

# Smooth Scaling Ahead: Progressive MAS Simulation from Single PCs to Grids\*

Les Gasser, Kelvin Kakugawa, Brant Chee, and Marc Esteva

Graduate School of Library and Information Science  
University of Illinois at Urbana-Champaign  
{gasser|kakugawa|chee|esteva}@uiuc.edu

**Abstract.** The emerging "Computational Grid" infrastructure poses many new opportunities for the developing science of large scale multi-agent simulation. The ability to migrate agent experiments seamlessly from simple, local single-processor development tools to large-scale distributed simulation environments provides valuable new models for experimentation and software engineering: first develop local, flexible prototypes, then as they become more stable progressively deploy and experiment with them at larger scales. Currently this kind of progressive scalability is hard for both practical and theoretical reasons: Practically, most agent platforms are designed for just one environment of operation. Smooth scalability is more than a matter of increasing agent numbers. Smooth scaling requires clear integration and consistent alignment between a variety of MAS system and simulation architectures and differing underlying infrastructures. This paper reports on recent progress with our experimental platform MACE3J, which now simulates MAS models seamlessly across a variety of scales and architecture types, from single PCs, to Single System Image (SSI) multicomputers, to heterogeneous distributed Grid environments.

## 1 Introduction

The emerging "Computational Grid" infrastructure [1, 2] poses many new opportunities for the developing science of large scale multi-agent systems. The ability to migrate agent simulations and software seamlessly from simple, local single-processor development tools to large-scale distributed experimental environments will provide valuable new software engineering models for agent-based systems. These new models will couple incremental development of agent simulations with controlled testing and execution. The issue of controlling test and execution is critical because of the complexity of agents as "software engineering units" [3] and because of the need for robustness as experiments are scaled up:

---

\* Work supported by NSF Grant 0208937 and by Fulbright/MECD postdoctoral scholarship FU2003-0569

the control gained through deterministic simulation is a key engineering technique for validating large-scale distributed multi-agent systems and simulations whose behavior can't be easily captured analytically for reasons of complexity.

Scalability is often approached as a problem of size—number of agents, for example. Size *is* a dominant dimension for some types of large distributed systems. For instance, typical program structures used in large-scale distributed scientific computing—a principal target of the Grid initiatives—employ homogeneous, non-interacting components, such as the typical, regularly-structured “data-parallel” codes found in many data analysis packages, and the assumption that software decision logic is stable while data changes from run to run. In these cases, more processing resources translates directly into the ability to process more data and solve “larger” problems.

For large systems with *dynamic, heterogeneous, interacting* components such as MAS simulations, scalability raises several other issues. First, scalability requires having appropriate programming models and tools for building and/or integrating many heterogeneous agents. These tools must manage distributed execution resources and timelines, enforce full encapsulation of agents, and offer tight control over message-based multi-agent interactions. The most useful and general programming tools meeting these requirements are based on *distributed object models* such as the actor model [4]. However, in their pure form distributed object programming models introduce subtle constraints on agent and system construction and operation, which create some difficulties for developing scalable, controllable heterogeneous agent simulations. We detail many of these below.

Second, making simulations smoothly scalable requires aligning simulation models to a variety of programming infrastructures and to a different variety of execution system architectures so that it is possible to conveniently deploy, manage, and control large collections of heterogeneous simulated agents across the different resource pools. For example, simulations designed and run on individual PCs will not have truly concurrent execution, since any single processor necessarily serializes behaviors (see below). In contrast, the so-called *Single System Image (SSI) cluster* technique binds together a collection of resources to create an abstraction of a distributed system as a unified, single-point-of-access pool of concurrent execution resources or threads. At first, this seems like an almost ideal programming abstraction for concurrent agents, because it hides details of resource allocation and agent deployment. However, there are critical tensions between the monolithic *sequential* programming approach offered by the single PC environment, and the actual affordances of other distributed platforms across which multi-agent simulations must be developed and deployed to achieve large scales, such as the monolithic *concurrent* programming approach offered by SSI, and the *heterogeneous, distributed, concurrent* environment of actual deployed Grids. Grids in existence (e.g., Teragrid [5]) are large and diverse collections of execution resources, with heterogeneous, not regular architecture. Care must be taken to align models based on PC and SSI environments to heterogeneous and fully-distributed environments.

Finally, issues such as infrastructure reliability, functional completeness, and the state of documentation for some kinds of environments including SSI and Grids can be problematic—Grid services and technologies are a case in point, and this state-of-the-art must be accommodated for realistic agent systems.

In the light of these issues, this paper reports on current developments in our MACE3J experimental platform [6]. The principal innovation reported is the ability to abstract away many of these underlying issues of heterogeneity. This abstraction is accomplished with tools that support seamless deployment of large-grain simulation-based multi-agent systems progressively across a variety of system architectures, ranging from single PCs through SSI clusters to fully distributed Grid environments (in our case based on the Globus Toolkit and the Open Grid Services Initiative) [7, 8]. Below we report on the general structure of the MACE3J system that accomplishes this goal. We next introduce several key dimensions across which multi-agent system simulations need to adapt, and we use these as the basis of a taxonomy of progressively scalable architectures that support MACE3J. We illustrate our progress with reference to several deployed experiments that run across all architectures, and show how making these experiments led to insights about the necessary abstractions.

## 2 MACE Overview and Design Philosophy

MACE3J is a scientifically oriented multi-agent testbed whose design philosophy is driven by three objectives [6]. We state these here because they are strongly impacted by the need for smooth scalability across architectures.

**1. Repeatability and control:** MACE3J should support control and randomized repeatability in simulations, which is useful for both development and experimentation.

**2. Transitionable Models:** Agents should be built of components that can be transitioned from simulated implementations or environments to real ones.

**3. Generation of knowledge about behavior and structure:** MACE3J should support instrumentation that gathers and analyzes data generated by agents and system behaviors.

### 2.1 Understanding Simulation

A significant focus of MACE3J is simulation support. We view simulation support as the provision of four interlocking types of facilities. MACE3J provides all four of these.

1. *Modeling facilities* capture characteristics of modeled systems such as agents or environments in codes that integrate easily with the activation, coordination, and data-gathering services below. MACE3J generalizes the concept of “agents” to *ActiveObjects*, which are defined in MACE3J with a set of interfaces. The *ActiveObjects* concept captures core functionality that allows for implementation of many different types of “agents”, so we

use the term to denote the foundations for a range of typical agent types. MACE3J modeling facilities include reusable components for constructing ActiveObjects, environments, and experiments, coupled with the ability to flexibly import these components and models from other projects.

2. **Coordination facilities** provide coherence and synchronization for the distributed objects that make up the MAS model. This includes a selectable combination of deterministic (simulation-driven), user-driven, environment-driven, and/or probabilistic control of simulation events, which allow simulations to be re-run exactly, while supporting probabilistic control of behavioral and timing aspects of simulations such as message delay and system failure (e.g., failure of message delivery or of execution). In MACE3J the fundamental coordination object is called `ActivationGroup`. `ActivationGroup` holds a timeline and a set of coordination routines that control the overall execution profile of a simulation.
3. **Activation services** that provide enactment (computing) resources for the distributed objects that make up the model. (Of course, these activation services assume an underlying infrastructure such as a single processor PC and SSI cluster, or a Grid environment.) This includes flexible control and steering of simulations through active user involvement in changing simulation parameters at run-time (blurring the distinction between simulation and enactment and facilitating agent transitions to application).
4. **Flexible data gathering, management, analysis, and presentation** is done through user-defined and system-defined probes and data streams.

As a simulation development environment, a key objective of MACE3J has been seamless transitioning of models across execution environments. This kind of model retargeting is valuable as an implementation technology and as a software engineering approach: start simple, validate, and expand to more sophisticated environments while exploiting new capabilities. The single processor (possibly threaded) PC platform is stable, well understood, and controllable, but limited in resources. Agent systems can be developed and prototyped rapidly on the single-processor PC platform because it is highly controllable and accepts heterogeneous, changing codes. In contrast, the Grid is less well understood, less flexible, and works best with more homogeneous and stable codes because of the overhead of distribution, startup, and coordination and because of the underlying heterogeneity of the Grid resources. Thus, we need a progression of different development environments and the ability to link them together in a rational, exploratory development process.

The aims of the tools that support such a progressive development approach are these:

1. Provide a simple, direct system model and API to enable maximum flexibility in Agent styles and granularity.
2. Minimize work for users by providing facilities for distributing, deploying, and controlling agent models.
3. Make coordination lightweight, by abstracting the simulation coordination to simple message patterns implemented in infrastructure.

4. Exploit features of existing platforms such as Grid toolkit services to provide agent simulation layer services, to the extent possible. Current examples include deployment services, directory services and communication services.

### 3 Managing Uncertainty in Scalable MAS Simulations

Here we introduce and develop two main sources of uncertainty for managing design and development of large-scale MAS simulations, with greatest relevance in situations where scale and complexity are related: concurrency and distribution. Concurrency and distribution are inherent properties of MAS, and they introduce several kinds of uncertainty into MAS behaviors.

Concurrency introduces event-ordering uncertainty because concurrently running agents execute at arbitrary rates relative to each other. For a MAS with interactions, increasing scale can increase uncertainty in the ordering of important (interactive) events.

Distribution across space and/or time<sup>1</sup> introduces two types of uncertainty. *Decision uncertainty* occurs when information about the states of remote entities (other agents or environments) that could influence local decisions are inaccessible because those states are distributed. *Semantic uncertainty* occurs when distribution causes agents to translate communicated references or objects into local interpretations that may vary by local context (e.g., [10]). These kinds of uncertainty are fundamental to MAS. Design tools and processes that help control and incrementally modulate distribution- and concurrency-induced uncertainty ease the complexity of engineering and simulating multi-agent systems.

One aim of deployed MAS is to be able to operate in the presence of these types of uncertainty. However, verification of this is hard to do. Our approach is to manage these types of uncertainty by building into middleware support for strategies of progressive, incremental relaxation of control over uncertainty, to gain confidence and experience.

**Incremental management of event-ordering uncertainty:** Concurrent execution of agent programs introduces uncertainty about the ordering of interactive agent events such as communications. Event ordering can significantly impact computation results in general, so this uncertainty can have large effects on system reliability, traceability, verifiability, and understandability. Thus, one approach to managing design complexity and improving confidence in MAS behavior is to first eliminate uncertainty in the ordering of events by making events completely repeatable and deterministic. This strict control can then be progressively released to explore system behavior and build confidence under increasing levels of event-ordering uncertainty. In this way, as a MAS experiment is developed, it can be moved to progressively more complex execution environments with progressively greater degrees of freedom in event ordering due to concurrent execution. MACE3J has two ways for exploiting this progressive approach to temporal control. First, time- and event-coordinating middleware combines

---

<sup>1</sup> Other dimensions of distribution beyond space and time are also introduced in [9].

explicit event control with the ability to change architectures. Second, the architectural abstractions of MACE3J allow designers to shift models to progressively more distributed underlying computational architectures that can supply progressively more concurrent resources (such as more processors), and that exhibit behavior closer and closer to the uncertainty of 'real' environments.

**Incremental management of distribution uncertainty:** Under true distribution, agents may not be able to access information about the internal states of other agents or of their environments. Hewitt called this the problem of “arms-length relationships” [11], and Lesser and colleagues represented it explicitly using partitioned global system models in the TAEMS modeling and simulation approach [12]. A TAEMS model holds an omniscient global view of a problem space and its constraints, while individual agents hold only partial local views. By contrasting the content and accuracy of partial local views with the omniscient global view, an experimenter can measure precisely where her agent control and information sharing strategies have succeeded or failed. Some simulation infrastructures—namely, shared-memory ones—make it far easier than others to model global shared knowledge and partial access to it. However real agent systems, as well as simulations whose aim is to explore actual runtime conditions of distributed agents, cannot rely on shared infrastructure variables. Specifically, multi-agent implementations that rely *at all* on pointers and/or shared data/variables rather than pure messages for agent interactions will not transfer to truly distributed cases. In this sense, a distributed infrastructure acts as a validation tool for the distributability of agent architectures and agent interactions, keeping them “honest.” Thus we can use MACE3J’s ability to smoothly scale across multiple infrastructures as a tool for exploring and validating a system’s ability to manage distribution-caused uncertainty.

## 4 Progressive Scaling

In MACE3J we combining support for these two approaches in one middleware layer, allowing designers and experimenters to move a MAS across a following spectrum of environments and control regimes that vary in their degree of actual or apparent concurrency and their degree of distribution. There are six cases:

**a) Deterministic single processor, single threaded, shared-memory testbeds** which strictly control all temporal progress and inter-agent interactions for completely repeatable performance. In this case, MAS application level coordination mechanisms can be explored, tuned, and verified deterministically against global states, at a cost of realism and the challenge of real variance.

**b) Randomized deterministic single processor, single threaded, shared-memory testbeds** which, again, control all temporal progress and inter-agent interactions for completely repeatable performance. By randomizing schedules and interaction order, some useful aspects of true concurrency can be achieved statistically. MAS application level coordination mechanisms can again be explored, tuned, and verified deterministically against global states, at a cost of realism and the challenge of real variance.

c) **Synchronized multiprocessor, multi-threaded, shared-memory testbeds**, in which time and interaction are controlled explicitly and flexibly. In this case, some aspects of time and concurrency (e.g. specific event and interaction types) can be left uncontrolled in MACE3J. Processing concurrency can be exploited to speed up development and testing as needed. The shared-memory aspect still allows for arbitrarily complete control and measurement of any interaction or temporal step that is desired.

d) **Unsynchronized multiprocessor, multi-threaded, shared-memory testbeds**, in which time and interaction are controlled explicitly and flexibly, and in which MAS application level coordination mechanisms can be explored and verified. In this case, all aspects of time and concurrency must be left uncontrolled. Processing concurrency can be exploited to speed up development and testing as needed. The shared-memory aspect still allows for arbitrarily complete control and measurement of any interaction or temporal step that is desired.

e) **Synchronized multi-threaded, distributed processor, *distributed memory testbeds*** (e.g., Grid) in which time and interaction are controlled explicitly and flexibly. In this case also, some aspects of time and concurrency can be left uncontrolled. However, the distributed-memory, message-passing aspect adds the requirement of “purer” distributed object techniques (e.g., eliminating shared variables, distilling communications contents to serializable media such as strings). The appropriate middleware infrastructure still allows for arbitrarily complete and repeatable control of any interaction or temporal step, so as to verify behavior and results while approaching realistic execution (temporal, representational, and resource) environments.

f) **Unsynchronized multi-threaded, multiprocessor, distributed memory testbeds** (Grid) in which time and interactions are controlled only through MAS application level coordination mechanisms. (This is the desired end-state for MAS.)

## 5 Experiences and Discussion

We have used MACE3J in each of the settings mentioned above for a variety of simulation studies, controlling and progressively releasing various aspects of execution and interaction as described above. Details of some of the sample problem scenarios used can be found in [6]. Currently our most sophisticated agent simulation is the TaskModel experiment. TaskModel takes an arbitrary graph of interdependent tasks (a *workflow*) and maps them to an arbitrary set of agents for execution. Simulation drives an arbitrarily-timed set of *problem-instances* through the task network. Agents use a base of *local organizational knowledge* to reason about task interdependencies, and dynamically reallocates tasks to agents following specifiable *task migration* and *task allocation* policies. Varying any of these dimensions modulates the complexity, difficulty, and scale of the experiment.

Below we discuss several of the key problems that have emerged in using MACE3J across these scenarios and environments, and how we handled and

learned from them. For reasons of length, here we focus only on the Grid infrastructure case as it is the most sophisticated of the environments we've explored.

## 5.1 Grid Rationale and Experiments

Grid services are a new technology for Internet-scale distributed computing [1]. While Grid services are based on the concept of *Web services*, Web services are stateless and non-transient, and this makes them ill-suited as hosts for distributed object simulation components such as agents. Grid services, do, however, have several important features that make them particularly suited for distributed computing, including service factories and lifecycle management, which allow distributed objects (agents) to be instantiated and have a controllable level of persistence. Other key advantages of Grid services are their platform- and language-independence, which have always been important considerations in the MACE3J system.

We use Grid services as a distributed computing technology to expose and connect MACE3J `ActivationGroup` and `Agent` objects through the Internet. There are many advantages to this approach. In particular, the toolkit we are using (Globus' GT3 [7]) employs the standardized Grid service specification that covers a wide array of features, including security, indexing, and management of services. In addition, GT3 allows rapid construction of services without the complexity of handling WSDL service descriptions. GT3 encompasses the programming and hosting environment for Grid services, which allows services to be built and deployed in a relatively seamless fashion.

A high-level overview of the mechanics of running an agent experiment can be broken down into three phases. The first phase involves setting up the Grid hosting environment, by deploying the MACE3J services to the Grid installation. The next phase is running the hosting environment and exposing the `ActivationGroup` and `Agent` services on the Internet. The final phase is to initialize and run a simulation experiment.

## 5.2 Implementing agents as "pure" distributed objects

Perhaps the most significant problems we encountered in moving across single PC environments, shared-memory SSI clusters, and distributed Grid environments were failures arising from incomplete or "impure" distributed object models. These led to significant reimplementations in both simulation and middleware components. Four areas of concern arose in our experiments: object identity, inter-object communication, replication, and enforcement of encapsulation.

**Identity for distributed Grid-based objects:** In the Grid, local objects contain proxy objects that "stand in" for remote partners. Local proxies translate local method calls to remote communications and translate results back. This one-step removal from direct interaction yields a disjointed and asynchronous view of remote agents in the system, fostering difficulties in low-level areas like creation and destruction of agents.

**Inter-agent communication:** Inter-agent communication problems arose because of the structure of Messages in original versions of MACE3J. Originally a *Message object* encapsulated both pertinent metadata (sender, receiver, time) and message content objects. Using message content objects makes message management much simpler in the local-memory (PC or SSI) infrastructure, and it has the advantage of being able to transfer arbitrary objects—e.g., documents or programs—directly in messages. However, local objects are not communicable in Grid messages, for two reasons. First, message content objects may have arbitrary and context-dependent boundaries (e.g., hash tables with runtime-loaded objects as contents; method or field definitions inherited at runtime) so a message sender must incorporate a dynamic theory of object “edges”, and the context for an object is hard or impossible to package and communicate remotely. This means objects sent to remote contexts have uncertain interpretation (execution) semantics when they arrive<sup>2</sup>. The solution is to reduce context by restricting messages to be strings. Abstractly speaking, strings are also freighted with contextual baggage, but the problem of string interpretation is a user-level issue, not a system-level one. Thus we reworked the communication protocol to be more “pure” and use only Strings as media.

**Information Replication:** The third problem was information replication. In MACE3J, the central axis of coordination for simulation is an object called the **ActivationGroup**. Initially, the single PC version of MACE3J used a static Configuration object that was referenced directly by every part of the system for global configuration values. However, early experiments soon showed this approach to be incompatible with remotely distributed objects that would not share the same program space for configuration. The distributed nature of the Grid system led us to rework how configuration values were accessed by having agents use messages to query the central **ActivationGroup** for configuration values. This in turn led to two types of messages: those for system coordination and those for inter-agent communication.

**Encapsulation:** Finally, programming support is needed for enforcing encapsulation of agents across all platforms types. This is perhaps the key issue for smooth transitioning of agent types across execution environments. With current widely-used object-based programming environments it is too easy to overlook pointers and object references that extend beyond the encapsulation boundaries of an agent. The instance of context for objects passed in messages, detailed above, is one such encapsulation failure, but the problem is a general one. In the progressive development model, proper encapsulation can be verified at the single-processor environment level so that transitions to the truly distributed environment of the Grid is smooth and seamless.

The limitations these types of problem place on the development of scalable agent systems is that many cross-platform issues have to be taken into consideration early in the experiment development process. The result is that transitions

---

<sup>2</sup> C.f., the fundamental issue of *distributed semantics* mentioned above and treated in [10].

to future environment platforms are more easily facilitated with "purer" encapsulation of agents and agent services throughout the system.

## 6 Conclusions

In conclusion, we find that smooth scalability for multi-agent simulations is not just a size problem—it is also a problem of system architecture style, managing degrees of distribution, and managing degrees of concurrency. The dream of seamless integration across multiple infrastructure scales and programming models is achievable, though current technologies to achieve it are only just emerging and need much more work to be robust. Leading-edge infrastructure like the Grid are still too hard to use, under-documented, and error-prone, but their advantages are becoming clear. The ability to migrate MAS simulations from simple but resource-poor environments to complex, realistic ones, maintaining progressive control over development parameters and building confidence in behavior, can be a valuable strategy for experimentation at large scales.

## References

1. Foster, I., Kesselman, C., eds.: *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, Menlo Park, CA (2003)
2. Berman, F., Fox, G., Hey, T.: *Grid Computing: Making the Global Infrastructure a Reality*. Wiley (2003)
3. Jennings, N.R.: On agent-based software engineering. *Artificial Intelligence* **117** (2000) 277–296
4. Agha, G.A.: *ACTORS: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA (1986)
5. Teragrid: <http://teragrid.org/> (2004)
6. Gasser, L., Kakugawa, K.: MACE3J: Fast flexible distributed simulation of large, large-grain multi-agent systems. In: *Proceedings of AAMAS*. (2002) 745–752
7. Globus: The globus toolkit 3 programmer's tutorial (2003) Version 0.2.2, <http://www.casa-sotomayor.net/gt3-tutorial/index.html>.
8. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maguire, T., Sandholm, T., Vanderbilt, P., Snelling, D.: *Open grid services infrastructure (OGSI) version 1.0*. Technical report, Global Grid Forum (2003) Global Grid Forum Draft Recommendation.
9. Bond, A.H., Gasser, L.: An analysis of problems and research in DAI. In Bond, A.H., Gasser, L., eds.: *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers Inc., Menlo Park, CA (1988) 3–35
10. Gasser, L.: Boundaries, identity and aggregation: Plurality issues in multi-agent systems. In Demazeau, Y., Werner, E., eds.: *Decentralized Artificial Intelligence III*. Elsevier (1992) 199–212
11. Hewitt, C.: The challenge of open systems. *Byte Magazine* **10** (1985) 223–242
12. Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., NagendraPrasad, M., Raja, A., Vincent, R., Xuan, P., Zhang, X.: Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems* **9** (2004) 87–143