Interface Customisation Tool.** The AVT makes use of a set of interfaces to enable the developer of the system to view/change components of the agent. The manner and form that such interfaces take can vary. The developer may wish, for example, to add additional interfaces to the AVT reflecting new or meaningful views of the agent mental state. The Interface Customisation Tool (ICT) helps the developer achieve this. Each interface is developed via the use of an *Interface Hierarchy*, supporting both the reuse of interface components and the incremental development of interfaces.

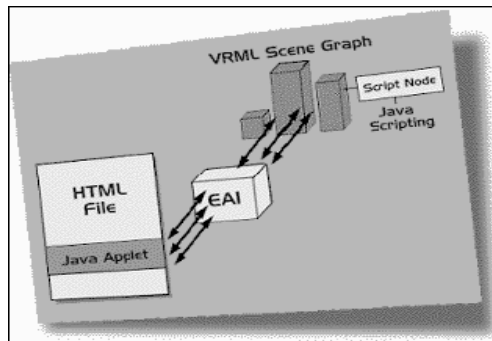
**Agent Factory Visualiser.** The Agent Factory Visualiser (AFV) provides a 3D representation of the 2D world introduced in section 3.2 and represents the focal point of discussion within this paper. It does this through the use of a *proxy server* and the Virtual Reality Modelling Language (VRML) [14].

VRML is a 3D-specification language developed for the Internet. The first specification, VRML 1.0 was released by Silicon Graphics, Inc. in the spring of 1995 and provided the basic Open Inventor file format [1]. VRML 1.0 defines a 3D graphical geometry based on several different classes of *nodes*. Nodes are arranged in hierarchical structures called *scene graphs* that impose an ordering to these collections of nodes. Nodes can be broadly classified under the headings: geometry, transformations, attributes, lighting, shading, and textures.

The VRML 1.0 specification has more recently been updated to VRML 2.0 (now known as VRML97). This newer specification includes facilities for incorporating animated behaviours and significantly more interactive capabilities into VRML. The first version of VRML 2.0 was published in August 1996 [3] [VC197] [VC297].

There currently exists a variety of Browsers/Plug-ins that enable a user to view VRML (via a file known as a world file, and given the postfix .wrl). In particular, the AFV has been tested extensively using Silicon Graphic's CosmoPlayer 2.0.

Of particular interest within the VRML97 specification is the inclusion of the External Authoring Interface (EAI). The EAI acts a communication interface between a VRML world and its external environment – in the shape of a Java applet [5]. It defines a set of functions on the VRML Browser that the Java applet can perform to affect this VMRL environment [4].



**Fig. 4.** EAI Interaction with Java Applet and the VRML Scene

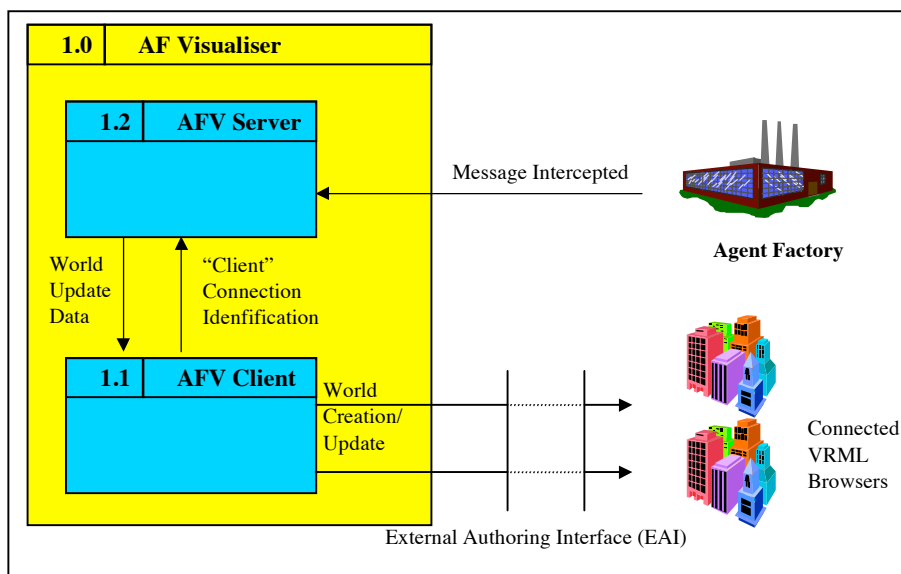
Figure 4 illustrates the relationship between the Java Applet, the EAI, and the VRML scene. The EAI provides the API to allow the Java Applet to interact with the VRML scene. The EAI classes are shipped with an EAI enabled browser, of which CosmoPlayer is the only one presently available [12]. All actions in VRML are called Events. Any input into the scene are called EventIn's and outputs or the resultant scene changes are called EventOuts. The Java EAI classes create an interface between EventIn's / Out's and the scene nodes and can comensurate with these changes updates the attributes of relevant nodes accordingly [16].

So, what can one do with the EAI? For Agent Factory, the key functionality required was:

- the ability to access *named nodes*;
- the ability to create and add VRML to the world from string descriptions and urls;

- and the ability to capture EventOut's and initiate EventIn's to any nodes the interface has access to (access occurs through the first and second abilities);

The Agent Factory Visualiser (AFV) is essentially a proxy server. Users load a Hyper-Text Markup Language (HTML) page in a standard web browser which has a VRML Browser Plug-in (e.g. Silicon Graphics CosmoPlayer 2.0). A VRML World has been embedded within this HTML page, along with a Java Applet. Embedding the VRML World within the HTML Page results in a VRML Browser being embedded in the page allowing the user to view the world. The Java Applet interacts with the VRML browser through the EAI.



**Fig. 5.** The Agent Factory Visualiser

The Java Applet (the AFV Client) communicates via TCP/IP with a proxy server (the AFV Server). This server captures events occurring in Agent Factory, and sends update commands to the AFV Clients currently connected to it. This design is illustrated above in Figure 5.

Having introduced the operation of the AFV the subsequent section will now illustrate the usage of the tool and indeed the Agent Factory system.

#### 4. Ro and Bot: A Virtual Robotic Demonstrator

A virtual robotic demonstrator was designed and developed based around two Virtual Robots: *Ro* and *Bot*. These robots are based on a simple design that depicts

agents moving and interacting with a virtual (VRML) world, as well as each other. This research through the use of Agent Factory has sought to develop robot agents that are *rational*, *social* and *intentional*. This demands that the robots must have a much more sophisticated representational model than that typically ascribed to such reactive entities. This model must encompass a robot *mental state* that reflects its beliefs both about the environment within which it is situated and itself. This contrasts somewhat with the *de facto* agent architecture namely the *subsumption architecture* proposed by Brooks [2]. To this ends we have utilised Agent Factory to prototype a virtual robotic demonstrator.

#### 4.1 Robot Creation

In creating our virtual robot agents we again adopt a reuse metaphor. *Ro* and *Bot* are based on two distinct designs: a *MoveAgent* and a *BlockAgent*. Both represent an encoding of trivial behaviour patterns. In the case of the former move agents keep moving in the direction they currently face until they encounter an obstruction whereupon they stop. In the case of the latter they move until they encounter an obstacle whereupon they push it until the obstacle encounters another obstacle. These designs represent a specialisation of a more generic agent class namely a *VRMLAgent*. This in turn is a specialisation of *RobotAgent* and *Agent* in general. Thus we can see that an *Agent Design Hierarchy* exists which offers a taxonomy of preexisting agent designs that can be reused in the construction of new agent communities.

As part of the creation process, every agent must be deposited within a community (a scheduler). Upon creation, the developer must also situate them within a particular world. Specifically the robot must be positioned and oriented within this world. This is achieved via an interface in the Agent Viewer Tool (see section 3.3 and figure 3). In addition to setting up this mapping between the world and agent, the developer must also position and orient the agent in the world. This is achieved through a second interface in the AVT.

Figure 6 above shows the initial set up of the world. Here, the two agents *Ro* and *Bot* are positioned at (6,10) and (7,6) respectively, and two obstacles and three partitions have been added. In fact, the figure below shows the state of the world at time t1. That is, after one iteration of the scheduler. Hence, the agents below have beliefs, gathered through perception of their environment.

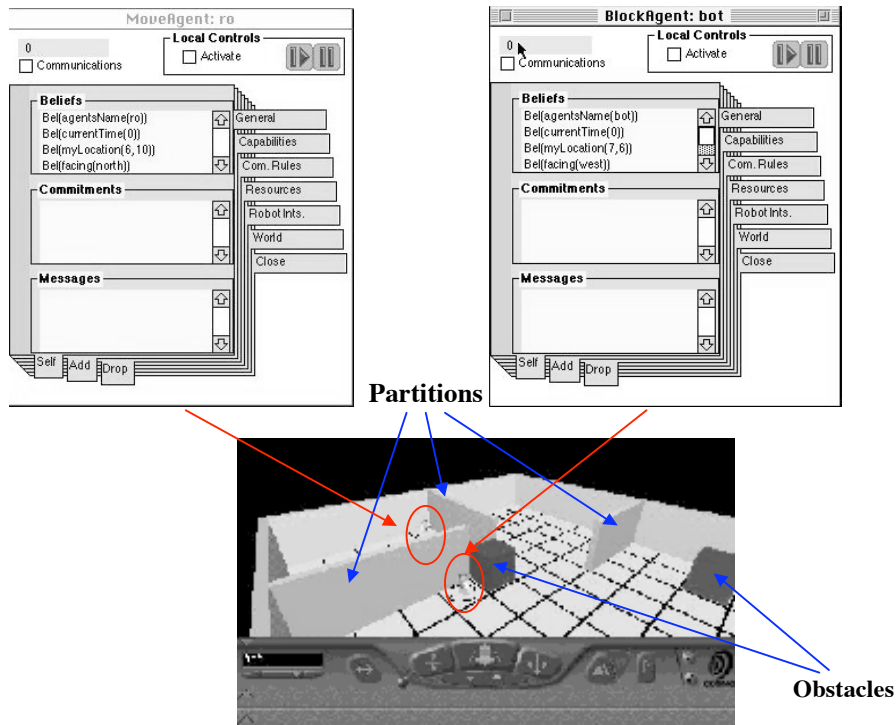


Fig. 6. A First Glance at a World

## 4.2 Robot Deliberation

Once the agents have been created and connected to a world, the next stage is the activation/execution of the community. As indicated in section 3.2.3, the scheduler uses a discrete linear model of time. The current model only increments the clock when *each* and *every* agent within a given community has been scheduled, that is all commitments for that particular time grain honoured.

In figure 7 below, the agent *Bot* and the corresponding view of *Bot* are shown at time  $t_8$ . The diagram shows the AVT displaying the *beliefs*, *commitments*, and *messages* of the agent (see sections 2 & 3.2 for descriptions of beliefs and commitments). In particular, the diagram identifies three beliefs:  $Bel(myLocation(5,6))$ ,  $Bel(facing(west))$ , and  $Bel(seen(box1,at(4,6)))$ . The standard *Bel* epistemic operator is used. Given that north is defined as reducing values of  $y$  (i.e.  $(3,4)$  is north of  $(3,6)$ ), it can be seen that *box1* is in front of the agent (as *box1* is 1 square west of the agent), and that this situation is represented accurately in the VRML world.

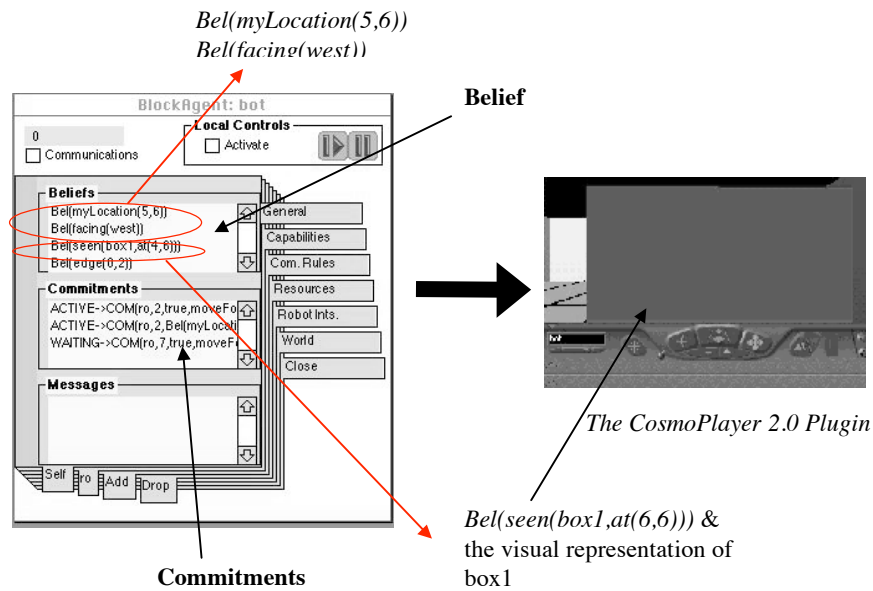


Fig. 7. Bot Moving Box1

The agent *Ro* has made a commitment to move forwards until it reaches an obstacle. In figure 8, it can be seen that the agent *Ro* has finally reached an obstacle - a wall. In fact, the view given in figure 8 is that of *Ro* with a west orientation and not the direction *Ro* is actually facing. This view shows the agent *Bot* and the second obstacle (*box2*).

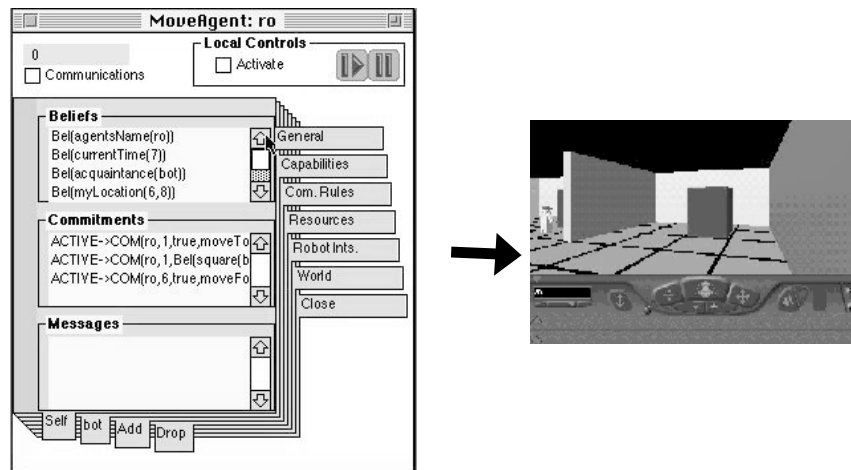


Fig. 8. Ro Reaches a Wall

In addition, the figure indicates that the agent has a commitment to move to a location. The agent also has sub-commitments (a plan) that realise this higher commitment. Further, one can see that these commitments are in one of a variety of possible *states* indicating the current progress of the commitment.

### 4.3 Robot Visualisation

The Agent Factory Visualiser provides a mechanism for displaying diverse representations of robots in VRML. In this demonstrator, the agents are represented in VRML by the *legoman* object. Of course other unique objects could be selected from an object portfolio and differing agent behaviour types selected from the *Agent Design Hierarchy* on to these. The result would be *avatars* performing collaborative actions in a shared workspace with the robots experiencing real-time shared behaviour.

For each agent, that agent's name is placed around the base of the *legoman*, and in some cases, a different coloured *legoman* is used (this kind of detail may be useful at either an individual, or group level of agents in the community).

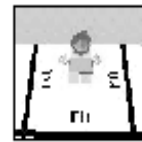


Fig. 9. Robot Ro

The ability to represent an agent in a variety of forms and colours provides a rich medium for the visualisation of multi-agent systems. Alternate predefined views are also provided which enable the user to see the world from an aerial view and the view of each of the robot's occupying the world. Of course the plug in enables the user to determine their preferred view at any given instance.

## 5. Discussion

Within the context of this paper we have introduced Agent Factory a rapid prototyping environment for multi-agent systems. We have focused upon the toolset this system offers and specifically considered the visualisation capabilities afforded to support the understanding of agent behaviours. This system represents a significant step forward in that it provides a robust multi-agent fabrication facility together with a rich set of visualisation tools that provides invaluable support for the understanding and subsequent engineering and fine tuning of community interactions. The marriage of BDI based techniques with virtual reality viewing tools that supports browser based observation is we feel a first.

This virtual robotic demonstrator illustrates one possible use of VRML, however, VRML could be used to represent other less obvious worlds such as information spaces, virtual market spaces and so forth. The PRISM Laboratory is about to take receipt of three Nomad Scout II robots that will be equipped with Mercury RF1

transmitters and on board laptops thus supporting inter robot communication and local autonomy. Future work will investigate the experimentation with BDI controlled robot behaviours in the virtual robot world with the subsequent trials of these robot controlled behaviours in the real world. Presently Agent Factory generated events are communicated to the AFV via a LAN. In the real robot world Agent Factory Agents residing on the laptops on the robots would control the physical movement of and sensory interpretation from the robot. Agent Factory events would thus be transmitted in a wireless mode. The seamless translation from the virtual to the real world offers a compelling opportunity to evaluate deliberative architectures for robot control.

## Acknowledgements

We gratefully acknowledge the support of Forbairt through Basic Research Grant No.SC/96/621 COMEDY (COMmitment Evolution and DYnamics) and The National Software Directorate under the Programme for Advanced Technology IMPACT (Initiative for Mobile Programmable Agent-based Computing Technology), without whos kind support this research would not have been possible.

## References

- [1] Bell, G., Parisi, A., Pesce, M., - The VRML Consortium, (1996), VRML 1.0 Specification, *url: <http://www.vrml.org/Specifications/VRML1.0/>*
- [2] Brooks, R. A., A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, 2: pp14-23, 1986.
- [3] Carey, R., Bell, G., Marrin, C. - The VRML Consortium, (1997), VRML Specification ISO/IEC 14772-1, *url: <http://www.vrml.org/Specifications/VRML97/>*
- [4] External Authoring Interface Working Group, (1997), *url: <http://www.vrml.org/WorkingGroups/>*
- [5] Kimen, S., (1997), VRML Is – Java Does – and the EAI can help. *url: <http://cosmosoftware.com/eail/java.html>*
- [6] Moulin, B. and Chaib-draa, B. (1996), An Overview of Distributed Artificial Intelligence, in Foundations of Distributed Artificial Intelligence (G.M.P. O'Hare and N. Jennings eds) pp 3-57, John Wiley and Sons, Inc.
- [7] O'Hare, G.M.P. and Abbas, S., Commitment Manipulation within Agent Factory, *Proceedings of Decentralised Intelligent and Multi-Agent Systems, DIMAS '95, 22-24 Nov. 1995, Cracow, Poland.*
- [8] O'Hare, G.M.P. and Jennings, N.R. (Editors.), Foundations of Distributed Artificial Intelligence, *Sixth Generation Computer Series, Wiley Interscience Publishers, New York, 1996, 296 pages. ISBN 0-471-00675*
- [9] O'Hare, G.M.P., "The Agent Factory: An Environment for the Fabrication of Distributed Artificial Systems", In O'Hare, G.M.P. and

- Jennings, N.R.(Eds.), *Foundations of Distributed Artificial Intelligence*, *Sixth Generation Computer Series*, Wiley Interscience Publishers, New York, pp 449-484, 1996.
- [10] Rao, A.S. and Georgeff, M.P. (1991), Modeling Rational Agents within a BDI Architecture, in Proceedings of Second International Conference on Principles of Knowledge Representation and Reasoning, (J.Allen, R.Fikes, and E.Sandwall eds) pp 473-484, Morgan-Kaufmann, San Mateo, CA.
  - [11] Searle, J.R. (1969), *Speech Acts*, Cambridge University Press, Cambridge, UK.
  - [12] Silicon Graphics Inc., (1997), *Creating Interactive Java Applications with 3D and VRML*, *url: <http://cosmosoftware.com/developer/>*
  - [13] Shoham, Y., (1993), Agent-Oriented Programming, *Artificial Intelligence (60)*, pp 51-90.
  - [14] Vacca, J.R., VRML: Bringing Virtual Reality to the Internet, *Academic Press*, 1996.
  - [15] The VRML Consortium, (1997), *url: <http://www.vrml.org/>*
  - [16] The VRML Consortium, (1997), Standards and Specifications, *url: <http://www.vrml.org/Specifications/>*