A Blackboard System for Interpreting Agent Messages

Marc Cavazza (1), Steven J. Mead (1), Alexander I. Strachan (1) and Alex Whittaker[i] (2)
(1) University of Teesside, TS1 3BA, Middlesbrough, UK.
{m.o.cavazza, steven.j.mead, a.i.strachan}@tees.ac.uk
(2) Sony Computer Entertainment Europe, 37, Kentish Town Road, NW1 8NX, UK.
Alex_Whittaker@scee.net

Abstract

We describe a prototype blackboard system for coordinating multiple agents in a real-time strategy game. Agents have limited cognitive abilities and communicate with the blackboard through structured messages. Agents sense their local environment at regular intervals and send perceptual information to the blackboard working space in the form of structured messages. The blackboard system performs high-level interpretation of these low-level percepts to generate high-level tactical and strategic information. In this paper, we describe the design of the various components of the blackboard. The system has been fully implemented and experiments have been performed with data from the Dropship™ game.

Introduction

Simulating large numbers of agents using limited resources is a recurring challenge in computer games. Real-time strategy (RTS) games typically involve the simulation of large numbers of Non-Player Characters (NPCs) whose behaviours must be coordinated, whether to cooperate with, or oppose, the player.

In real-time strategy games, where the focus is typically on the behaviours of groups of agents rather than each agent's individual characteristics, one approach to the agent-coordination problem is to centralise system control. This reduces overall complexity by allowing the internal structure of agents to remain simple, while the complexity resides within the centralised control system. Although this approach is not appropriate for games in which importance is placed on the uniqueness of individual agents (e.g. character-based adventure games), it is well-suited to the real-time strategy genre.

Another important aspect of agent- coordination is agent communication. Research in the field of multi-agent systems has developed the concept of an *agent communication language* (ACL), in which agents exchange information using structured messages. One area in which this method has been employed is in the field of military simulation (for instance the Control and Command Simulation Interface Language (CCSIL) (Salisbury et al., 1995)).

In this paper, we describe the use of a blackboard system to address the problem of agent-coordination using centralised control and communication through structured messages. The blackboard is used as a generic approach to processing agent communications centrally. Agents generate messages containing information about their internal states and surroundings, which are sent to the blackboard working space. These low-level messages provide the initial data for the system, and are interpreted by the system in terms of tactical concepts. This approach enables us to keep individual agent design simple and efficient without having to embed complex message-parsing procedures in the agents themselves.

Figure 1: The Dropship™ Game (Copyright © SCEE 2000)

The framework for our experiments is a computer game currently under development at Sony Computer Entertainment Europe (Camden studio), Dropship™. Dropship (Figure 1) is a battlefield simulation game in which the human player is confronted to enemy troops on the ground, at sea and in the air, according to various missions assigned to him to complete a game level. Unlike most RTS games, the human player in Dropship™ does not directly control an entire army, but controls a single unit which forms part of a larger computer-controlled force. Rather than playing the part of a commander-in-chief, the player is one pilot on the battlefield. This introduces the requirement that the game implement allied NPCs as well as enemy NPCs. Agent co-ordination can be implemented through various techniques, however, one major constraint is that the Dropship game should comprise a large number of agents: up to 400 units simultaneously populating a given level. In addition, agents have limited decision-making

processes and are essentially reactive to their local environment which they perceive through a small set of primitives (unlike, e.g. SOAR agents (Laird et al., 1987)).

Due to the limited intelligence of individual agents, there is a need to perform situation assessment and make tactical decisions at a global level. One possibility is to follow a traditional distributed AI approach in which tactical decisions emerge from the exchange of information between agents. There are, however, limitations and potential problems with that approach. Firstly, the flow of messages may become intractable: it is not always possible to limit exchanges to neighbouring agents (e.g. in the case of ground units requesting air cover, medical evacuation, etc.). Secondly, this emergent behaviour is not always accessible to the user. This is not only a problem when implementing and debugging the system, but also becomes a gameplay problem. It is a requirement that the behaviour of agents makes sense to the user, not only for enemy agents, but especially for allied troops whose immediate goals should be visible to the user with whom they are supposed to co-operate. Thirdly, there is an interest in making the messages themselves meaningful to the user if necessary. This would enable, for instance, the creation of a communications background (using sound generation or text messages on a dedicated "console" portion of the screen that could contribute to the game realism by re-creating an atmosphere). The fact that messages are directly manipulating an ontology which is related to the scenario also facilitates the knowledge-acquisition process by which the content of the blackboard's knowledge sources is acquired. One major difference of this compared to the CCSIL approach is, of course, that these messages are sent to and processed by a central structure, the blackboard. This provides a unifying principle which lends itself to future modifications and experiments, such as the use of distributed games over several platforms .

In the following sections, we describe the message-format and how messages are generated in response to game events. We then present the blackboard architecture and describe the constraints for attaining real-time performance. After discussing the issue of blackboard control and describing the knowledge sources, we give a detailed example of actual behaviour of the implemented system.

Message Structure

The structure of an agent communication language (ACL) is largely constrained by the agents' "cognitive" abilities. In our experiments agents have elementary perceptive abilities from which they can generate messages. These perceptive abilities are illustrated in the figures in the *Example* section, which show the development interface for the system.

Agents can also generate messages about their position (e.g. *Alive_At_Pos*), internal states (*Health_Low*, *Ammo_Low*), the actions they undertake (*SAM_Deployed_At_Pos*), actions they want other agents to undertake (*Request_Support*, *Request_Repair*) and their current goal(s).

Because our agents have limited cognitive abilities, there is no match, in the general case, between the message structure and the high-level event structure. This reflects that a single agent (e.g. a single tank) within a large-scale simulation has only a limited viewpoint of the global set of events. For instance, agents would perceive elements of their environment (*Forest_In_View*, *River_In_View*). This allows the derivation of local information[1] but does not provide global knowledge which could be used to formulate a course of action. This limited viewpoint also applies to tactical concepts: for instance, in the case of an air-strike an agent would only perceive its effects in terms of damage to units within the agent's local environment (i.e. not in terms of overall strategy). These agents are thus limited to sending various types of messages: they broadcast information on their immediate physical environment, inform about their status or the status of neighbouring units, and can send requests for assistance (covering fire, repair, refuelling, engineering work, etc.). Several message-formats have already been defined as part of agent communication languages such as KQML (Finin, 1995), FIPA's ACL (FIPA, 2000), and CCSIL (Salisbury, 1995). As a general rule, ACLs fall into two categories: those which aim primarily for flexibility and expressivity, and those which sacrifice these in favour of computational efficiency. The former category is often based on linguistic theory and includes KQML and FIPA's ACL. The latter category includes CCSIL, which is designed to reflect the structure of specific events, so as to optimise processing. Our own approach clearly falls into the second category, and has, unsurprisingly, been inspired in part by CCSIL.

Our primary aim in designing the message-format was efficiency, which is important if the system is to achieve real-time performance. The message-parsing procedure, performed each time-step, constitutes a large part of the computational cost of the overall system, so it was important to consider a message-structure which led to optimised performance.

[1] Not unlike CCSIL's *Obstacle_Geometry*_type or *Trafficability*.

Message Field	Value
Sequence No.	432
Code	MESSAGE_ENEMY_IN_VISUAL_RANGE
Header	(Time: 234325)(Lifespan: 10000)(Type: PERCEPT) (Delete: 0)(Viewed: 0)
Agent	(ID: 3)(Align: ALLIED) (U-type: ENTITY_DROPSHIP_ASSAULT) (W-type: WEAPON_MISSILE_A2A) (Quadrant: (3,4)) (Location: (254,634))
Object	(ID: 34)(Align: ENEMY)(U-type: ENTITY_SAM_MOBILE) (W-type: WEAPON_MISSILE_G2A) (Quadrant: (3,4)) (Location: (200,640))

Figure 2: A Sample Message

A principal factor which affects the efficiency of the message-parsing procedure is the level of complexity of the message-structure; a complex message-structure requires a complex parsing algorithm. The most common source of structural complexity in parsing is the use of recursively embedded structures, so for this reason we designed a flat message structure which does not permit recursive embedding. The reduction in expressivity resulting from this decision is more than compensated for by the gains in efficiency.

An example of a message generated by the system is shown in Figure 2. The example message translates as "Allied assault-dropship has enemy mobile-SAM unit within visual range" and includes position and weapon information for both entities. The remainder of this section describes our final message-structure.

A message's sequence number is a unique integer-identifier assigned chronologically by the system when the message is created, which is equivalent to a time stamp.

The header contains mostly content-independent information such as the time at which the message was posted to the working space, the message-type (e.g. *Percept*, *Threat*, *Situation_Assessment*), and two flags indicating, respectively, the number of times the system has parsed the message, and whether the message is to be deleted at the next time -step.

The message code (e.g. *Enemy_in_visual_range*, *Alive_at_position*) is equivalent in semantic terms to an action predicate, however the predicative structure for that action might not be reflected within a single message due to the partial knowledge an agent has of its environment. The action structure will be reconstructed on the blackboard from the incoming flow of messages.

The agent, objects, and parameters are arguments of the message code, providing further information regarding entities involved in the action (e.g. unit-types and locations). A message has one agent, but can have any number of objects or parameters. Parameters are intended to provide numerical information such as number of enemy units in a given area, or strength of enemy air-defences.

The Blackboard System

A blackboard (Engelmore and Morgan, 1988; Nii, 1986) is a specific architecture for knowledge-based systems which integrates various knowledge structures, called *knowledge sources* (KSs), around a central data structure known as the *working space*. Blackboards have been designed specifically to cope with complex, ill-structured problem domains, and to allow exploratory programming of knowledge-based systems by integrating heterogeneous knowledge sources. The main difficulties encountered with blackboards are their control strategy and securing real-time performance. We will see how these difficulties can be partially overcome in the specific context of computer games. The architecture of our blackboard system is shown in Figure 3.

The working space is the centralised data-structure used to store the messages generated by the agents and the knowledge sources. Agents send messages to the working space via the *controller buffer*, while messages generated by knowledge sources are sent to it directly. Agents send messages to the current time-slice of the working space (see below).

The working space is a hash table implemented as a sparse array, and uses direct-indexing and collision lists. This significantly reduces the storage requirements. The storage location of a message within the current working space time slice is determined by the XZ-coordinates of the sending agent, or the main agent of interest (in the case of messages posted by knowledge sources). Although in principle a situation could arise in which all messages originated from the same zone (and therefore all the messages would be stored in the same cell), this is highly unlikely. This would cause the working space to degrade to a single linked-list, but could be avoided by considering the problem when deciding on the size of game-zones.

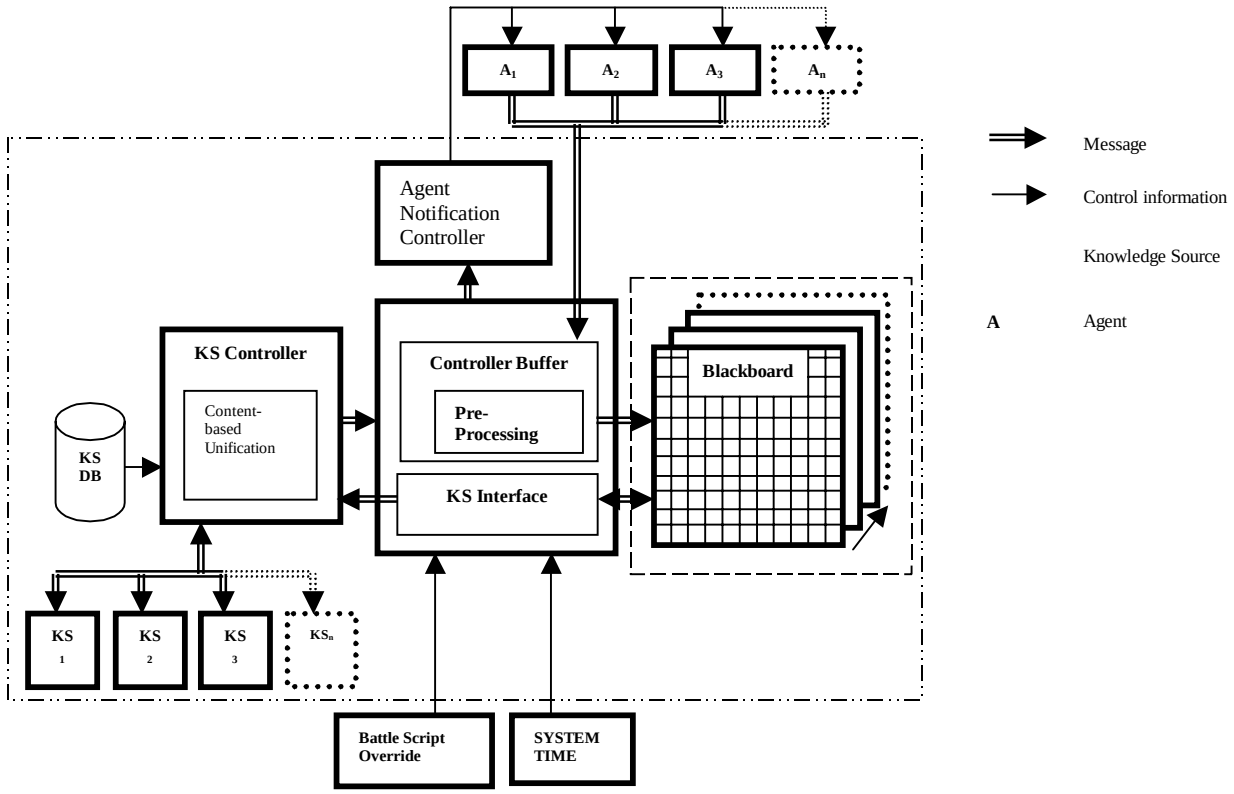

Figure 3: The Blackboard Architecture

At each time-step, a new time-slice is generated for the working space, and the oldest is discarded. As a result of structuring the working space according to time and the positions of units, the blackboard is well-suited to spatial and temporal reasoning.

Knowledge Sources & Blackboard Control

Each KS contains a set of production rules and an inference engine which operates on those rules. The choice and partitioning of KSs is determined by gameplay concepts, but generally each KS is responsible for a different aspect of the overall problem. In our case, KSs are assigned for different aspects of situation-assessment (e.g. Threat, Battle-damage Assessment, Mission-objectives, Search-and-rescue, Unit-movement-planning, Debriefing).

Rules are parsed at system start-up to produce an internal representation. A rule has one or more preconditions and a single postcondition. Preconditions are conjoined and disjunction is obtained by producing multiple rules with different preconditions but identical postconditions. The format for conditions is based on the message-format and includes additional information, such as: a negation flag and a *quantifier*. This is not a quantifier in the strict sense, but indicates the number of matches required for the precondition to succeed (possible usage: "If an allied unit has three enemy units in sight, request backup").

The Blackboard Cycle

Figure 4 shows the blackboard cycle. At each time-step, agents sample their environment and generate low-level messages, known as *percepts*. These percepts are sent to a temporary buffer (the *controller buffer*) for pre-processing, before being posted to the blackboard working space. Due to the reactive nature of message generation, identical messages about the same event can in principle be generated by different agents, which would differ only by their sender ID and time-stamp. The purpose of pre-processing is therefore to remove near-duplicate messages, so as to reduce redundant processing at the pattern-matching stage, and also to check message validity and obsolescence. Specific procedures exist for referent-resolution, i.e. checking that various messages refer, for instance, to the same object. Other than these operations, most message-unification is content-specific.

After pre-processing has taken place, a new time-slice is added to the working space. The pre-processed messages are copied to the appropriate locations in the new time-slice and deleted from the temporary buffer.

The next stage is pattern-matching, in which each KS in turn scans the working space for matches between

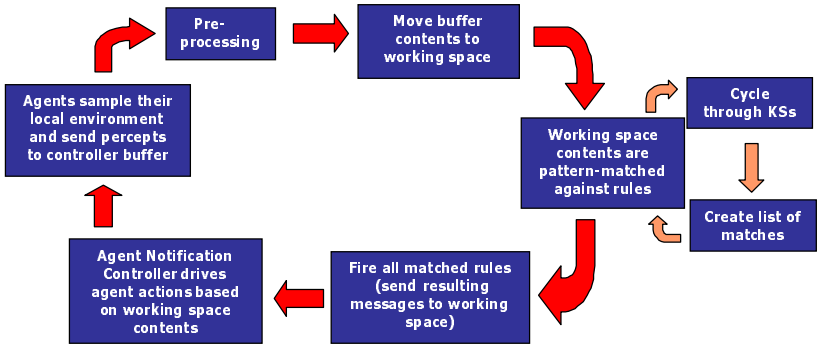

Figure 4: The Blackboard Cycle

messages and rules. Pattern-matching can be computationally expensive, especially when a large number of messages are to be processed by knowledge sources. Experiments to implement pattern-matching algorithms such as RETE (Forgy, 1982) in the context of computer games have not been fully conclusive (Wright, 2000). This is why we have taken a minimalist approach to pattern-matching, specifying the message formats on a type basis, which avoids having to recursively parse message structures.

In a traditional blackboard, knowledge sources exchange information through the blackboard's working space, progressively refining a solution to the problem being solved. The data on the blackboard are heterogeneous: initial problem data, partial solutions, hypotheses to be examined by other knowledge sources, or even control knowledge. The fact that data show various level of abstraction is a factor of complexity. In order to decrease the complexity of inferences, we reduced the number of entities present on the blackboard to only initial agent messages and instantiated templates, which correspond to a unit of situation assessment.

In other words, we have essentially implemented a monotonic, pipelined approach. We are not taking advantage of the ability of blackboards to explore hypotheses and alternatives (either by backtracking or maintaining concurrent hypotheses). Rather, we are essentially relying on i) their integration of various knowledge sources and ii) a unified paradigm for message processing provided by the blackboard working space.

Blackboard control is a difficult aspect of blackboard implementation. The blackboard cycle implemented in our system is represented in figure 5 and encompasses the whole cycle of message production and interpretation. The overal*l* cycle depends *simultaneously* on agent sampling their environment and on blackboard control. The overall cycle is determined by i) message production by the agents and the rate at which they sample their environment ii) how the flow of messages is buffered and processed iii) blackboard control and the selection of activ*able knowledge sources* to be fired and iv) how actions decided on the blackboard are passed back to the agents. Agents constantly sample their environments on a time-scale corresponding to user actions (which are by nature non-deterministic) and agent ongoing behaviour (current motion, attacks, etc.). They encode relevant information in messages that they post on the blackboard. Some assumptions related to the closed world hypothesis have to be made in order to limit the number of messages posted. For instance, the message alive_at_position combines information about the agent's health status and its position. The rationale for this combined message is that it corresponds to the availability of a given unit for tactical actions. While it is possible to determine the sampling rate of position updating (on the basis of agents' speed, for instance), a new message has to be generated each time the health-value is modified, as this depends on circumstances external to the agent. This can help in making hypotheses on whether an agent is alive or dead, based on a history of messages received.

The set of messages on the blackboard is matched to the various KSs through the simplified pattern-matching procedure that is tuned to message structure. The activable knowledge sources constitute Knowledge Source Instantiations (KSIs) that are waiting for the controller to fire them. The control, in our case,
is under the dependence of simple domain heuristics. In other words, we could describe our approach as centralised decision-making for distributed agents, though it retains some aspects of traditional blackboard control, something which distinguishes our implementation from a simple variant of distributed problem solving. Several authors have discussed the relations between blackboard architectures and distributed architectures for problem solving: see for instance (Craig, 1989) and the comments on his CASSANDRA system in (Carver and Lesser, 1994). In the next step, KSIs are fired, which results in updating templates on the blackboards working space. At this stage, messages which have been processed can be discarded unless they contain information that might be

used again (unlike hypotheses, which can be deleted once examined). The new information posted by KSs on the working space can further trigger the KSs, though this has to be synchronised with the incoming flow of messages. This is an illustration of the need for control: decisions have to be taken, in some cases, to give priority to the thorough processing of a set of data from one time-step (to avoid saturation forward chaining of the KS) before taking new messages on board. Specific KSs are in charge of returning information orders to the agents. These can be either fired opportunistically or wait for the completion of situation assessment (as these actions are irreversible).

Example: Establishing Air-Defence Priorities

The system is able to perform non-trivial inferences from agent messages, as the following example illustrates.

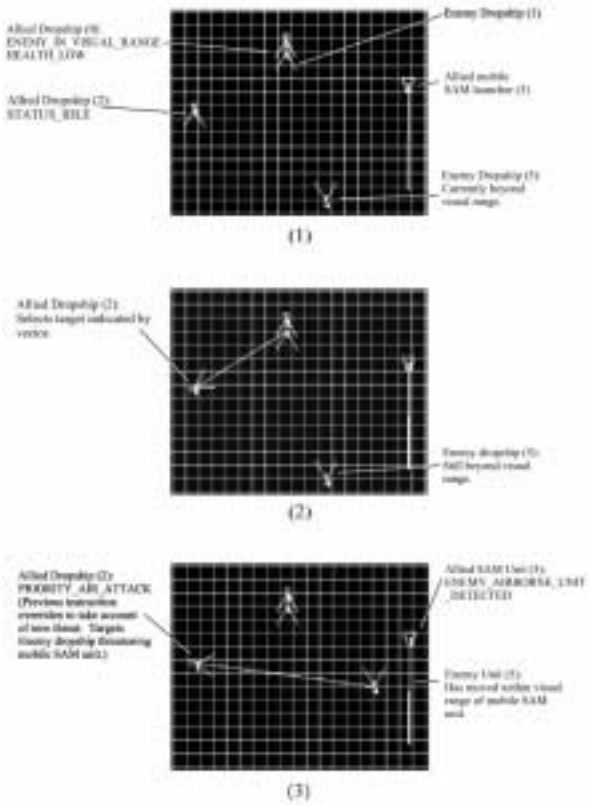

Figure 5: Establishing Priorities in Air Defence

At time-step 1 (Figure 5.1) allied dropship 0 has low health and an enemy in visual range, while allied dropship 2 is idle. An enemy dropship (5) is approaching the allied SAM launcher (3) but is currently beyond its visual range. The situation at time-step 1 is represented in message form by each unit posting a message with code *Alive_At_Pos*, while in addition 0 posts *Health_Low* and 2 posts *I_Am_Idle*. These are the low-level percept messages from which inferences will be made. At time-step 1 the threat KS finds a full match for the *UnderThreat* rule which is matched if a unit has posted Health_low and Enemy_in_visual_range.

This rule is fired by the KS controller, causing the message *Under_Threat* to be posted. The agent field of the *Under_Threat* message contains the threatened agent's details, while the object field contains information about the enemy agent, such as weapon, unit-type, and location. At time-step 2 (Figure 5.2), the messages *Under_Threat* (posted by unit 0) and *Status_Idle* (from unit 2) provide a match for the rule *AttackEnemyThreateningAlly* in the search_and_rescue KS. The message is posted and at the end of the blackboard cycle is detected by the agent notification controller, which orders unit 2 to select 1 as its current target and move to attack it. This is shown as a blue line connecting the two units. At time-step 3, an enemy dropship (5) moves into visual range of an allied SAM launcher (3), prompting unit 3 to post the percept message *Sam_Detect_Airborne_Enemy*. This is a message specifically intended to detect air-attacks on SAM units. This situation leaves unit 3 unsupported, putting allied air-defences at risk. What must be decided is which allied unit has the higher priority for support. An airborne unit such as 0 is equally manoeuvrable as its attacker and is equipped with counter-measures (e.g. chaff, flares, ECM) and various weapons with which it can defend itself. A ground-based unit such as 3, however, is more vulnerable to air-attack because of lack of manoeuvrability. A mobile SAM launcher is also of particular strategic importance as part of allied air-defences, so it is in allied interests to prevent the destruction of (3). At time-step 4 (Figure 5.3), the two messages *Attack_Enemy* and *SAM_Detect_Airborne_Enemy* provide a match for the rule *PriorityAirAttack*, causing the message *Priority_Air_Attack* to be posted. This causes unit 2 to change its current target to the enemy dropship (5), which is attacking the mobile SAM launcher (3). This change can be seen through the blue targeting line, which has now moved to point to 5. The above scenario is an example of how the blackboard system is used to synthesise high-level knowledge from percept data. Three non-trivial inferences (*Under_Threat*, *Attack_Enemy*, and *Priority_Air_Attack*) have been generated from percepts, and were used to drive the actions of an agent.

Conclusion

The use of a blackboard for multi-agent co-ordination has many advantages in terms of modularity, flexibility and expressivity. The system has been fully implemented and tested using actual data from Dropship™ levels. At this stage, we mainly use our blackboard to support

exploratory programming. However, blackboard systems have a real potential for the co-ordination of multiple agents: a central controller helps keeping the agent internal complexity tractable, while being compatible with various paradigms for agent communication. The fact that blackboards maintain explicit data representations also offers perspectives for enhancing user feedback, for instance by generating natural language messages to the user.

Acknowledgements

The authors wish to thank Rob Parkin of SCEE™ for giving us access to the Dropship data.

References

Carver, N. and Lesser, V., 1994. "The Evolution of Blackboard Control Architectures". *Expert Systems with Applications*, Special Issue on The Blackboard Paradigm and Its Applications, vol. 7, no. 1, pp. 1-30.

Craig, I. (1989). *The CASSANDRA Architecture: Distributed Control in a Blackboard System*. Ellis Horwood.

Englemore, R. and Morgan, T., 1988. *Blackboard Systems*. New York: Addison-Wesley.

Finin, T., Labrou, Y., and Mayfield , J. (1995). *KQML as an Agent Communication Language* in Jeff Bradshaw (Ed.), Software Agents, MIT Press, Cambridge, 1995.

FIPA. (2000). Foundation for Intelligent Physical Agents Homepage: http://www.fipa.org

Forgy, C. (1982). Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. Artificial Intelligence 19 (1): 17-37.

Laird, J., Newell, A., and Rosenbloom P. S. (1987). Soar: An architecture for general intelligence, *Artificial Intelligence* 33:1-64.

Lakin, W.L., Miles, J.A.H. and Byrne, C.D., 1988. "Intelligent Data Fusion for Naval Command and Control". In: R. Englemore and T. Morgan (Eds.), *Blackboard Systems*, New York: Addison-Wesley.

Nii, H.P., 1986. "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures". *AI Magazine*, 7 (2).

Salisbury, M. R., D. W. Seidel, L. B. Booker, December (1995). A Brief Review of the Command Forces (CFOR) Program. *Proceedings of the Winter Simulation Conference*, Arlington, Virginia

Salisbury, M. R., March (1995). "Command and Control Simulation Interface Language (CCSIL): Status Update". *Proceedings of the Twelfth Workshop on Standards for the Interoperability of Defense Simulations*, 639-649, Orlando, Florida.

Terry, A., 1988. Using Explicit Strategic Knowledge to Control Expert Systems. In: R. Englemore and T. Morgan (Eds.), *Blackboard Systems*, New York: Addison-Wesley.

Wright I. and Marshall, J. (2000). RC++: A Rule-Based Language For Game AI. Proceedings *GameOn 2000: 1st International Conference on Intelligent Games & Simulation*, London, UK.

---

[i] Alex Whittaker's Current Address:
Elixir Studios Ltd., 93 Bayham St., Camden, London, NW1 0AG.