Communicating Process Architectures 2009
Peter Welch, Herman Roebbers and Tobe Announced (Eds.)
IOS Press, 2009

An Application of CoSMoS Design Methods to Pedestrian Simulation

Sarah CLAYTON [1], Neil URQUHART and Jon KERRIDGE

School of Computing, Edinburgh Napier University, Edinburgh, EH10 5DT.

Abstract. In this paper, we discuss the implementation of a simple pedestrian simulation that uses a multi agent based design pattern developed by the CoSMoS research group. Given the nature of Multi Agent Systems (MAS), parallel processing techniques are inevitably used in their implementation. Most of these approaches rely on conventional parallel programming techniques, such as threads, Message Passing Interface (MPI) and Remote Method Invocation (RMI). The CoSMoS design patterns are founded on the use of Communicating Sequential Processes (CSP), a parallel computing paradigm that emphasises a process oriented rather than object oriented programming perspective.

Keywords. JCSP, CoSMoS design methods, pedestrian simulation, multi-agent systems

1. Introduction

The realistic simulation of pedestrian movement is a challenging problem, a large number of individual pedestrians may be included in the simulation. Rapid decisions must be made about the trajectory of each pedestrian in relation to the environment and other pedestrians in the vicinity. Parallel processing has allowed such simulations to include tens of thousands of pedestrian processes travelling across large areas. The multi-agent systems paradigm has recently been used to significant effect within pedestrian simulation, typically each pedestrian within the simulation equates to a specific agent. Examples of these are described in [1,2]. In [2], the parallel aspect of the simulation is implemented using MPI.

The Complex Systems Modelling and Simulation infrastructure (CoSMoS) research project [3], a successor to the TUNA [4] research project, is an attempt to develop reusable engineering techniques that can be applied across a whole range of MAS and simulations. Examples include three dimensional simulations of blood clot formation in blood vessels, involving many millions of processes running across a number of networked computers [5,6].

The aims of the CoSMoS project are to provide a 'massively-concurrent and distributed' [7] infrastructure based on a *process-oriented* programming model. This process-oriented paradigm derives from the process algebras of CSP [8] and π-calculus [9]. The purpose of these methods is the elimination of perennial problems in concurrency arising from conventional parallel techniques, such as threads and locks. These problems include deadlock, livelock and race hazards.

Initial demonstrations produced by the CoSMoS research group are implemented using occam-π [10], a language developed to enable the direct implementation of concurrent systems complying with the principles of CSP and π-calculus. To this end, the Kent Retargetable occam Compiler (KRoC) [11] was developed at the University of Kent Computing Labora-

[1]Corresponding Author: *Sarah Clayton, School of Computing, Edinburgh Napier University, Merchiston Campus, Edinburgh EH10 5DT, UK.*. Tel: +44 131 455 2477. E-mail: `s.clayton@napier.ac.uk`.

tory. Similar capabilities are also provided by the Java Communicating Sequential Processes (JCSP) [12] packages, also developed at the University of Kent. As a library for the mainstream Java language, JCSP is more accessible to general programmers. It provides a means of implementing the semantics of CSP, that uses the underlying Java threading model, without the developer needing to be concerned with the details. The work described here has been created using JCSP.

2. CSP and the π-calculus

In this section, we give a brief summary of the main concepts within CSP and the π-calculus. They are part of a rich set of semantics for concurrent programming, made directly available to programmers through the occam-π programming language and the JCSP library for Java.

CSP and the π-calculus are formalisms for designing and reasoning about concurrent systems. Development environments that allow the application of these formalisms encourage a process-oriented, rather than object-oriented approach, to programming.

2.1. Processes

Processes have their own thread of control and entirely encapsulate all their data and maintain their own state. These cannot be altered by other processes [13]. Other processes may send a message to a process requesting that its state be updated; such messages may be refused by the receiving process until it is in a correct state to deal with it. This gives two important wins for concurrent design: firstly, state never changes without the process owning that state making the change and secondly, a process never needs to manage data for requests with which it cannot deal. No locks are needed. Of course, care must be taken to avoid deadlocks caused by the refusal of messages.

2.2. Networks, Channels and Barriers

Processes combine in parallel to form *networks*. Processes interact through synchronising on shared *events*. Instead of being propagated by *listeners*, events in process oriented systems are built upon *channels* and *barriers*. A network of processes is itself a process, so layered architectures – reflecting the layered structures of real life – are simply modelled.

Barriers are a fundamental part of Bulk Synchronous Processing (BSP) [14]. They are directlty modelled by multiway events in in CSP. They allow multiple and heterogeneous processes to synchronise their activities, and enforce lockstep parallelism between them. When a process synchronises on a `Barrier`, it waits until all other processes enrolled on the `Barrier` have also synchronised before continuing processing.

Processes communicate privately to each other using channels. Channels are synchronous. Any process writing to a channel will block until the reader is able to process the message. Although it is possible to implement buffering in these channels (JCSP supports this directly), no buffered channels are used here. `One2OneChannels` allow point to point communication between processes, and `Any2OneChannels` allow many processes to communicate with a single process. Other channel types are available but are not discussed here.

3. Implementation

The structure of the simulation is based partly on the server structure described in [2] and the description of space described by Andrews et al. in [7]. The simulation uses fine grained parallelism, and is scalable. It has three main elements, `Agents`, `Sites`, and an overall `SiteServer`. These are implemented as processes rather than passive objects, and communi-

cate with each other using channels, rather than through method calls. This allows processes and their data to be completely encapsulated. Any change of state or control information is communicated, as necessary, to other processes through channels.

The space to be simulated is modelled by multiple `Site` processes, in a way similar to [7]. The `SiteServer` process acts as a server to all its client `Sites`. A `Site` process acts as a server to an ever-changing set of *mobile* `Agent` clients – see Figure 1. The use of a client-server architecture here eliminates the dangers of deadlock [15].

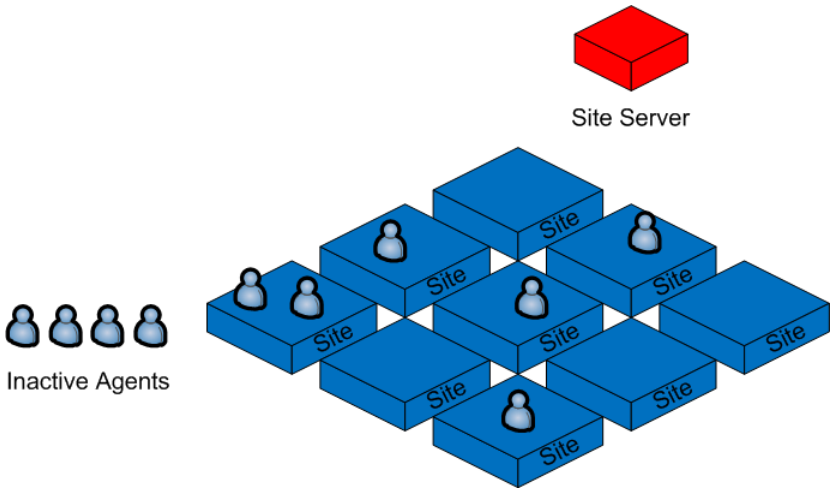

Figure 1. Server layers based on Quinn et al. [2]

`Agents` are mobile across sites: they may deregister themselves from any given `Site` and migrate to another. Each `Site` has a register channel for this purpose, where it receives registration requests from `Agents`, in the form of their server channel ends. This allows communication to be immediately established between the `Agent` as client and the `Site` as server. The process of registration (by a `SiteServer`) is described in the code in Listing 1:

```
private void register()
{
    boolean polling = true;
    timer.setAlarm(timer.read() + timeout);
    Vector<ServerChannelEnd> servers = new Vector<ServerChannelEnd>();
    while (polling)
    {
        int index = alternative.select(); // wait for a timeout or channel register
        if (index == TIMER)
            polling = false;
        else if (index == CHAN)
        {
            ServerChannelEnd s = (ServerChannelEnd) register.read();
            servers.add(s);
            timer.setAlarm(timer.read() + timeout);
        }
    }
    for (ServerChannelEnd newserver : servers)
    {
        newserver.read();
        newserver.write("Hello");
        server.add(newserver);
    }
}
```

Listing 1: Code for client registration

The SiteServer operates in a manner conceptually similar to that described in [2]. Agents communicate their current location to the Site with which they are currently registered. Sites then engage in a client-server communication with the SiteServer, which aggregates all this information. In the next phase of the communication, the SiteServer returns this global information to each Site, which then passes it on to each Agent. Agents then act on this information and alter their current position. This is described in Table 1 below, and compares the three main processes of the simulation. The implementation of the pedestrian simulation is illustrated in Fig. 1.

3.1. Discover and Modify

In order to ensure that all processes are updated and modified in parallel, two Barriers are used: discover and modify. During the discover phase, all Sites are updated by the SiteServer with the global coordinates of every Agent. Each Site then updates all Agents that are registered with it.

As explained in [16,6], autonomous software agents perceive their environment and then act on it. This creates a two phase process for each step of the simulation, discovery and modification, that all processes comply with. These phases are enforced by barriers, described above. The tasks carried out by each type of process for each step of the simulation are described in the Table 1.

Table 1. Processing Sequence

Agent		Site		SiteServer
Synchronise on discover barrier				
		Request global coordinates	→	Receive requests
		Receive global coordinates	←	Send global coordinates
Request update	→	Receive requests		
Receive update	←	Send global coordinates		
Synchronise on modify barrier				
Modify state				
Send state	→	Receive state		
Receive ACK	←	Send ACK		
		Send updates	→	Receive updates
		Receive ACK	←	Send ACK
				Aggregate updates into global coordinates

All communications between processes are on a client-server basis. In effect, a client-server relationship involves the client sending a request to the server, to which the server is guaranteed to respond [17,15]. Processes at the same level do not communicate directly with each other, only with the process at the next level up. As stated in [6]: *"such communication patterns have been proven to be deadlock free"*. See [15] for a proof.

3.2. Description of Space

The division of space between sites allows for a simulation that is scalable and robust, separating out the management of agents between many processes. The `Site` processes themselves have no knowledge of how they are situated. Each `Agent` class has a `Map` object that provides information about the area that a `Site` is associated with and the means with which the `Agent` can register with this `Site`. In this way, `Site` processes can be associated with spaces of any shape or size. These spaces can range from triangles, simple co-planar two dimensional areas, complex three dimensional shapes, to higher dimensions with dynamically forming and shifting worm-holes.

At the edges of each space, an `Agent` may either migrate to the next Site, or encounter a hard boundary, requiring it to change direction. This is determined by the existence of a reference to the adjacent `Site`, if one exists. This is a reference to the `Site`'s register channel, an `Any2OneChannel`, which allows many writers (the `Agents` seeking to register) and only one reader (the destination `Site`).

The register process happens in two phases. First the `Agent` must inform its current `Site` that it wishes to deregister, during the discovery phase. During the modify phase, before any other operation or communication is carried out, the `Agent` writes a reference to its communication channels to the register channel of the new `Site`, and waits for an acknowledgement. In this way, while an arbitrary number of `Agents` may wish to migrate from `Site` to `Site` at any one time, these attempts will always succeed. An image from the software is shown in Fig. 2 below.

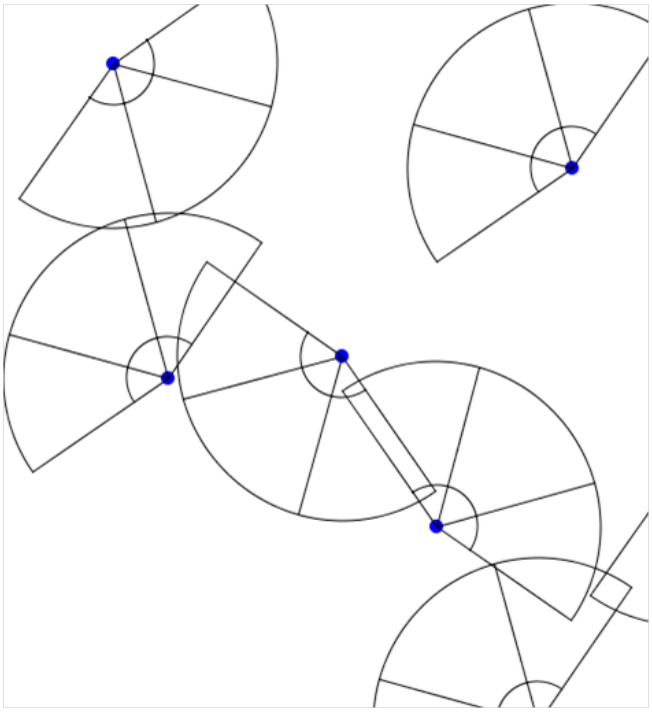

Figure 2. Pedestrian agents in the application showing the arc of their field of vision

The current work implements simple reactive agents. These contain little in the way of intelligence in making their choices. Their field of view replicates that of humans. The span of human vision is 160 degrees. Central vision only occupies 60 degrees of this, with peripheral vision on each side occupying 50 degrees [18]. The minimum distance between agents is delimited by the inner arc of their field of view. Should any other Agent approach this, they will react by choosing a different direction.

4. Results

A number of test runs were performed to evaluate how the simulation performed when the number of agents was incremented. This was done in order to demonstrate the scalability of the system. This test was carried out with one `Site` object, and the number of `Agents` incremented by ten for each run. The results are summarised in Table 2.

Table 2. Results from test runs

Number of agents	Total time	Avg time per step (ms)	Avg time per Agent (ms)
10	151	15.11	1.51
20	412	20.62	1.03
30	845	28.17	0.94
40	1394	34.86	0.87
50	2057	41.13	0.82
60	2853	47.55	0.79
70	3741	53.44	0.76
80	5035	62.94	0.79
90	6207	68.97	0.77
100	7817	78.17	0.78
110	9401	85.46	0.78
120	11111	92.59	0.77
130	12923	99.41	0.76
140	15243	108.88	0.78
150	17454	116.36	0.78
160	20030	125.19	0.78
170	22251	130.89	0.77
180	25443	141.35	0.79
190	28694	151.02	0.79
200	31104	155.52	0.78

As can be seen from Table 2, the time to update each `Agent` during each step of the simulation is more or less constant. This is illustrated in Fig. 3 below.

These average times tend to decrease as the number of `Agents` increase. This reflects the overhead of setting up support processes, such as the display processes. Thereafter, the average times per Agent tend to settle at around 0.78 ms. However, beyond a certain point, the Java Virtual Machine (JVM) is no longer able to allocate any more threads and throws an exception. At the same time, as discussed below, it is unlikely that the number of `Agents` will exceed thirty in this application.

5. Conclusion

In this paper, the application of CoSMoS design patterns in the development of MAS simulations has been discussed. The principles of concurrent processing using non-conventional techniques based on CSP and π-calculus have been explained. The client-server pattern that guarantees livelock and deadlock free concurrency has also been discussed. This offers a firm foundation for future work, using MAS, to simulate pedestrian behaviour.

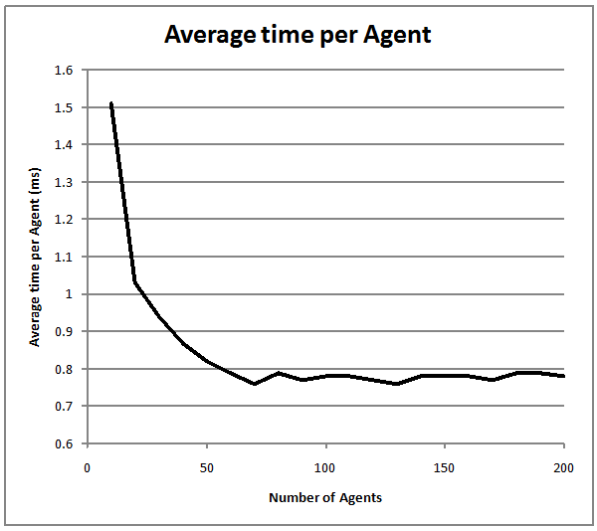

Figure 3. Average update times per Agent (ms) by test run

6. Future Work

Although there is an upper limit to the number of threads the JVM can allocate, it would be possible to increase the number of Agents by distributing the application across a number of JVMs and networked computers. This can be done using the `NetBarrier` feature of the development version of JCSP. However, this would require a redesign of the structure of the application more in line with the occoids example described by Andrews et al. in [7] than the layered server based structure of Quinn et al.'s work in [2].

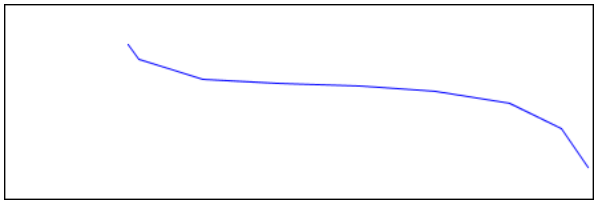

Figure 4. Pedestrian trajectory recorded using infra-red sensors along a $15 \times 4m$ corridor

Although many simple agents have been used to simulate emergent behaviour [19], the purpose of the TRAMP project [20] is the simulation of human behaviour derived from data collected by infra-red sensors. As shown in Fig. 4 actual human movements, when navigating across a space, are described by elegant and coherent curves. This is difficult to replicate using simple agents. In order to achieve this aim, agents trained using Learning Classifier Systems (LCS) [21] will be developed, and their interactions studied. The training data will allow the creation of agents that display realistic behaviours.

The aim of the TRAMP research project is to simulate simple interactions between individuals, such as overtaking, group behaviour and obstacle avoidance, at the microscopic level. Issues of emergence, at the macroscopic level, are not dealt with. The environment being studied is relatively small, a straight corridor measuring 15×4 metres. The density of pedestrians passing through rarely exceeds 30 people. However, this provides a rich set of data on microscopic behaviours.

References

[1] J. Dijkstra, H.J.P. Timmermans, and A.J. Jessurun. A Multi-Agent Cellular Automata System for Visualising Simulated Pedestrian Activity. In S. Bandini and T. Worsch, editors, *Theoretical and Practical*

Issues on Cellular Automata - Proceedings on the 4th International Conference on Cellular Automata for research and Industry, pages 29–36, October 2000.

[2] M.J. Quinn, R.A. Metoyer, and K. Hunter-Zaworski. Parallel Implementation of the Social Forces Model. *Pedestrian and Evacuation Dynamics*, pages 63–74, 2003.

[3] S. Stepney, P.H. Welch, J. Timmis, C. Alexander, F.R.M. Barnes, M. Bates, F.A.C. Polack, and A. Tyrrell. CoSMoS: Complex Systems Modelling and Simulation infrastructure, April 2007. EPSRC grants EP/E053505/1 and EP/E049419/1. URL: `http://www.cosmos-research.org/`.

[4] S. Stepney, P.H. Welch, F.A.C. Pollack, J.C.P. Woodcock, S. Schneider, H.E. Treharne, and A.L.C. Cavalcanti. TUNA: Theory Underpinning Nanotech Assemblers (Feasibility Study), January 2005. EPSRC grant EP/C516966/1. Available from: `http://www.cs.york.ac.uk/nature/tuna/index.htm`.

[5] P.H. Welch, B. Vinter, and F. Barnes. Initial Experiences with occam-pi Simulations of Blood Clotting on the Minimum Intrusion Grid. In H.R. Arabnia, editor, *Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'05)*, pages 201–207, Las Vegas, Nevada, USA, June 2005.

[6] C.G. Ritson and P.H. Welch. A Process-Oriented Architecture for Complex System Modelling. In A.A. McEwan, W. Ifill, and P.H. Welch, editors, *Communicating Process Architectures 2007*, pages 249–266, July 2007.

[7] P. Andrews, A. Sampson, J.M. Bjorndalen, S. Stepney, J. Timmis, D. Warren, P.H. Welch, and J. Noble. Investigating Patterns for the Process-Oriented Modelling and Simulation of Space in Complex Systems. In S. Bullock, J. Noble, R. Watson, and M.A. Bedau, editors, *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pages 17–24, 2008.

[8] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985. ISBN: 0-13-153271-5.

[9] R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.

[10] P.H. Welch and F. Barnes. Communicating Mobile Processes: Introducing occam-pi. In A.E. Abdallah, C.B. Jones, and J.W. Sanders, editors, *25 Years of CSP*, volume 3525 of *Lecture Notes in Computer Science*, pages 175–210, April 2005.

[11] P.H. Welch, J. Moores, F.R.M. Barnes, and D.C. Wood. The KRoC Home Page, 2000. Available at: `http://www.cs.kent.ac.uk/projects/ofa/kroc/`.

[12] P.H. Welch, N.C. Brown, J. Moores, K. Chalmers, and B. Sputh. Integrating and Extending JCSP. In A.A. McEwan, W. Ifill, and P.H. Welch, editors, *Communicating Process Architectures 2007*, pages 349–369, July 2007.

[13] P.B. Hansen. Java's Insecure Parallelism. *ACM SIGPLAN Notices*, 34:38–45, April 1999.

[14] W.F. McColl. Scalable Computing. In *Computer Science Today: Recent Trends and Developments*, pages 46–61, 1996.

[15] J.M.R. Martin and P.H.Welch. A Design Strategy for Deadlock-free Concurrent Systems. *Transputer Communications*, 3(4):215–232, October 1996.

[16] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons, 2002. ISBN 978-0470519462.

[17] P.H. Welch, G.R.R. Justo, and C.J. Willcock. Higher-Level Paradigms for Deadlock-Free High-Performance Systems. In R. Grebe, J. Hektor, S.C. Hilton, M.R. Jane, and P.H. Welch, editors, *Transputer Applications and Systems '93, Proceedings of the 1993 World Transputer Congress*, volume 2, pages 981–1004, Aachen, Germany, September 1993.

[18] V. Bruce, P.R. Green, and M.A. Georgeson. *Visual Perception: Physiology, Psychology, and Ecology*. Psychology Press, New York, 1996. ISBN 1-84169-238-7.

[19] V.J. Blue, M.J. Embrechts, and J.L. Adler. Cellular Automata Modeling of Pedestrian Movements. In *'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on Systems, Man, and Cybernetics, 1997.*, volume 3, pages 2320–2323, Orlando, FL, 1997. IEEE.

[20] S. Clayton, N. Urquhart, and J.M. Kerridge. Tracking and Analysis of the Movement of Pedestrians. In *Third Annual Scottish Transport Applications and Research Conference*, March 2007.

[21] J.H. Holland. Studying Complex Adaptive Systems. *Journal of Systems Science and Complexity*, 19:1–8, March 2006.