A Transactional Architecture for Simulation

Tim Hoverd
Department of Computer Science
University of York
York, UK
Email: tim.hoverd@cs.york.ac.uk

Adam T. Sampson
School of Computing
University of Kent
Canterbury, UK
Email: ats@offog.org

***Abstract*—We are developing a concurrent, agent-based approach to complex systems simulation as part of the CoSMoS project. In such simulations an agent's behaviour can typically be characterised as a series of queries and updates to its environment—a "transactional" pattern of interaction familiar to programmers of database systems. We explore how ideas from the field of databases, such as optimistic approaches to consistency and replication, may profitably be applied to the field of simulation, and how the constraints of modern databases can be relaxed to yield better performance while maintaining simulation validity.**

***Keywords*-science; simulation; concurrency control; database concurrency operations**

I. Introduction

The CoSMoS project[1] is developing an approach to agent-based modelling and simulation of complex systems using concurrent software engineering techniques. Using the CoSMoS design process, a complex system is modelled as a collection of interacting, concurrent agents, each of which may have its own behaviour. CoSMoS has been especially concerned with systems that demonstrate *emergent* behaviour, where complex group behaviours arise from simple rules followed by individual agents: for example, Reynolds' boids [1] are simulated birds that form flocks as an emergent behaviour.

The initial simulations built as part of the TUNA project [2] used the simple approach of modelling interactions between agents directly as channels or method calls. As more complex simulations were constructed, the lack of structure to the communications made it increasingly difficult to reason formally about the behaviour of the simulations. Model-checking is one way to guarantee correctness of a concurrent simulation [3]—but this is often awkward, because model-checking languages are not very expressive, and intractably slow for even modest-sized systems. A better approach is to use a structural design rule such as I/O-PAR [4], which describes how to construct components of a parallel system that, when used to construct a complete system, will guarantee that the system as a whole is free from deadlock and livelock problems. However, the design rules applicable to systems in which any component may communicate with any other in an unstructured fashion allow only very simple patterns of behaviour.

[1] `http://www.cosmos-research.org/`

A. Environment orientation

To enable the engineering of more complex simulations as part of the TUNA and CoSMoS projects, we needed to introduce a common structure to the interactions between agents [5]. Our approach is based on the observation that the agents in complex systems—for example, flocking birds and trail-following ants—do not communicate directly with each other, but interact through the medium provided by their environment. Agents place information about their current state into the environment, and receive information about other agents—often in a highly-filtered form—from the environment. We call this approach *environment orientation* [6], because it requires us to model our agents primarily in terms of their interactions with the environment.

This approach is familiar in the context of real-world communication using stigmergy [7]—but we argue that all communication in a complex system can be considered as being mediated by the environment. Information placed in the environment may persist for some time, like the pheromones used by ants to form trails, or be transitory, such as the "photons" used to simulate visibility among a flock of birds.

Each agent in an environment-oriented simulation has both *internal state*, private to that agent, and *external state* which is made visible to other agents through the environment—how the agent "wishes to be seen" by other agents. Each agent follows a simple cycle of behaviour:

- retrieve information from the environment;
- compute new values for its internal and external state;
- publish its external state to the environment.

The regularity of the interactions between agents and their environment permits a wide variety of implementations; for example, we can construct massively-concurrent simulations using the client-server pattern [4], [8]. (The introduction of the environment as an explicit mediator is a common pattern in the process-oriented programming paradigm, used to simplify interactions between concurrent processes when there is no natural ordering between them.)

We have already made practical use of this approach to simplify the implementation of several concurrent, distributed CoSMoS simulations [9], [10], and we plan to continue its use in future case studies. In the remainder of this paper, we will consider further some of the implications of environment orientation on the construction of complex systems simulations.

II. Transactions and locking

Environment orientation can be considered as a *transactional* approach to complex systems simulation. Each agent in a simulation is responsible for maintaining the information published about itself. It does this by performing a sequence of transactions against the environment, which is a shared database of external states: during each cycle, an agent will *query* the environment for the external states of the other agents it is interested in, and *update* the environment with its new external state.

As the external state stored for each agent is only written to by that agent, there is no possibility of lock contention when updating the environment. That is, as long as agents reading the contents of a particular agent's external state are prevented from viewing inconsistent state information while an update is in progress—that is, updates to the environment are atomic—then no further locking mechanism is necessary. Several approaches exist to ensure this sort of consistency: we can either make individual updates atomic, using the same techniques that conventional databases use to implement read-committed transaction isolation, or we can use *phase synchronisation* between concurrent agents to enforce write-before-read ordering [11].

In addition, these atomic updates of external state by individual agents mean that it is never necessary to roll back a transaction. We observe that this is also true of real-world complex systems: agents in the real world are not able to roll back their changes to the environment.

With such an overall design, there are many possible implementations of the state database used in the complex systems simulation, including relational database management systems and Linda-like [12] tuple spaces. Another possibility is software transactional memory [13], an approach to concurrent programming which makes database-like transactional operations available on shared memory. While the guarantees of full atomicity for multi-step operations that STM provides are not required for environment-oriented simulation, the STM approaches to lock-free atomic memory updates to shared memory on modern multicore systems are directly applicable to fine-grained simulation, without the considerable overhead of rollback.

III. Embodiment

In an environment-oriented complex systems simulation, the environment is responsible for managing the state database and providing those facilities that are *embodied* [14] by the environment. Typically, the environment embodies aspects of real-world physics. For example, [6] describes how these embodied services may result in information being presented to agents, the clients of the environment, using various topological representations of the information.

The choice of properties to embody within the environment, though, is not always clear. For example, consider the implementation of pheromone trails in a simulation of ant foraging, in which trails are laid by ants, and fade away as time progresses. Trails could be implemented as agents, spawned into the simulation by ants, which maintain a pheromone level at a particular physical location as part of their external state; the behaviour of the pheromone agent would simply be to reduce its level periodically, and exit once the pheromone was exhausted. Alternatively, the trails could be embodied within the environment: the environment itself would "understand" that the pheromone levels represented an aspect of the physical world, and would manipulate the levels itself to implement decay.

There are trade-offs to be made here in terms of simplicity and generality, and in terms of efficiency. If all physical properties are implemented using specialised agents, the external state database is effectively just a tuple store (albeit one that will be accessed by, perhaps, many simulated agents executing across a distributed system); this simplifies its implementation and makes it directly applicable to all types of environment. However, these agents increase system load, and may require different patterns of interaction with the environment, complicating the communication patterns of the simulation—for example, you may need to ensure that all environmental properties have been updated before agents can observe the environment. On the other hand, if properties are implemented by the environment itself, this does not complicate the patterns of interaction between agents and the environment—but it requires the environment to be aware of the details of these properties, reducing its generality. It may prove convenient to strike a balance between the two in a practical simulation framework: implement a few common physical properties in the environment itself, but make it possible to extend the simulation with additional properties by writing specialised agents.

IV. Time and fairness

In a simulation following the I/O-PAR design rules, the communications between the agents give the simulation as a whole a shared sense of granular time: no agent can proceed to the next time step until it has communicated with all its neighbours. When agents do not directly communicate with each other, this shared sense of time is lost: agents can perform transactions whenever they like, which makes it possible for agents to execute at different virtual rates.

This is, of course, what happens to real-world complex systems agents as is discussed in [6]. However, in the real world, no agent can "run ahead" of the others; agents execute in a perfectly fair parallel manner, with the rate of their behaviour only limited by the inherent physical properties of the world. In a simulation, freewheeling is not acceptable: we must introduce a sense of time in order to ensure that agents' access to the shared computational resources is scheduled fairly, with no agent able to starve another of execution time.

In the current CoSMoS simulations we provide the shared sense of time by an explicit barrier synchronisation at the end of each timestep [11], an approach inspired by Bulk Synchronous Parallelism [15]. We can further subdivide each timestep to control access to shared resources in one or more of a set of predefined "phases".

We plan to improve performance for phase-based simulations by combining the "virtual time" technique used in event-based simulation [16], in which a unitless *virtual time* is represented simply as a monotonically-increasing tag tracked by simulation components, with the phase-regulating "clock" primitives provided by the X10 programming language [17]. Our clocks will keep track of a time- and phase-ordered list of future events; this will allow agents to run in strict time order with the maximum possible concurrency for each time step, while avoiding the unnecessary synchronisations that are a common problem in phase-based programs.

However, the problem with all these approaches is that they offer an inherently discretised representation of time. Discretising time has been shown to affect the accuracy of complex systems simulations in the same way that discretising space does [18], and results in reduced expressiveness of simulation systems [19]; we feel that continuous time is an extremely useful feature for complex systems simulations.

Furthermore, we must support these notions of continuous or discretised virtual time across non-uniform multicore and distributed systems—a problem that is hard to solve efficiently. Within the CoSMoS project we are investigating the extent to which this can be done using "sloppy" synchronisation: allowing the clocks on neighbouring systems to drift away from each other within predefined tolerances.

V. Reproducibility

Complex systems simulations are generally seen as providing the ability to perform completely reproducible experiments—providing a clear advantage over experimentation on real-world complex systems, where reproducibility is generally not feasible, and allowing published results to be directly reproduced by other researchers. However, several of the techniques we have described introduce nondeterminism into our simulations: for example, concurrent transactional state updates without external synchronisation allow updates to occur in any order, and "sloppy" implementations of time synchronisation in a distributed simulation may trade off consistency against performance.

We believe that a degree of nondeterminism will be acceptable in many circumstances. Simulation can be considered as a scientific instrument that we use to understand the behaviour of a system, and like all instruments it has a degree of uncertainty in its results that can be established by calibration. The scientific method is very good at dealing with real-world experiments with nondeterministic behaviour; the same techniques—error bounds, sensitivity analysis, and other statistical techniques—can be applied to interpret the results of nondeterministic simulations.

It is rare that the results of a single run of a complex system simulation are directly useful, just as the result of a single real-world experiment is rarely considered sufficient. We normally want to run our simulation many times with the same parameters, and aggregate the results to give a better understanding of the typical behaviour of the system; this will have the effect of "averaging out" the effects of nondeterminism on the individual results. In addition, permitting a greater degree of nondeterminism will generally speed up the simulation, making it practical to run it more times—and, unlike in the real world, we can ensure that the initial conditions for a set of experiments are always exactly the same.

When debugging a simulation, being able to reproduce a single run exactly is sometimes useful. To support this, we could give the programmer a control that allows them to trade determinism against performance—for example, by reducing the degree of concurrency and enabling additional explicit synchronisations when greater determinism is required. Furthermore, techniques exist for debugging nondeterministic concurrent systems [20] where software is instrumented so that a rough trace of its execution path is retained. Subsequent debugging runs can then be automatically steered down the same execution path, with the trace being iteratively refined with feedback from the programmer until the desired behaviour is reproduced. This approach could be applied to a nondeterministic simulation.

Fundamentally, we believe that a complex systems simulation will still be *more* reproducible (i.e. have less variance in its results) than a corresponding real-world experiment—although it is still important to construct a reasonable validity argument for any simulation. Given the inherent inaccuracy in most existing simulations, we believe that encouraging scientists to reason about the accuracy of results obtained via simulation in the same way that they would for "wet" experiments is appropriate.

VI. Robustness

The real world—the most complex of all complex systems—is inherently extremely robust. Systems composed of a huge number of independent agents from which useful behaviour emerges will usually continue to function in a

wide range of different circumstances: agents are born and die, information is delayed, lost or corrupted, and interactions are complex and unpredictable. While undesirable behaviours exist, such as the auto-immune diseases that appear in organisms equipped with complex immune systems, they are rare; complex systems usually return themselves to some kind of stable state.

Engineered systems fare poorly by comparison, usually demonstrating extreme sensitivity to both initial and changing conditions. Systems built by humans tend to be fragile, with a wide variety of spectacular failure modes [21]. This applies even to existing simulations of complex systems, which often display high sensitivity to parameter values and to implementation details such as the scheduling order of concurrent processes.

An ideal complex systems simulation would be as robust as the complex system itself. In our transactional approach, each agent's behaviour is largely independent of the other agents in the simulation, and is not affected by the heavy hand of precise time steps and explicit global synchronisation. We believe that this decoupling should tend to increase the robustness of our simulations, and that we should actively try to build robust simulations by testing the sensitivity of our simulation to artificially-induced "faults" such as blocked access to parts of the simulation, changes to the order in which state updates are recorded, and forcible introduction or termination of agents. We are investigating such systematic manipulations of our simulations with the ultimate intention of being able to *measure* the robustness of a particular simulation.

VII. Conclusion

We have described a "transactional" approach to the simulation of complex systems, based on our experience of existing approaches to simulation, and we have outlined some of the implications that the use of this approach would have on the construction of simulations. Many of the individual techniques we have described are already being used successfully during the construction of simulations for the CoSMoS project.

We are presently building new concurrent simulations using a "purely transactional" approach to implementation. While our initial results have been encouraging, so far we have only considered systems at a relatively small scale without distribution; we plan to extend our work to larger systems to further test the effectiveness of our approach.

Acknowledgements

We would like to express our thanks to other members of the CoSMoS project, and in particular to Susan Stepney of the University of York for many helpful discussions. This work is part of the CoSMoS project, funded by EPSRC grants EP/E053505/1 and EP/E049419/1, and a Microsoft Research Europe PhD studentship.

References

[1] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987. [Online]. Available: http://citeseer.ist.psu.edu/reynolds-flocks.html

[2] "TUNA: Final Report," 2007.

[3] S. Schneider, A. Cavalcanti, H. Treharne, and J. Woodcock, "A layered behavioural model of platelets," in *ICECCS 2006*. IEEE, 2006.

[4] P. H. Welch, G. R. R. Justo, and C. J. Willcock, "Higher-Level Paradigms for Deadlock-Free High-Performance Systems," in *Transputer Applications and Systems '93*. IOS Press, 1993, pp. 981–1004.

[5] C. G. Ritson and P. H. Welch, "A process-oriented architecture for complex system modelling," in *Communicating Process Architectures 2007*, vol. 65. IOS Press, 2007, pp. 249–266.

[6] T. Hoverd and S. Stepney, "Environment orientation: an architecture for simulating complex systems," in *Proceedings of the 2009 Workshop on Complex Systems Modelling and Simulation*, S. Stepney, P. H. Welch, P. S. Andrews, and J. Timmis, Eds. Luniver Press, 2009, pp. 67–82.

[7] G. Theraulaz and E. Bonabeau, "A brief history of stigmergy," *Artif. Life*, vol. 5, no. 2, pp. 97–116, April 1999. [Online]. Available: http://dx.doi.org/10.1162/106454699568700

[8] J. M. R. Martin and P. H. Welch, "A design strategy for deadlock-free concurrent systems," *Transputer Communications*, vol. 3, no. 4, 1997.

[9] P. S. Andrews, A. T. Sampson, J. M. Bjørndalen, S. Stepney, J. Timmis, D. N. Warren, and P. H. Welch, "Investigating patterns for the process-oriented modelling and simulation of space in complex systems," in *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, S. Bullock, J. Noble, R. Watson, and M. A. Bedau, Eds. MIT Press, Cambridge, MA, 2008, pp. 17–24.

[10] P. S. Andrews, F. A. C. Polack, A. T. Sampson, J. Timmis, L. Scott, and M. Coles, "Simulating biology: Towards understanding what the simulation shows," in *Proceedings of the 2008 Workshop on Complex Systems Modelling and Simulation, York, UK, September 2008*, S. Stepney, F. Polack, and P. Welch, Eds. Luniver Press, 2008, pp. 93–123.

[11] F. R. M. Barnes, P. H. Welch, and A. T. Sampson, "Barrier synchronisation for occam-pi," in *2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. CSREA Press, 2005, pp. 173–179.

[12] D. Gelernter, "Generative communication in Linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, pp. 80–112, January 1985. [Online]. Available: http://dx.doi.org/10.1145/2363.2433

[13] N. Shavit and D. Touitou, "Software transactional memory," in *PODC '95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 1995, pp. 204–213.

[14] S. Stepney, "Embodiment," in *In Silico Immunology*, D. Flower and J. Timmis, Eds. Springer, 2007, ch. 12, pp. 265–288.

[15] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach using BSP and MPI*. Oxford University Press, 2004.

[16] D. R. Jefferson, "Virtual time," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 3, pp. 404–425, July 1985. [Online]. Available: http://dx.doi.org/10.1145/3916.3988

[17] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar, "X10: an object-oriented approach to non-uniform cluster computing," in *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA: ACM, 2005, pp. 519–538.

[18] I. Cohen and D. Harel, "Two views of a biology-computer science alliance," in *Proceedings of the 2009 Workshop on Complex Systems Modelling and Simulation*, S. Stepney, P. H. Welch, P. S. Andrews, and J. Timmis, Eds. Luniver Press, 2009, pp. 1–8.

[19] C. Elliott, "Why program with continuous time?" http://conal.net/blog/posts/why-program-with-continuous-time/, accessed on January 13th, 2010.

[20] S. Park, Y. Zhou, W. Xiong, Z. Yin, R. Kaushik, K. H. Lee, and S. Lu, "Pres: probabilistic replay with execution sketching on multiprocessors," in *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. New York, NY, USA: ACM, 2009, pp. 177–192.

[21] Risks, "Forum On Risks To The Public In Computers And Related Systems," http://catless.ncl.ac.uk/Risks, accessed on 2nd November, 2009.

[22] S. Stepney, P. H. Welch, P. S. Andrews, and J. Timmis, Eds., *Proceedings of the 2009 Workshop on Complex Systems Modelling and Simulation*. Luniver Press, 2009.