IMPLEMENTATION OF A TILEWORLD TEST-BED ON A DISTRIBUTED BLACKBOARD SYSTEM

K.W. Choy, A.A. Hopgood, L. Nolle, B.C. O'Neill
The Nottingham Trent University
School of Computing and Technology,
Burton Street,
Nottingham NG1 4BU
United Kingdom
E-Mail: {kw.choy, adrian.hopgood, lars.nolle, brian.oneill}@ntu.ac.uk

KEYWORDS

TileWorld, multi-agent simulator, distributed blackboard system, DARBS

ABSTRACT

Research into multi-agent systems has increased over several years, leading to demand for multi-agent system simulators to investigate the interaction between different agents and their environment. The more complex and intelligent the agents become, the more difficult it is to simulate them. Distributing the simulation across different processors would improve its performance. This paper describes the implementation of a multi-agent test-bed called TileWorld on a distributed blackboard system, DARBS. The slow-down of the simulator on a single processor has been measured as the number of agents increases, and the likely effect of migrating to a multiple-processor system is assessed.

1. INTRODUCTION

Research in multi-agent systems is currently on the increase. Investigating the behaviour of the agents' interaction with each other and with the environment is inevitably part of this research. There is a need for a test-bed to simulate the agents in an artificial environment that can be both dynamic and static. TileWorld (Pollack and Ringuette 1990) is a well-established test-bed for agent systems and MA-TileWorld (Ephrati et al. 1995) is its multi-agent version.

Unfortunately, this type of simulation becomes processor-intensive as the number of agents is increased. An ideal solution to this would be to distribute the agents across different processors. A distributed blackboard system such as DARBS (Distributed Algorithmic and Rule-based Blackboard System) (Nolle et al. 2001) would be a suitable architecture.

1.1 TileWorld Test-bed

From here onwards, MA-TileWorld will simply be referred to as TileWorld. In this test-bed, the world is set up as a two dimensional grid. There are agents, tiles, holes, and obstacles in the TileWorld. The objective of the agents is to score as many points as possible. They score points by moving around the TileWorld to find and pick up tiles which they put into holes. The agents have a limited view of the TileWorld. The viewing radius is a variable that can be set. For example, in the DARBS TileWorld test-bed, the agents have a viewing radius of 5 cells. Each cell can only be occupied by one agent at a time. Agents cannot move to a cell with an obstacle in it. An example of a 10 × 10 TileWorld is shown in Figure 1.

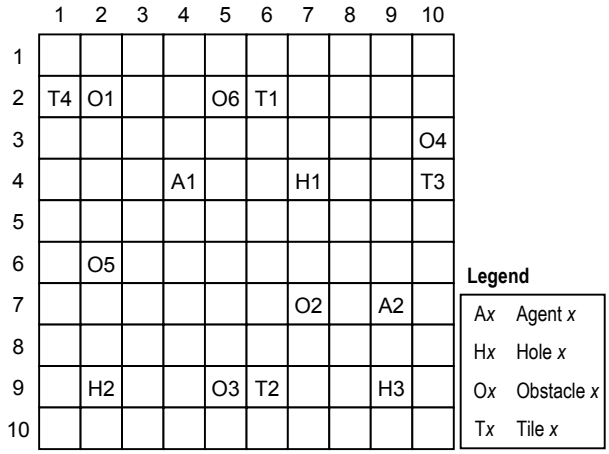


Figure 1: A 10 × 10 TileWorld

The holes, obstacles, and tiles in the TileWorld can change dynamically, i.e. they can appear and disappear in different locations in the TileWorld. The rate of change is set by a variable, and this can be used to reflect the dynamically changing real-world environment. The TileWorld test-bed has been widely used to test the behaviour and interaction of multiple agents in a dynamic environment (Lees et al. 2003; Kinny et al. 1992). There is also a variant of the original TileWorld test-bed that includes "gas station" objects to top up the resources of the agents (Uhrmacher and Schattenberg 1998). In this variant, the agent's resource-management skill is investigated by making each move consume fuel. Carrying a tile would cause the agent to consume more fuel. Therefore the agent would need to balance the consumption of resources with scoring points.

Proceedings 18th European Simulation Multiconference
Graham Horton (c) SCS Europe, 2004
ISBN 3-936150-35-4 (book) / ISBN 3-936150-36-2 (CD)

1.2 DARBS

A blackboard system is an artificial intelligence (AI) technique that is analogous to a team of experts who communicate their ideas by writing them on a blackboard (Engelmore and Morgan 1988). The experts are represented by sets of rules, conventional procedures, neural networks, or other program modules. These modules are termed knowledge sources (KS). The blackboard is an area of global memory containing evolving information. The system's current state of understanding of a problem is stored here as it develops from a set of data towards a conclusion. DARBS is a distributed blackboard system developed at the Open University and the Nottingham Trent University (Nolle et al. 2001). The original non-distributed blackboard system was called ARBS and was run on a single processor machine (Hopgood et al. 1998).

DARBS was selected for this work as it is a research-based distributed blackboard system that does not have a central control module. Commercially available distributed blackboard systems were not considered owing to their cost and the limitations on access to their source code.

In DARBS, the blackboard system is modelled on the client/server model where the blackboard (BB) is the server and the KSs are the clients. Therefore, DARBS can be seen as a distributed blackboard system with the blackboard server running on one PC and other KS clients running on different PCs. This allows different KSs to run in parallel and thus to be truly opportunistic (Engelmore and Morgan 1988). For this reason, DARBS may also be considered as a multi-agent collaborative system (Jennings and Wooldridge 1998). Different KS clients on different PCs can work independently, in keeping with the TileWorld test-bed where each agent can be run as a separate KS client

DARBS does not have a control module and as such uses broadcast messages to inform other KSs that a change in a particular partition of the blackboard has occurred. It is then up to the KS to decide what to do. It may either stop what it is doing and check exactly what has changed or it may finish what it is doing and then check.

1.3 Design criteria

One of the benefits of the blackboard architecture is that there is central storage of information about the current problem. All KSs store their working memory on the blackboard, visibly to other KSs or users. New KSs can then be developed that can make use of this information or the information can be use for debugging purposes. A balance is needed, as too little information on the blackboard defeats the purpose of central storage and too much information would create a great communication overhead. Partitioning the information on the blackboard can help, but again there is a balance. Too much information on a single partition would slow down the blackboard server in searching for the information, and too many partitions would cause the KS clients to monitor more partitions for changes.

The format of the information stored on the blackboard is also important. The more intelligible the information, the greater the amount of data that is used to store it and the more data the communication channel has to send. Also putting the information in a format that is difficult to search or query would slow down the overall system. With all these criteria in mind, TileWorld was designed and implemented on DARBS.

2. DESIGNING TILEWORLD ON DARBS

The natural way the TileWorld test-bed fits a blackboard system made it ideal for DARBS. In the TileWorld environment all the agents and other objects interact with each other and hence the agents can be directly implemented as KS clients and the world itself can reside on the blackboard. This gave all the agents equal access to view and change the world, thus allowing true parallel agent simulations. The setup of the TileWorld on DARBS is as shown in Figure 2. The *Initiator KS* is the KS that sets up the TileWorld with its parameters and generates the world. The *Display TileWorld KS* displays the TileWorld and its contents in a graphical format so that it is easy for users to view the simulation. The display KS also needs to keep track of the changes in the world and make sure that the graphical representation is as up-to-date as possible. Each *Agent KS* controls their respective agent on the TileWorld and makes changes to the world according to the behaviours that are coded in them.

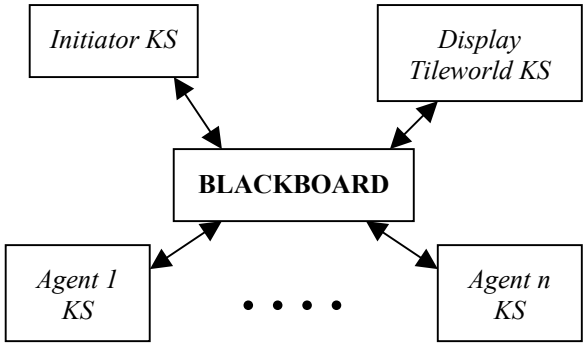


Figure 2: TileWorld on DARBS

To simplify the design, the objects in the TileWorld (i.e. tiles, holes, and obstacles) cannot move themselves. However, it is possible to make the objects in the TileWorld dynamic, i.e. appear and disappear with time by simply adding another KS that uses a probability rate to change the number and position of tiles, holes, and obstacles in the TileWorld. A benefit of the blackboard architecture is that new KSs can easily be added to provide more agents or to change the way the TileWorld is being simulated.

Careful organisation of the data on the blackboard is required to make sure that the agents and all other KSs can interact with each other properly through the blackboard. The organisation of the data also needs to minimise the number of partitions with which a particular KS works. This is to reduce the number of times the KS needs to restart, since a KS restarts whenever a partition that is it working with has changed. As mentioned earlier, the format of the data on the blackboard also needs to be carefully constructed so that queries from the KS clients are as easy, efficient and legible as possible. With these criteria in mind, the blackboard was partitioned as shown in Figure 3. The most important data string format is the one on the TileWorld Environment partition, for example:

[Location 1 , 4 contains Agent 1 , NO HOLE , NO OBSTACLE , NO TILE]
[Location 1 , 5 contains NO AGENT , Hole 1 , NO OBSTACLE , NO TILE]
[Location 1 , 6 contains NO AGENT , NO HOLE , Obstacle 1 , NO TILE]
[Location 2 , 4 contains NO AGENT , NO HOLE , NO OBSTACLE , Tile 1]

The order of objects is fixed as Agent-Hole-Obstacle-Tile to facilitate queries from the KSs. If the location contains an agent then "Agent *x*" where *x* is the number of the agent would replace "NO AGENT", and similarly for holes, obstacles and tiles.

As can be seen in Figure 3, all Agent KSs can access the TileWorld Environment partition and, in theory, can see the whole TileWorld. This could be argued to be an incorrect implementation of the TileWorld test-bed, but it is assumed that all agents are benevolent and will only access those areas of the TileWorld Environment partition that they are intended to.

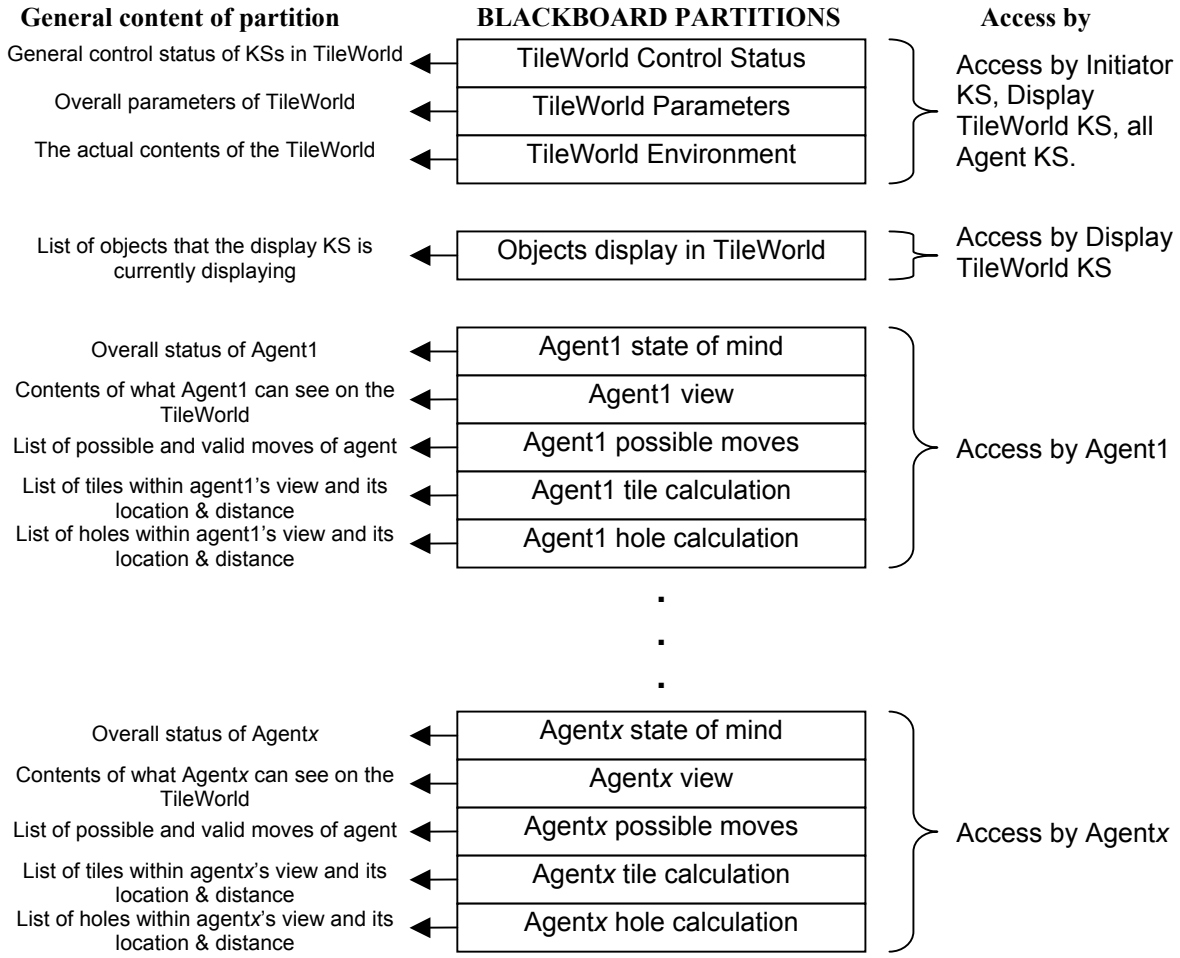


Figure 3: Partitions on the Blackboard

3. IMPLEMENTING THE TILEWORLD

Three types of KSs were implemented in DARBS for this TileWorld test-bed: *Initiator KS*, *Display TileWorld KS*, and *Agent KS*. A screen capture of the TileWorld running on DARBS is shown in Figure 4. Each is implemented as a rule-based KS, described below.

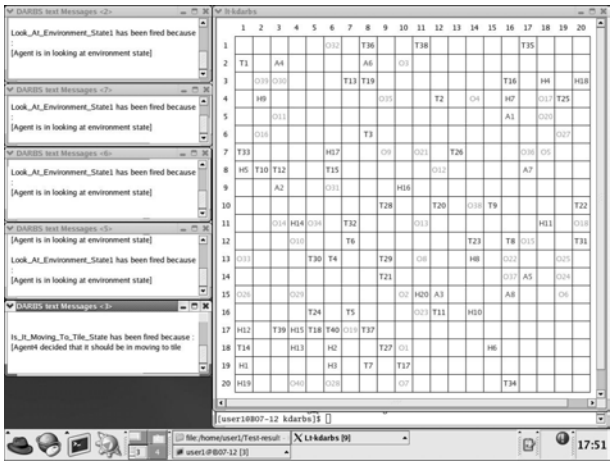


Figure 4: Screen Capture of TileWorld on DARBS

3.1 Initiator KS

The main purpose of the Initiator KS is to setup the parameters of the TileWorld and to generate the world from the parameters. The two main rules of this KS and their functions are:

- Init_TileWorld
  - Set the size of the TileWorld
  - Set the number of agents in the TileWorld
  - Set the number of holes in the TileWorld
  - Set the number of obstacles in the TileWorld
  - Set the number of tiles in the TileWorld
- Create_TileWorld
  - Generate a random position within the TileWorld for each of the agents, holes, obstacles, and tiles.
  - Store this information on the blackboard.

The FIRABILITY_FLAG of this KS is set to true, which means that this KS will run only once unless a change occurs in the partition in which it is interested.

3.2 Display TileWorld KS

The main purpose of the *Display TileWorld KS* is to display the content of the TileWorld in a graphical form. This is done with the help of the Qt library from Trolltech (Dalheimer 2002). The Qt library provides the graphical function calls for drawing and updating the graphical TileWorld with its agents, holes, obstacles and tiles. The rules in this KS and their functions are as follows:

Display_Initial_Screen
  - Draw the TileWorld grid and label it

- Update_Agent_Display
  - Find the location of all the agents in the TileWorld from the blackboard and display them accordingly.
- Update_Hole_Display
  - Find the location of all the holes in the TileWorld from the blackboard and display them accordingly.
- Update_Obstacle_Display
  - Find the location of all the obstacles in the TileWorld from the blackboard and display them accordingly.
- Update_Tile_Display
  - Find the location of all the tiles in the TileWorld from the blackboard and display them accordingly.
- Update_Total_Objects_Display
  - Find the objects that are currently being display in the TileWorld.
- Update_Deleted_Tile
  - Delete the objects that are no longer on the blackboard from the displayed TileWorld.

The FIRABILITY_FLAG of this KS is also set to true.

3.3 Agent KS

The Agent KSs control the agents in the TileWorld. There is a set of 31 rules for each agent in the TileWorld. The behaviour and intelligence of the agent are coded in the following rules (all rules are listed here, and the function of selected rules is explained):

- Initialise_Agent
- Update_Internal_Status
- Generate_SearchSpace_State
- Look_At_Environment_State1
- Look_At_Environment_State2
- Is_It_Exploring_State
  - Change the agent state to Exploring state if the agent is in Thinking state and the agent is currently carrying no tile and there is no tile within the viewing range OR
  - Change the agent state to Exploring state if the agent is in Thinking state and the agent is currently carrying a tile and there is no hole within the viewing range.
- Is_It_Moving_To_Tile_State
  - Change the agent state to Moving To Tile state if the agent is in Thinking state and the agent is not carrying a tile and there is a tile within the viewing range and the tile is not on the same grid as the agent.
- Is_It_Hole_Filling_State
- Is_It_Moving_To_Hole_State
  - Change the agent state to Moving To Hole state if the agent is in Thinking state and the agent is carrying a tile and there is a hole within the viewing range and the hole is not on the same grid as the agent.
- Is_It_Picking_Up_Tile_State

- Generate_Possible_Moves
- Is_North_Move_Valid
- Is_North_Move_NotValid
- Is_East_Move_Valid
- Is_East_Move_NotValid
- Is_South_Move_Valid
- Is_South_Move_NotValid
- Is_West_Move_Valid
- Is_West_Move_NotValid
- Exploring_State
  - Generate a random step based on the possible moves and the last made move. Store the random generated step on the agent's state of mind partition. Change agent's Exploring state to Making A Move state.
- Moving_To_Tile_State
- Moving_To_Hole_State
- Get_Tile_Distance_State
- Get_Hole_Distance_State
- Find_Closest_Tile_State
- Find_Closest_Hole_State
- Generate_Step_Closer_To_Tile_State
- Generate_Step_Closer_To_Hole_State
- Pick_Up_Tile_State
- Hole_Filling_State
  - Drop the tile it is carrying into the hole it is standing on and recalculate the agent's score. Change agent's state back to Generate Searchspace state.
- Making_Move_State

The FIRABILITY_FLAG of this KS is set to true_always, which means that this KS runs continuously and restarts if a change occurs in the partition in which it is interested.

4. TESTS AND RESULTS

A preliminary test has been carried out on a single Intel Pentium 4 2.8GHz processor with 1GB of DDR RAM running the Red Hat 9 Linux operating system. All the agents in the TileWorld have a viewing radius of two cells. An 18 × 18 TileWorld was run with 30 obstacles, 15 tiles, and 25 holes. The position of the obstacles, tiles, and holes, and the initial position of the agents were all randomly generated using the C++ standard random number generator function, rand() with a seed of 10. The TileWorld test-bed was run with one, two, three, four, and five agents. For all the runs, the average time taken for an agent to make a move was calculated over 10 moves. The time per move was calculated by subtracting the time of an agent's move from the time of its subsequent move. An agent is considered to make a move when it has changed the TileWorld environment (i.e. moved to another cell, picked up a tile, or dropped a tile into a hole). Restarts due to the TileWorld being changed by other agents are not considered as moves. Figure 5 shows the average time per move normalised to a single-agent TileWorld's average time per move.

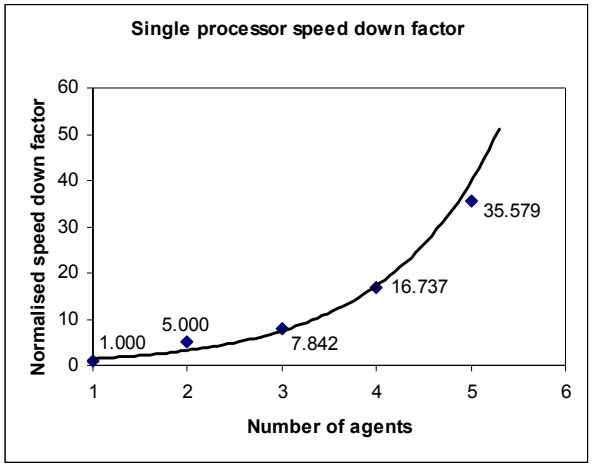


Figure 5: Normalised Single Processor Slow-down Factor

In Figure 5, an exponential function is fitted to the results of the agent simulation on the TileWorld. This shows that the slow down is approximately exponential as the number of agents being simulated increases. This is as expected, as the single processor has to time-slice between more processes as the number of agents increases. Another reason for the slow down is that as the number of agents increases, the number of restarts each agent has to make increases, thus increasing the time required for each agent to make a move. This can be improved by having more intelligent restarts, for example by checking whether changes to the partition affect the current thinking of the agent, and if so to what extent, before deciding to restart again.

A multi-processor test is currently in preparation. This is expected to show a less marked slow down in simulation as the number of agents increases. It is inevitable that there will be some slow down as the number of agents increases because there will be communication overheads and access contention between agents for the blackboard. One way of reducing this is to reduce the amount of information stored on the blackboard. However, this would conflict with the concept of openness of information on the blackboard, which provides upgradeability and helps users to understand what is happening in the simulation. Therefore a balance needs to be struck between the amount of information to be stored on the blackboard and the speed of the simulation.

5. CONCLUSIONS AND FUTURE WORK

The TileWorld test-bed has been successfully implemented on DARBS running on a single PC using the Linux operating system. It is expected that the slow performance will improve when this system is run in parallel on separate PCs connected together via TCP/IP as the load of the extra agents would be distributed over

the separate PCs. The speedup factor between *n* agents in a single processor and *n* agents in multi-processors can then be compared and evaluated. Further tests will be carried out to investigate the causes of slowness and, where possible, improvements to the system will be made. One cause of slowness is that the KS tries to fire all the rules even when certain rules are known beforehand to be dependent on other rules. A rule dependency table (Hopgood 1994) can be created before runtime to prevent the KS from trying to fire any rule until the rules upon which it depends have fired.

An improvement for this TileWorld test-bed would be to have another KS that relays what the agent sees from the TileWorld Environment partition onto the Agent View partition. In this way, the agent would have no direct access to the TileWorld Environment partition other than making a move on the TileWorld. This would make it easier to implement intelligent restarts as the Agent KS would only need to restart on changes to the Agent View partition and could ignore changes to the TileWorld Environment partition.

Another consideration is the way the KS checks whether a rule's condition is met. The current implementation checks every sub-condition before evaluating whether the composite condition is true. This can be improved on by evaluating the composite condition as it checks each sub-condition. Consider the following example:

IF
[
    condition1
    AND
    condition2
    AND
    condition3
]
THEN
[
    actions
]

The KS can stop checking *condition2* and *condition3* if *condition1* is false because the composite condition will be false. This type of on-the-fly evaluating would reduce the number of messages sent to the blackboard as each condition-check requires the KS to send a message to the blackboard. The use of on-the-fly evaluation can reduce the communication traffic, thus allowing other KSs to send and receive messages faster.

REFERENCES

Dalheimer, M.K. 2002. "Programming with Qt". Germany: *O'Reilly*.

Engelmore, R.; Morgan, T.; ed. 1988. "Blackboard Systems". Great Britain: *Addison Wesley*, pp. 2-15.

Ephrati, E.; Pollack, M. E.; Ur, S. 1995. "Deriving multi-agent coordination through filtering strategies" *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 679-685.

Hopgood, A.A. 1994. "Rule-based control of a telecommunications network using the blackboard model", *Artificial Intelligence in Engineering*, 9, pp 29-38.

Hopgood, A.A.; Phillips, H.J.; Picton, P.D.; Braithwaite, N.St.J. 1998. "Fuzzy logic in a blackboard system for controlling plasma deposition processes" *Artificial Intelligence in Engineering*, 12, pp. 253-260.

Jennings, N. R.; Wooldridge, M. J.; ed. 1998. "Agent Technology: Foundations, Applications, and Markets". Germany: Springer, 1998. pp. 32-34.

Kinny, D.; Georgeff, M.; Hendler, J. 1992. "Experiments in Optimal Sensing for Situated Agents", *Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence*, PRICAI'92, pp. 1176-1182.

Lees, M.; Logan, B.; Theodoropoulos, G. 2003. "Adaptive Optimistic Synchronisation For Muti-Agent Distributed Simulation", *Proceedings 17th European Simulation Multiconference*, pp. 77-82.

Nolle, L.; Wong, K. C. P.; Hopgood, A. A. 2001. "DARBS: A Distributed Blackboard System", *Research and Development in Intelligent Systems XVIII*, Bramer, Coenen and Preece (eds.), Springer, pp 161–170.

Pollack, M. E., and Ringuette, M. 1990. "Introducing the Tileworld: Experimentally Evaluating Agent Architectures", *Proceedings of the Eighth National Conference on Artificial Intelligence*, AAAI Press, pp. 183-189.

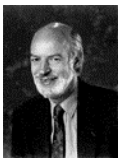
Uhrmacher, A.M.; Schattenberg, B. 1998. "Agents in Discrete Event Simulation", In: Andre Bargiela and Eugene Kerckhoffs (eds.) *Proceedings of the 10TH European Simulation Symposium ``Simulation in Industry – Simulation Technology: Science and Art'' (ESS'98)*, SCS Publications, Ghent, pp. 129-136.

AUTHOR BIOGRAPHIES

KUM WAH CHOY is a PhD student at the Nottingham Trent University. He obtained his BEng (Hons) in Electronics & Computing in 2001. He has spent a placement year with Xerox Ltd as a Software/Test Engineer. His current research is in the field of distributed embedded systems, agent system, blackboard systems, and artificial intelligence.

ADRIAN HOPGOOD is professor of computing and head of the School of Computing and Technology at the Nottingham Trent University, UK. He is also a visiting professor at the Open University. His main research interests are in intelligent systems and their practical applications. He graduated with a BSc (Hons) in physics from the University of Bristol in 1981 and obtained a PhD from the University of Oxford in 1984. He is a member of the British Computer Society and a committee member for its specialist group on artificial intelligence.

LARS NOLLE graduated from the University of Applied Science and Arts in Hanover in 1995 with a degree in Computer Science and Electronics. After receiving his PhD in Applied Computational Intelligence from The Open University, he worked as a System Engineer for EDS. He returned to The Open University as a Research Fellow in 2000. He joined The Nottingham Trent University as a Senior Lecturer in Computing in February 2002. His research interests include: applied computational intelligence, distributed systems, expert systems, optimisation and control of technical processes.



BRIAN O'NEILL graduated in 1968 from Trinity College, University of Dublin and obtained his PhD from Liverpool University in 1972. Before taking up the lecturing post at Nottingham Trent in 1977, he held research posts at Glasgow and Southampton Universities. His research activities originated from work on electronic instrumentation for powder flow control. This work required the development of fast real-time control and processing, using parallel processing, for its implementation. From this start he has widened his area of research to the design of hardware packet routing switches for inter-processor communication.