Implementing a Blackboard System in a Distributed Processing Network

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

Abstract. A blackboard system is a software architecture that mimics a group of human experts gathered around a physical blackboard to solve a problem. In the software model, the blackboard is an area of shared memory and the experts are software modules called knowledge sources. DARBS is a distributed blackboard system in which the knowledge sources are implemented as parallel or concurrent processes. The Nottingham Trent University has developed a distributed processing network called SARNet that is suitable for the implementation of DARBS. This paper discusses the implementation of DARBS on the SARNet with a view towards producing an embedded blackboard system for building intelligent behaviour into machines.

Keywords: agent, blackboard system, distributed processing network, DARBS, SARNet, embedded system

1. Introduction

Throughout the evolution of artificial intelligence (AI), many different techniques have been developed to solve a wide variety of problems. Each AI method is generally only suited to a certain class of problems. For example, neural networks are particularly well-suited to solving pattern classification problems, especially if the patterns are not known to the designer [1]. Rule-based systems, on the other hand, are ideal for problems where the domain knowledge is fully known and can be coded into rules [2]. Unfortunately, real world problems are often not so straightforward and often require a mix of different AI methods. The blackboard system [3] provides a way to integrate a variety of different AI techniques within one system. It is analogous to a team of experts who communicate their ideas by writing them on a blackboard [4]. The team of experts here are represented by the different AI methods working together to solve the overall problem on the blackboard.

Although the speed of available microprocessors has progressively increased, eventually further increases will not be possible owing to the finite speed of light [5]. Because of this, different ways of increasing the performance are being researched. One approach is to use parallel processing, where a group of microprocessors is connected together to compute data simultaneously. There are two main ways for the microprocessors to communicate with each other: either through a common shared memory ("tightly coupled") or by passing messages through a communication network ("loosely coupled") [6]. The shared memory model is efficient when a small number of processors are connected together, but a bottleneck occurs with a large number of processors due to memory access contention [5]. This problem is avoided in the message-passing model as each processor has its own local memory, although there is a communication overhead when passing messages between processors [5]. In general, the message-passing model scales better than the shared memory model [7].

The miniaturisation of electronic components has made it possible for an entire system to be fitted onto a single integrated chip. This has led to the development of embedded systems, i.e. small self-contained purpose-specific systems that comprise both hardware and software [8]. They are included in, for example, mobile phones and photocopiers. Embedded distributed processing networks are the integration of distributed processing networks in an embedded system. For example, a robotic arm in an assembly plant may have many embedded processors to control its movement and interpret its

sensors. These embedded processors communicate with each other in order to perform their assembly task.

In the software model, the blackboard is an area of shared memory and the experts are software modules that act as agents, often called knowledge sources (KSs). This model maps well to an embedded distributed processing network. Each processor node can represent a knowledge source and the communication network is the means of communication to another processor node representing the blackboard. In the future, this may lead to the development of distributed intelligent embedded systems. For example, an intelligent tank may contain an agent for each role undertaken by the crew: commander, driver, gunner, and loader. Each agent can be represented as a knowledge source running on an individual processor node in the distributed processing network. The mission objective can be written onto the blackboard and each agent can perform its specific task to accomplish the overall mission objective.

2. SARNet

The SARNet is an embedded distributed processing network that is being developed at the Nottingham Trent University. It is a message passing network consisting of a network of SARNodes (processor nodes) connected together via an ICR C416 router [9]. The SARNodes were designed with an embedded system in mind. Each SARNode consists of a StrongARM SA-110 RISC processor, 8 megabytes SDRAM, a 32-bit timer, I/O and UART debug port, and an OS-Link (over sampling link) [9] communication module. The ICR C416 router uses the OS-Link communication protocol that uses the wormhole routing strategy, which has a low communication overhead [10][11]. The SARNet can have many different switch network configurations, one of which is shown in Figure 1. Each ICR C416 router can be connected to other routers using any one or more of its OS-Link channels to make up a bigger network. A PC can also be connected to the SARNet via an interface card [12] as shown in Figure 1.

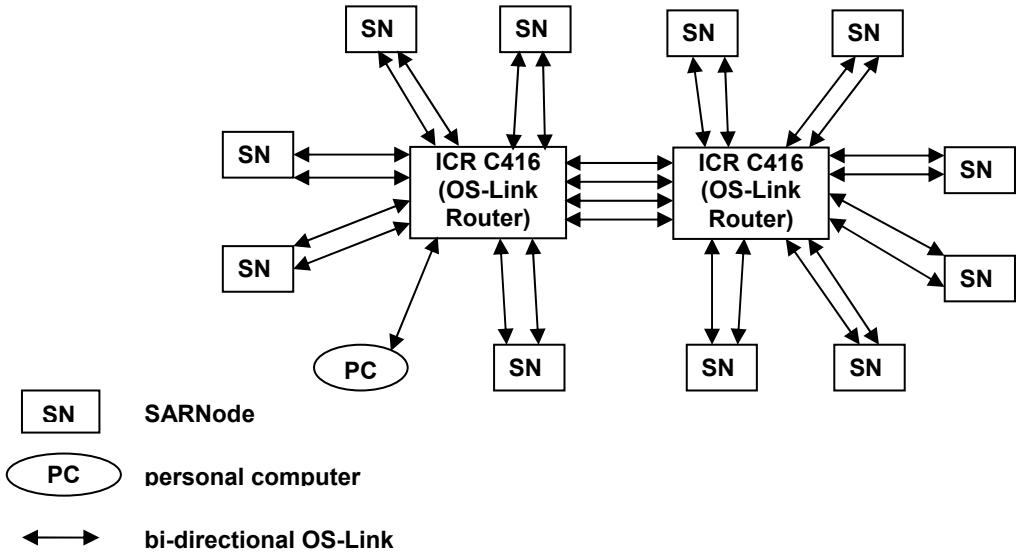

Figure 1. *Typical network topology of SARNet*

The operating system for the SARNode is SARNUX, designed for use in an embedded system [13]. In SARNUX, the communication is based on the communicating sequential process (CSP) model [14]. In this model, messages between processes are passed through channels. The processes can be internal (within the same processor) or external (on different processors). Each channel is a dedicated single direction communication link between two processes. If a bi-directional link is required between two processes, two separate channels would need to be declared. Communication between two processes

only takes place when both the sender and the receiver are ready. Otherwise the first process (either the sender or the receiver) would be blocked while waiting for the other process. This is also used as a synchronisation method.

3. DARBS

DARBS (Distributed Algorithmic and Rule-based Blackboard System) [15] is the distributed version of ARBS [16]. It is ideally suited for implementation on an embedded distributed processing network. As DARBS is a distributed blackboard system, its knowledge sources and the blackboard run as separate processes. The structure of DARBS can be seen in Figure 2.

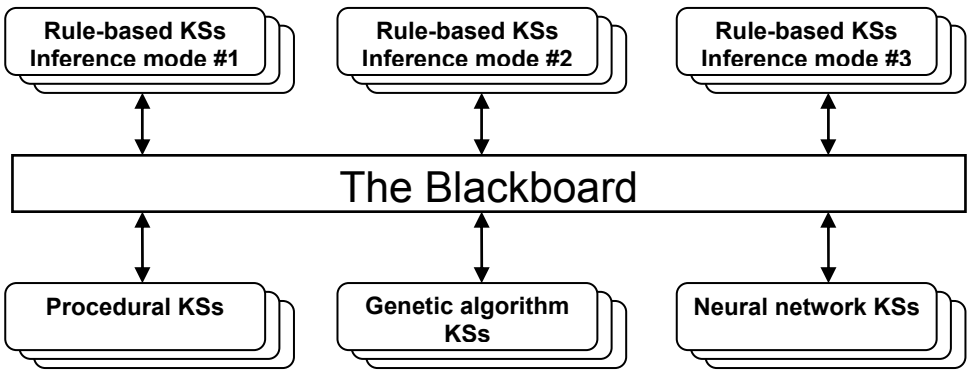

Figure 2. *DARBS structure*

The design of DARBS is based on the client/server model where the blackboard is the server and the knowledge sources are the clients. DARBS runs on a PC-based Linux operating system, and uses TCP/IP as the communication protocol between processes. The processes can reside on a single computer or on different PCs networked together via TCP/IP. Data on the blackboard is organised into partitions. DARBS is currently being applied to the original ARBS problem of interpreting ultrasonic images of welds [17].

4. Implementing DARBS on the SARNet

There were a variety of challenges in implementing DARBS on the SARNet. Here we present the main one – the need to develop a new inter-process communication (IPC) model.

The DARBS communication module uses the Linux operating system's IPC model that, in turn, uses the TCP/IP communication protocol. The Linux operating system uses signals as a form of interrupts. Each process in the Linux operating system can register its interrupt service routine or function to be called when a particular interrupt/signal is generated [18]. For example, the signal that is generated when a message is received on Linux's IPC model is SIGIO [18]. So, by registering a function that is triggered by a message from the communication channel to the SIGIO signal, this function would be called whenever a message is received on Linux's IPC model. Linux's IPC also provides a listen-to-port function for the server. In the client/server model, the server initiates a service on a port and listens on that port for possible clients. Once the server has started to listen on a port, the client can make a call to connect to that port and the call would automatically pass to the server. The server can then register the client's file descriptor and start to provide services to it. Linux's IPC also provides a function call to send data to a process (client or server) by using the process's file descriptor. DARBS's communication classes make use of these Linux functions to implement the client/server model. The server's communication class has an extra broadcast function that sends a message to all the clients except for the current client, i.e. the client that last sent a message to the server.

As mentioned earlier, the SARNUX communication model is based on the communicating sequential process (CSP) model [14]. SARNUX has, however, employed buffers in the external receiver link side of the communication channels to prevent link blocking. This is because the SARNode has two physical OS-Link links, and they operate on the send-and-acknowledge protocol [19]. If a process is not ready to receive, but there is a message waiting for it to receive on the physical link, then that physical link would be blocked. Other processes that are ready to receive or transmit cannot use that link. This can easily lead to a deadlock [20]. To prevent this from happening, receiving buffers are used. However, if the receiving buffers were full the link would be blocked. The onus is on the application designer to ensure that this does not happen. Because of the different communication models used by SARNUX and Linux, a new DARBS Inter-Process Communication model had to be designed that would minimise the changes required to the core DARBS source code.

4.1. New inter-process communication model

The original DARBS program has three communication classes to handle the Linux IPC model. The new IPC model that has been designed comprises four new communication classes, with the original three DARBS communication classes changed to wrapper classes (Figure 3). The main reason the original communication classes were kept and changed to wrapper classes is because this minimised the changes required to the core DARBS source code. The OSLinkToTcpClient and OSLinkToTcpServer classes are the interface classes between the original LnTcpClient and LnTcpServer communication classes and the actual SARNUX IPC handler classes (OSLinkClientHandler and OSLinkServerHandler).

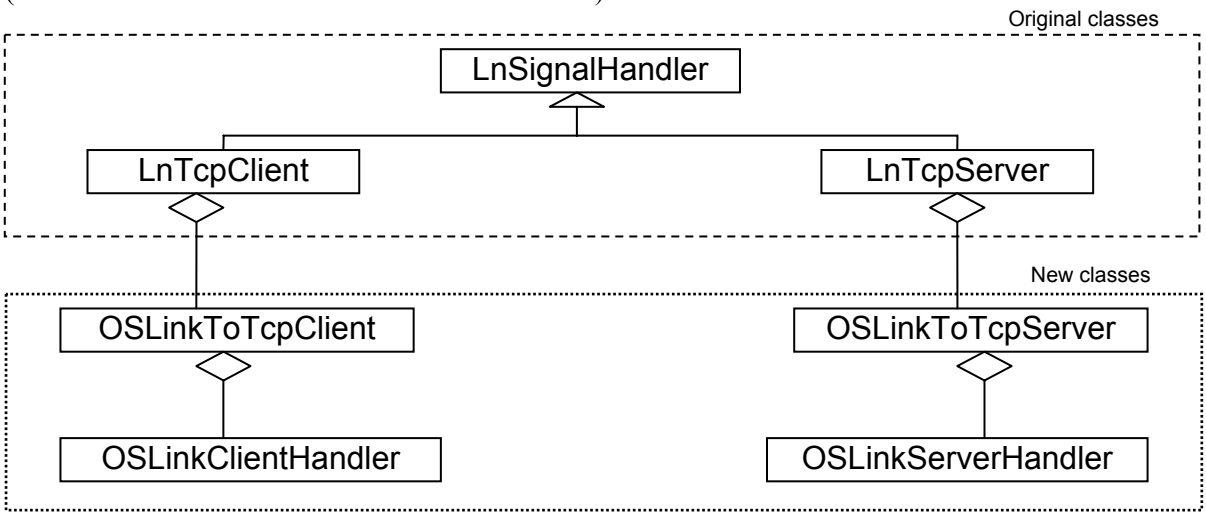

Figure 3. *New inter-process communication classes*

The IPC model has two sides: the client side and the server side. On the client side, the main functions required from the new IPC model are transmit, receive, connect to server, and set up the call-back function. Due to the block send-and-receive nature of the CSP model used in SARNUX, the only way for the main KS client program to continue running and still have the communication running in the background is to have separate threads running concurrently for receiving and transmitting. With multiple threads running, some form of synchronization and communication is required between the threads. The KS client side of the new IPC solves this problem by the design shown in Figure 4.

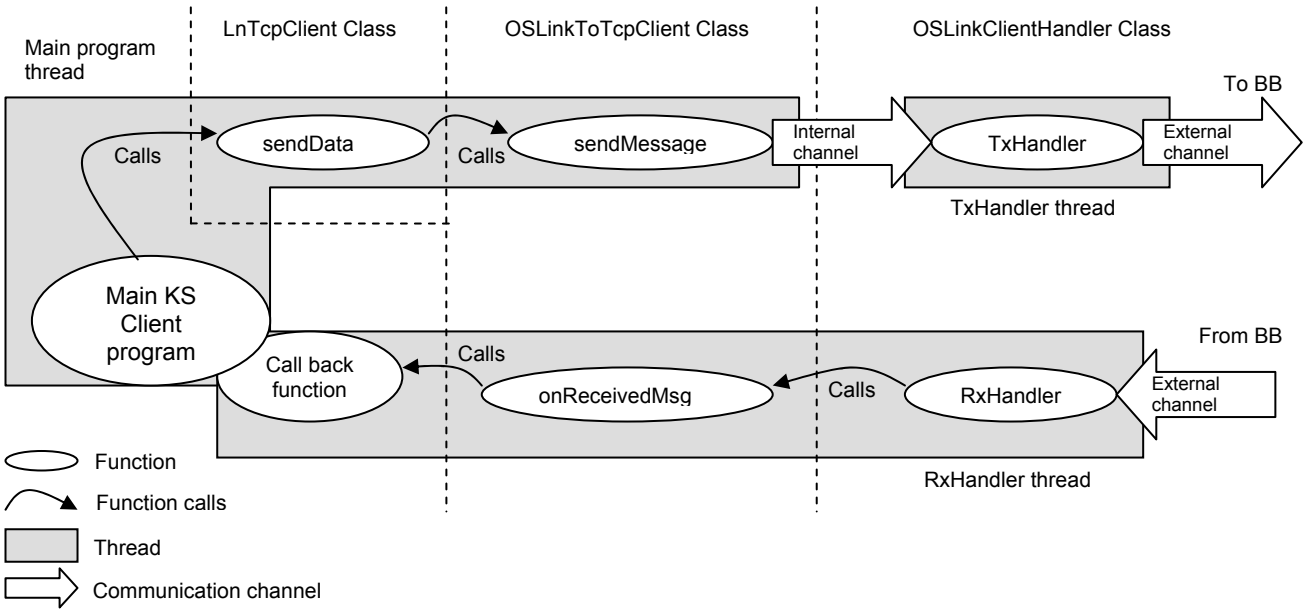

Figure 4. *IPC model for the client side*

For the server side, the IPC model is more complicated than the client's IPC model as the server has to deal with multiple transmissions coming from different clients at the same time. The main functions required for the server side's IPC model are open listening port, receive from multiple sources, transmit to single source, broadcast to multiple source, and set up the call-back function. To achieve this, multiple TxHandler threads and RxHandler threads are created to handle each of the KS clients in the system. Because this is a static system, the number of KS clients in the system is known beforehand and therefore it is possible to create a TxHandler and RxHandler thread for each KS client in the system (Figure 5 & Figure 6). To achieve synchronisation, a semaphore is used on the receiving part of the server side's IPC model. This semaphore also helps to maintain data integrity on the blackboard.

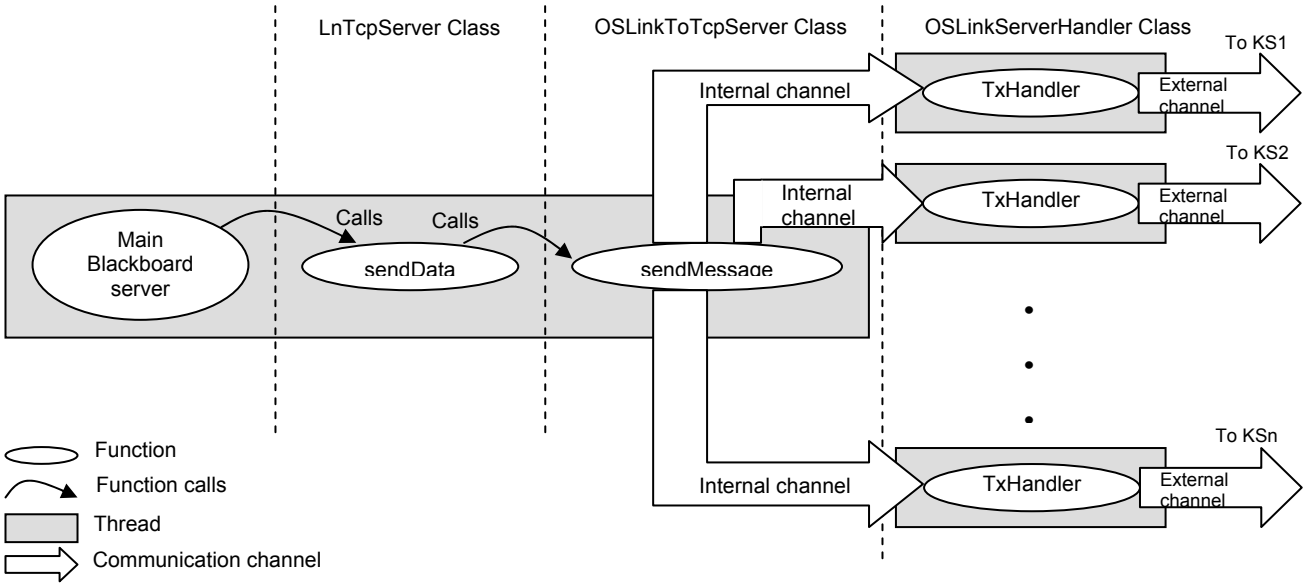

Figure 5. *IPC model for server side's transmitting part*

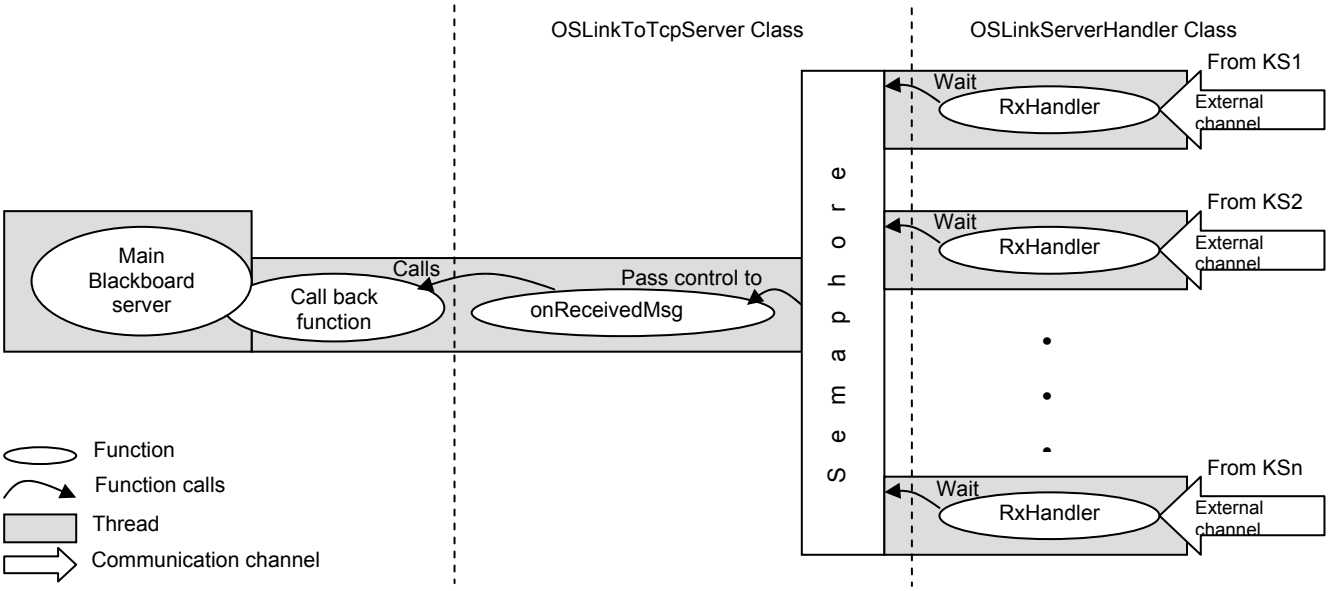

Figure 6. *IPC model for server side's receiving part*

5. Testing

After implementing DARBS on the SARNet, tests were carried out to prove that DARBS was working as designed. The test plan was carried out in three phases. The first phase was to test that the new DARBS IPC could send and receive messages. The second phase was to test the blackboard to make sure that it could handle multiple KS clients transmitting messages to it simultaneously. The third phase was to test whether DARBS on the SARNet could run rule-based KSs and perform the same functions as the original DARBS running on Linux.

For the first and second phases, the original DARBS terminal program was used. The KS clients in the DARBS terminal program are simple terminal clients. The KS clients take commands from the user and transmit them to the full running blackboard server used in DARBS. In the first phase, one KS client and one blackboard server were run on two SARNodes respectively. The result from this test was positive as the new IPC functioned as it was designed to. A partition could be created and items added to the partition on the blackboard via the terminal client. Also, the blackboard could transmit back to the terminal client the contents of the partition. This proved that the new IPC can transmit and receive messages.

In the first part of the second phase, all four SARNodes were used, i.e. one blackboard server and three terminal clients. Unfortunately, the clients crashed when the blackboard server was trying to broadcast a message to them. Investigations revealed that a deadlock occurred when the clients tried to print out debugging messages on the UART debug port. This is because there are multiple threads running in a SARNode and each one of them tries to access the single UART debug port concurrently. This problem was rectified by running the print out of the debugging messages function with the interrupts masked. This guaranteed the print out of debugging messages function was mutually exclusive and therefore other threads would not be time-sliced in between access to the UART debug port. The test was rerun and this time the system worked. All the KS clients received the broadcasted message from the blackboard server correctly and could issue commands to it correctly. In the second part of the second phase, all three KS clients were programmed to automatically send requests to create a partition each and to add 50 different data items to their partition. The KS clients were started simultaneously and the blackboard server handled multiple requests from the clients. The result of this test was that the blackboard server handled all the requests properly, as designed. This proved that the semaphore worked and that the data integrity of the blackboard server was not compromised.

In the third phase, the TestCompare KS of the original DARBS was used. TestCompare KS is a simple rule-based KS that consists of two rules. The first rule adds data sets to a partition and the second rule looks through the data sets to find data that are more than a certain value and report them. DARBS on the SARNet started up correctly and requested the KS file (testcompare.dkf) to be downloaded to the KS client via the OS-Link. Testcompare.dkf was sent to the KS client and was received correctly by the KS client (checked by the debug message coming from the UART debug port). Next, the KS client requested the setdatacompare.drf rule file followed by testcompare.drf rule file. This was also sent to the KS client via the OS-Link and checked using the UART debug port. As soon as the KS client received the last testcompare.drf rule file, DARBS started to run. Rules were created and fired. The test data were added to the blackboard and later compared for values that were more than 10 and reported them. The function of TestCompare KS in DARBS on SARNet was compared with the TestCompare KS in DARBS on Linux and they both functioned identically. This proved that DARBS has been successfully implemented on the SARNet, and that it works as designed.

6. Discussion

The new IPC minimised the changes required to the core DARBS source code. This was proven in the testing phase of the development of DARBS on the SARNet. This IPC can also be made more general and thus can be a standard interface for emulating Linux's communication model over SARNUX's CSP-based communication model. With this general IPC, Linux-based applications can be easily transferred onto SARNUX. However, on the negative side, this type of emulation is not ideal for an embedded system as it introduces extra overheads and thus slows the overall system performance. For embedded systems, the application program should ideally be as efficient and small as possible. There is a trade-off between ease of portability and coding for efficiency. Future performance tests could be done to see if the emulation overhead is still within the acceptable tolerance of an embedded system.

There is also a memory constraint on the SARNode. The DARBS memory usage is quite large, at slightly more than 2 MB for the blackboard server. Because SARNodes were developed for an embedded system, the available memory space is only 8 megabytes per node. After splitting up the memory for the DMA engines of the OS-Link communication module, stack, page table, and STAB info, only about 6 MB of memory space is left which would then need to be shared with the application code and the heap memory space. As DARBS uses exception handling and STL [21], the heap memory consumption is relatively large. At the moment, DARBS is still within the 8 MB memory space but any larger applications would cause memory space problems. As DARBS was originally coded for a PC, without significant memory constraints, it contains some redundant code. Removing this redundant code would reduce the overall memory consumption and perhaps speed up the overall program. Meanwhile, research is underway on the next generation of the SARNode which would be implemented as a single system-on-a-chip [22] with increased memory space.

There were also weaknesses on both the SARNode and SARNUX in terms of its debugging facilities. The SARNode hardware only provides a UART debug port for debugging software, as used by SARNUX. This is suitable for debugging small programs on the SARNUX, but is not suitable for debugging large programs that run as multiple threads. The SARNUX operating system tries to compensate for this by providing a stack walkback feature for tracing back to the crash point of an application. This is only effective if the crash is not fatal and the operating system is able to recover. It would be better if the SARNode and SARNUX supported program step-through and runtime variable watches. These features proposed to be implemented in the next generation of the SARNode.

7. Conclusions and future work

This paper has introduced the distributed blackboard system DARBS and demonstrated its suitability for implementation on the SARNet embedded distributed parallel processing network. This research has opened up new opportunities for implementing a practical intelligent application in an

embedded distributed processing network. The next step now is to further test the system for its performance. A small test application for DARBS will be implemented that is small enough to run on a single SARNode. The test will involve measuring the average time taken to run the application on one SARNode, and then measuring the average time taken to run the same test application on four SARNodes. From this a speedup factor [23] can be calculated, i.e. $S_4 = t_{p1}/t_{p4}$ where S_4 is the speedup factor of four processors, t_{p1} is the execution time of the program on one processor, and t_{p4} is the execution time of the program on four processors. The same experiment will then be carried out on one and four Linux-based PCs connected by Ethernet. The speedup factor for the PC architecture can then be calculated. These two speedup factors can then be compared to see the effects of the system architecture.

Another future plan is to implement a distributed genetic algorithm on DARBS, to see if there is a behavioural difference between parallel and concurrent execution of the algorithm. Because the independently running KSs are equivalent to agents in a multi-agent system, it will be interesting to see the effects of parallel and concurrent execution on the behaviour of these agents.

References

[1] P. Picton, “Introduction to neural networks”, England: Macmillan, 1994. pp. 1-12.

[2] M. Negnevitsky, “Artificial Intelligence: A guide to intelligent systems”, UK: Addison-Wesley, 2002. pp. 25-52.

[3] A. Harrison, P. G. Thomas, "Integrating multiple and diverse abstract knowledge types in real-time embedded systems", Knowledge-Based Systems, vol 9, no 7, Nov 1996, pp. 417-434.

[4] A. A. Hopgood, “Intelligent Systems for Engineers and Scientists”, CRC Press, 2002, pp 224-226.

[5] L. Uhr, “Multi-computer architectures for artificial intelligence: Towards fast, robust, parallel systems”, Canada: John Wiley & Sons, 1987.

[6] J. M. Crichlow, “An Introduction to Distributed and Parallel Computing”, UK: Prentice Hall, 1988.

[7] K. L. Wong, “A Message Controller for Distributed Processing Systems”, Ph.D. thesis, The Nottingham Trent University, 2000.

[8] M. Barr, “Programming Embedded Systems in C and C++”, O'Reilly, 1999.

[9] “ICR C416 - 16-port Dynamic Routing Switch for Transputer Links - Data Sheet”, ICRouting Ltd., 1996. Obtainable at: <http://eee.ntu.ac.uk/research/parallel/icrltd.html>

[10] R. Hotchkiss, K. L. Wong, B. C. O’Neill, G. C. Coulson, S. Clark, P. D. Thomas, “The Building Blocks for a Parallel Network Incorporating the StrongARM Microprocessor”. The 1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98), Las Vegas, Nevada, USA, July 1998. pp 1863-1870.

[11] R. Hotchkiss, B. C. O’Neill, S. Clark, “Fault Tolerance for an Embedded Wormhole Switched Network”, Parelec’2000, IEEE Computer Society proceedings, Aug 2000. pp. 79-83.

[12] J. H. Ng, B. C. O’Neill, S. Clark, “A PC Interface Board for Parallel ARM Processor Network”. PREP 2000, IEE, April 2000. pp. 469-474.

[13] E. W. K. Liew, D. Kaye, B. C. O’Neill, S. Clark, “Operating System Support for StrongARM Multi-Processor Communications”. Proceedings of the ISCA 13th International Conference on Parallel and Distributed Computing Systems, Aug 2000. pp. 334-339.

[14] C.A.R. Hoare, “Communicating Sequential Processes”, Prentice Hall, 1985.

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

[16] A.A. Hopgood, H.J. Phillips, P.D. Picton, N.St.J. Braithwaite, “Fuzzy logic in a blackboard system for controlling plasma deposition processes”, Artificial Intelligence in Engineering, 12, (1998) pp. 253-260.

[17] A. A. Hopgood, N. Woodcock, N. J. Hallam, P. D. Picton, “Interpreting Ultrasonic Images Using Rules, Algorithms and Neural Networks”, European Journal of NDT, Volume 2, No 4, April 1993. pp. 135-149.

[18] N. Matthew, R. Stones, “Beginning Linux Programming”, UK: Wrox Press, 1997.

[19] A. Glover, P. M. Grant, “Digital Communications”, UK: Prentice Hall, 1998. pp. 658-682.

[20] E. W. K. Liew, “A Distributed Real-Time Operating System for a Multi-Processor StrongARM Network”. PhD thesis, The Nottingham Trent University, February 2002.
[21] H. Schildt, “The Complete Reference: C++”, 3rd Ed., USA: McGraw Hill, 1998. pp. 807-927.
[22] E. G. Walters, “The Essential Guide to Computing: The Story of Information Technology”, Prentice Hall, 2000. pp. 128-129.
[23] C. Xavier, S. S. Iyengar, “Introduction to Parallel Algorithms”, USA: John Wiley & Sons, 1998. pp. 51-55.