Contents

List of Contributors

Matthias Scheutz
Department of Computer Science, Tufts University, Medford, Massachusetts, 02155, United States of America
e-mail: mscheutz@cs.tufts.edu

Jack J. Harris
Human-Robot Interaction Laboratory, Indiana University, Bloomington, Indiana, 47406, United States of America,
e-mail: jackharr@indiana.edu

Chapter 1
An Overview of the SimWorld Agent-based Grid Experimentation System

Matthias Scheutz and Jack J. Harris

1.1 Introduction

Computational modeling is becoming increasingly important, even in fields that have not traditionally used computational models (e.g., archeology or anthropology). Researchers in both the natural and the social sciences employ computer simulations to elucidate the time-course of physical and non-physical processes, or to explore the dynamics among different interacting entities, in an effort to discover new relationships that might lead to generalizable laws or to verify hypothesized principles as part of the empirical discovery loop (Peschl and Scheutz, 2001). However, there are two main obstacles to making effective use of today's (and likely also tomorrow's) computing environments. First, navigating the complexity associated with running large-scale computational simulations requires detailed knowledge about the available high performance computing environments. Such prerequisite knowledge includes: how to set up a simulation on the host computers (possibly including compilation on the target platform with installation of all the required libraries), how to schedule sets of simulations through the batch system, how to retrieve the resultant data, and how to troubleshoot if simulations do not finish (because they were terminated by the cluster's batch system for taking up more than the allocated CPU time, memory or storage allotment). The second obstacle is the management of increasingly large data sets that are the result of explorations of larger and larger model parameter spaces, both in terms of the dimensionality and sampling density of the space. This includes the preprocessing of data to facilitate statistical analysis and data mining, and the visualization of interesting relationships

Matthias Scheutz
Department of Computer Science, Tufts University, Medford, Massachusetts, 02155, United States of America, e-mail: mscheutz@cs.tufts.edu

Jack J. Harris
Human-Robot Interaction Laboratory, Indiana University, Bloomington, Indiana, 47406, United States of America, e-mail: jackharr@indiana.edu

among data sets. Either one of these obstacles is usually prohibitive for non-experts and will ultimately prevent modelers from using high performance computing resources to run large-scale simulations.

While there are certainly other challenges involved in making computational modeling more accessible to non-programmers (e.g., better modeling environments and tools for developing computational models in the first place), in this chapter we will focus on the above two challenges related to the high performance computing environment and the subsequent data analysis and visualization phase. Our goal is making the computational modeling process in high-performance computing environments as easy and intuitive as possible for modelers. This includes providing a computational framework that can automatically schedule, parallelize, distribute, and run simulations, using different strategies for the exploration of large parameter spaces. Moreover, it includes tools for automatically collecting data, organizing data in databases (that enables efficient data mining and statistical analyses), and visualizing data in effective, easily specifiable ways. Ultimately, we would like to have a framework that supports the *entire computational modeling process* (Peschl and Scheutz, 2001): from developing the first model, to testing and running it, to collecting, analyzing and visualizing data, to comparing data to empirical findings, revising the model, testing it, and so forth. Furthermore, this infrastructure should attempt to minimize model run times to speed up this process, for example, by automatically parallelizing the model (as we do not want to require modelers to be able to implement parallel code that can be executed on a cluster). Moreover, it would be desirable if the infrastructure could automatically handle vastly heterogeneous computing infrastructures (e.g., from dedicated homogeneous high performance computing environments, to heterogeneous ad hoc clusters with different operating systems). It would also be desirable for the infrastructure to dynamically adjust to changing computing environments since computational resources and resource availability can vary greatly over time and across research settings, and typically very specific knowledge is required to schedule and run processes in each environment.

To this end, we present "**S**im**W**orld **A**gent-based **G**rid **E**xperimentation **S**ystem", `SWAGES`, which has been under development for over a decade in our lab. `SWAGES` is used extensively for various kinds of agent-based modeling. In particular, `SWAGES` was co-developed with `SimWorld`[1] (Scheutz, 2001), an agent-based modeling environment built on top of the Birmingham `SimAgent` agent toolkit[2] (developed by Aaron Sloman). `SimWorld` is a generic simulation environment for spatial agent-based models that provides both interactive and batch mode execution and permits the definition of agent-based models in several programming languages. It has been used extensively for simulations of artificial life scenarios (e.g., Scheutz and Schermerhorn, 2008), evolutionary investigations (e.g., Scheutz and Schermerhorn, 2005), social simulations (e.g., Scheutz and Schermerhorn, 2004), swarms-based simulations (e.g., Scheutz et al., 2005) and individual-based biological models (e.g., Scheutz et al., 2010). It has also been used in education for teaching model

[1] `SimWorld` was the first simulation environment supported by `SWAGES`, hence the "SimWorld" prefix in `SWAGES`, even though it now works with many simulation environments.

[2] http://www.cs.bham.ac.uk/research/projects/poplog/packages/simagent.html

exploration and model development (e.g., Scheutz, 2008). SimWorld was the first simulation environment to support the automatic parallelization algorithms specified by and implemented in SWAGES (Scheutz and Schermerhorn, 2006). While the re-implementation of SimWorld in Java is still under development, several additional asynchronous scheduling algorithms have already been included. Evaluation of these asynchronous scheduling policies has demonstrated performance gains compared to the typical cycle-based scheduling policy used in the previous version of SimWorld and most other discrete-event simulators (Scheutz and Harris, 2010).

SWAGES can be used to explore large parameter spaces of various cognitive models, both connectionist models and classical cognitive architectures. On the connectionist side, our neural network simulator NNSIM has been used to explore parameter spaces of several neural networks, including neural networks for spatial attention (Scheutz and Gibson, 2006) and ideomotor compatibility (Boyer et al., 2009). On the cognitive side, a special purpose Lisp-based ACT-R "wrapper component" was developed to run the ACT-R cognitive architecture (Anderson et al., 2004) and pass various model parameters between the cognitive model and SWAGES, thereby enabling automatic explorations of large parameter spaces (e.g., using various ACT-R models in the "psycho-motor vigilance task" (Gluck et al., 2007)).

SWAGES started out as a set of shell scripts for running agent-based artificial life models on remote hosts in the late 1990s and has since evolved into a robust advanced modeling framework that meets all of the above requirements. Among its highlights are that it

1. runs on any platform that supports Java and can be automatically distributed over multiple hosts to guarantee high throughput for large numbers of parallel simulations;
2. can use a heterogeneous computing environment including any mixture of dedicated compute clusters or stand-alone hosts with no pre-installed software required on any host (only secure shell access is needed);
3. works with any simulation environment (including closed-source simulations) that can be minimally parameterized (e.g., through command line arguments or a special-purpose socket-based protocol);
4. can automatically parallelize simulations based on available computational resources for simulation environments that support parallelization, including synchronous and asynchronous scheduling algorithms, to maximize throughput using a dynamic pool of hosts;
5. provides a simple intuitive Web-based user interface for specifying, scheduling, and running large-scale model parameter spaces (including different supplied exploration algorithms as well as user-defined strategies that can automatically schedule additional simulations based on simulation outcomes);
6. enables automatic data retrieval and population of databases for efficient data mining;
7. facilitates automatic statistical analyses which includes both model fitting based on fitness criteria (error thresholds, types of models, etc.) and model discovery (based on constraints on model classes);

8. enables automatic visualization of different data sets obtained from large-scale simulations (and for linking in other visualization environments);
9. contains mechanisms for ensuring that simulations will eventually finish (despite crashes, interrupted simulations, or lack of available hosts), including checkpointing mechanisms if supported by the simulation model.

SWAGES will be described in detail in the following sections, starting with an overview of the system architecture. Next the new implementation of SWAGES in the distributed *Agent Development Environment* (ADE) (Scheutz, 2006) is discussed and some of the advantages of distributing its architecture are highlighted. Then describe an application of SWAGES in the context of a biological agent-based model to highlight how SWAGES addresses the challenges of exploring large model parameter spaces in high-performance computing environments. Finally, SWAGES is compared to other large-scale simulation frameworks and its new features are summarized.

1.2 System Architecture

The initial design goal for SWAGES is to simplify the modeling process by allowing modelers to define model parameter spaces they want to explore and submit them to SWAGES for execution. SWAGES then automatically schedules all simulations, runs them, collects the resultant data and transfers it to a specified place in the file system. It also performs simple statistical analyses and displays them through a Web-based user interface. Figure 1.1 shows an overview of the SWAGES architecture divided into "server-side" and "client-side" components as described by Scheutz et al. (2006) and Scheutz (2008).

The server-side components are responsible for scheduling, distributing, starting, and monitoring the execution of distributed simulation clients and possibly restarting failed clients to ensure that simulations will eventually finish. Specifically, users can submit experiments, check their status, perform simple statistics and view experiment results using any standard Web browser through a Web-based user interface provided by the *Web Server*. The *Web Server* forwards all submission requests to the *Experiment Server* which creates "experiment sets" based on user specifications that contain all necessary model parameters (e.g., the set of initial conditions across different experiments) as well as various SWAGES system parameters. System parameters can include priorities of experiments and scheduling parameters, levels of supervision and recovery parameters, formats of data collection and location for data storage, statistical analysis calculations to be used on the results and conversions of the output formats, and modes of user notification of simulation progress. The *Scheduler* is responsible for taking experiments from several priority-based queues (to which new experiments are submitted by the *Experiment Server*) and starting them on remote hosts. The experiment *Scheduler* will schedule new simulations when hosts become available and will only create experiment data structures for these simulations on demand (so as not to run out of memory

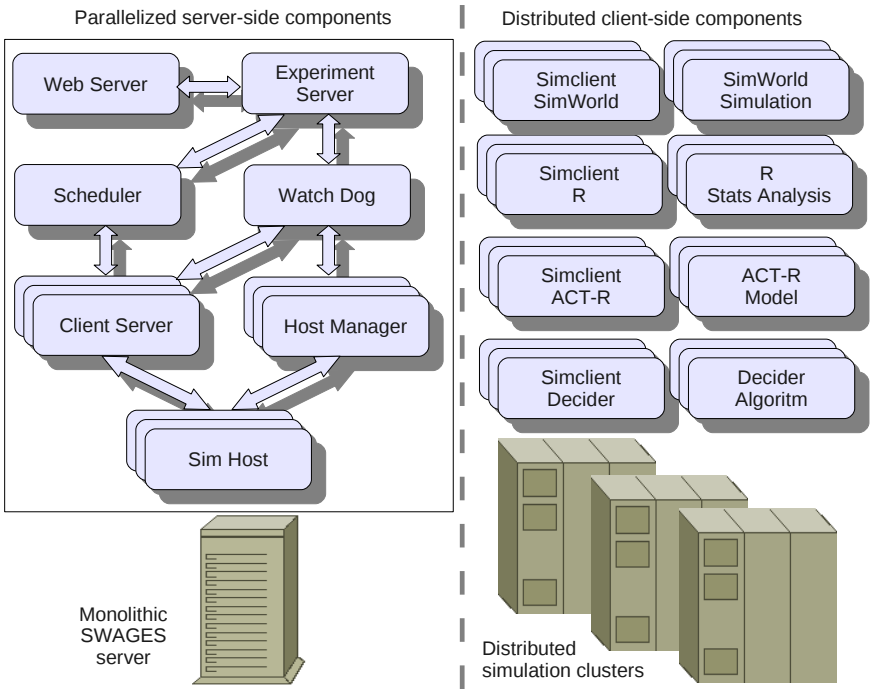

Fig. 1.1 Overview of the SWAGES architecture in 2009 (all server-side components within the solid square on the left had to run on the same host computer).

when processing large-scale experiment sets). The *Client Server* then manages a remote simulation and maintains an open communication channel with the simulation instance, keeping track of the simulation's progress, state, update, and degree of parallelization. This ongoing monitoring is critical for error detection and recovery so the client server can restart or resume the simulation elsewhere based on its saved state, if available, when a simulation crashes (e.g., due to OS problems on its host), is not responding (e.g., due to network problems), or cannot be continued (e.g., because its current host does not meet user-defined criteria for running simulations anymore). The *Watch Dog* implements a second level of supervision which is particularly important for dynamic computing environments where hosts can "disappear" from the pool of usable machines without notification. It regularly checks all simulation clients for progress, terminates clients that are stuck or not responsive, and reschedules simulations either from scratch or from saved states. The server-side representation of remote simulation hosts is achieved by the *SimHost* component, which keeps track of any simulations running on the host and is also responsible for monitoring the host's availability based on user-defined criteria (e.g., the remaining CPU time on a cluster host or whether a console user is logged on). The *Host Manager* keeps track of all available simulation hosts by managing a dynamic pool

of available resources. It can automatically request new hosts in high-performance computing environments by submitting requests on demand through the cluster's batch queuing system.

On the client side, special *SimClient*s representing particular simulation environments communicate updates about the simulations to the (server-side) *Client Server*s. *SimClient*s are responsible for saving the state of simulations (if supported by the simulation) and routinely checking that the simulation is still allowed to run on its current host according to user-defined criteria. Generic *SimClient*s are available to interface with simulations written in various programming languages (Java, Pop11, Scheme, Lisp, and R) and customized *SimClient*s are available for `SimWorld` (Scheutz, 2001) and ACT-R (Anderson et al., 2004). Furthermore, `SWAGES` also supports various special purpose reusable clients (e.g., a generic *gradient search client*[3]).

Recently, however, it became clear that to truly scale up to the requirements of tomorrow's modelers, several additional steps in the design and implementation of `SWAGES` needed to be taken. First, the server-side part of `SWAGES`, while parallelized, was monolithic (i.e., all server-side components had to run within the same Java virtual machine on the same host). And while multi-core CPUs are partly able to alleviate the *processing bottleneck*, they cannot help the *networking bottleneck* created by a monolithic grid engine that communicates with thousands (if not tens of thousands) of simulations simultaneously. The solution, therefore, is to distribute and duplicate (some of) the parallelized server-side components over multiple hosts. Second, for `SWAGES` to effectively handle large data sets (of hundreds of gigabytes and beyond), simply storing data as text files in the file system is not practical as data queries searching sequentially through these large files would take too much time. Rather, a database interface is required that allows automatic commits of data (as results are produced by *SimClient*s) into a database (which potentially can be distributed itself and facilitate efficient data querying and data mining). Third, to aid the modeler in the exploration of large parameter spaces, automatic statistical analyses on the returned intermediate data sets are required in order to determine whether a particular region of the parameter spaces should be further explored. Hence, additional mechanisms for automatic data analysis, model fitting and model discovery were developed and integrated. Fourth, the previously integrated simple visualization mechanisms were too limited to do justice to the complexities of large data sets. Therefore, a new visualizer component has been added to provide better, more effective automatic generation of data visualizations based on model space parameters and experimental results. Finally, to reduce the complexities of data exchanges between different components (from the simulation environment, to the database, statistical analysis and visualization) and to enable the easy integration of external components (e.g., simulation environments, statistical analysis tools and visu-

[3] This client is capable of wrapping a generic simulation process and supports batching a group of parameters to run on the client. This client has a two fold benefit: (1) it minimizes the amount of independent server connections, which can incur an overhead for very fast simulations and (2) aids distributed Monte Carlo search experiments.

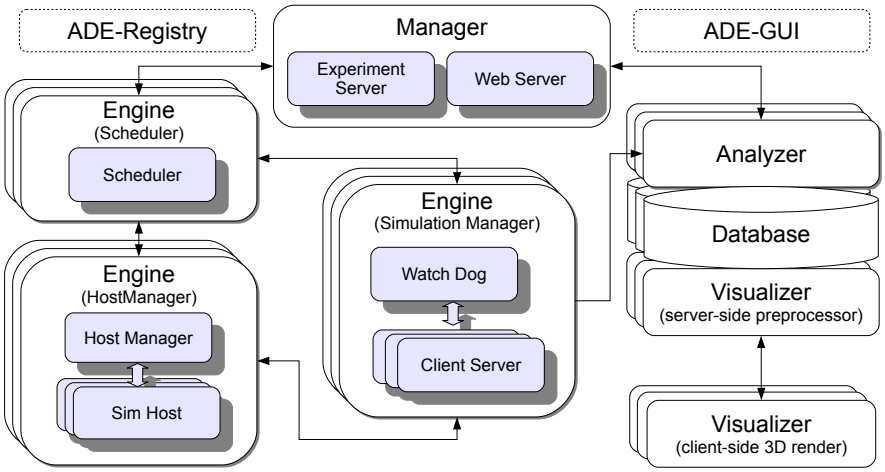

Fig. 1.2 Overview of the server-side components of the SWAGES architecture as currently implemented in ADE (large rounded boxes depict new ADE components, small grey rounded boxes depict old SWAGES components; the two dashed rounded boxes show ADE infrastructure components). ADE components can be freely distributed to any host, thus enabling the distribution of SWAGES server-side components. Except for the *Manager*, multiple copies of each component are now possible within a SWAGES instance, thus allowing SWAGES to scale with the number of simulations it is serving (see text for detail).

alizers), a new open, experiment definition and data exchange format based on the Extensible Markup Language (XML) was specified and implemented.

To address the required modification of SWAGES (for scaling up to large numbers of simultaneous simulations), all SWAGES components were re-implemented in the distributed ADE (e.g., Scheutz, 2006). ADE provides many features that are of critical importance for the distributed version of the SWAGES server-side components: automatic load balancing and host management, component supervision, error detection and restart, and various other mechanisms for *autonomic computing*. Moreover, ADE's tight security services provide fine-grained, method-level authentication, which is integtral in an open distributed computing environment. ADE also provides distributed graphical interfaces that allow location-independent system configuration and monitoring. To leverage these features, SWAGES components were converted into ADE components and new features related to large-scale computational simulation experimentation were added. At a high level, SWAGES is now composed of the following components: the *Manager*, the *Engine*, the *Analyzer* and the *Visualizer* (the extended server-side architecture of SWAGES as implemented in ADE is shown in Figure 1.2).

The *Manager* is responsible for receiving experiment specifications and potentially ending a simulation based on specified termination criteria. It initializes the *Engine* and the *Analyzer* together so they can communicate directly with each other in order to process simulation results.

The *Engine* receives the specification of a simulation, including simulation execution details and the parameter space for evaluation from the *Manager*, and is responsible for distributing the simulation environment across a collection of networked nodes and consolidating the results. The *Engine* is considered a reliable service and will ensure that results are returned from a simulation by handling events such as check-pointing and restarting of simulations.

As results are collected by the *Engine*, they are forwarded to the *Analyzer*. The *Analyzer* first stores the data in an indexed database to aid data analysis and future interactive exploration of the results. The *Analyzer* then performs a battery of statistical analyses on the data. Topographical function forms (which can potentially explain the relationship of parameters in an experiment) are produced for the specific parameter space and are evaluated in real-time using distributed statistical processing nodes as the data returns from the simulations. The *Analyzer* also provides a conduit for carrying out post-hoc data evaluation by data mining over the processed results (e.g., by supplying an interface for querying the simulation results closest to a particular parameter combination). This type of post-hoc data mining of a parameter space lets modelers identify optimal models during model validation and evaluation.

Complementing the *Analyzer*, the *Visualizer* can provide a way for quickly identifying interesting parameter relationships, understanding the breadth of a simulation's performance or even identifying problematic performance of a simulation through interactive three dimensional renderings of the processed data. The *Visualizer* can plot individual data points, line plots, lego plots and even surface plots in order to help explain the properties of the performance space.

1.3 System Implementation

The ADE middleware for designing distributed autonomous agent systems was chosen as the implementation platform for the extended version of SWAGES for several reasons. First, a number of other higher-level architectures have already been successfully developed within the ADE framework including DIARC, an embodied agent framework for robot software development (Scheutz et al., 2007) and ADE-Unreal, a virtual world simulator. Thus, ADE provides a suitable infrastructure that allows functionality associated with SWAGES to be distributed across a set of computers. ADE is also implemented in Java which aids easy code integration of existing SWAGES components. Furthermore, this distributed infrastructure permits the introduction of new processor and storage intensive functionality associated with large-scale computational experimentation.

To facilitate the integration of the separate functional components, each existing and new SWAGES service is implemented or "wrapped" as an ADE component. This is possible because ADE provides a class hierarchy for extending ADE services, which gives new services the ability to intercommunicate with one another as well as utilize core ADE features (such as server security, a server recovery

mechanism, service logging, and monitoring). Access to ADE services are readily accessible using this wrapping mechanism, however non-Java based services can also intercommunicate with ADE servers using Web services in the form of XML-RPC (http://xmlrpc.com/spec) or Representation State Transfer (Fielding and Taylor, 2002). New ADE services extending *ADEWebServicesServer* permit the exposure of a subset of remote methods based on user credentials and remote IP.

ADE enables secure message transport to SWAGES components via a remote method calling feature which utilizes Java's Remote Method Invocation, a set of application programming interfaces that allow developers to build distributed applications. ADE also provides a meta-level naming scheme to allow an abstraction over these interfaces capable of service migration. The remote method calling feature of ADE is protected using a centralized authentication and access system. This is the predominant mechanism by which ADE services communicate.

ADE also provides a method for creating distributed graphical user interfaces (GUIs) using ADE−GUI, an extensible framework that abstracts away many of the issues of distributed GUI development. This abstraction and encapsulation of functionality thereby aids the easy creation and integration of new GUI components. Many components within the system implement GUIs that can be viewed through ADE−GUI components, however some servers provide an additional Web-based user interface for control.

1.3.1 Key Components

Messaging - The SWAGES system communicates using our new Simulation Specification Markup Language (SSML). This simple XML specification allows all components to utilize the same data structures for defining simulation experiments. Such a shared representation is useful because many components need access to the same type of information (e.g., the *Analyzer* and the *Visualizer* both need access to schemas describing the results data). There are primarily three types of top-level SSML structures: Execution Details, Parameter Space Definition, and the Results Definition Schema. Using these basic structue, SWAGES components can communicate with one another (using the standard message passing mechanisms defined within ADE).

Manager - Centralized access to the collection of SWAGES components is provided through the *Manager*. The *Manager* is responsible for receiving all of the components' specifications used to process, analyze and evaluate the experiments. This role is similar to that of the *Experiment Server* in the previous version of SWAGES. The *Manager* receives the three basic SSML structures from a user and passes them along to other SWAGES components. The *Manager* also relays to the *Engine* simulation execution details along with the parameter space information which together define the scope of the simulations to evaluate. User interfaces to the *Manager* are provided via the command line and other external add-on ADE components in the

ADE network. The *Manager* also exposes programming methods using Web services to make the direct submission from external user appliations possible.

Engine - The *Engine* is essentially a distributed form of the server-side SWAGES components with several extensions to let it communicate externally with other ADE services. The *Engine* is comprised of three distinct ADE components that allow server-side components of SWAGES to run on multiple hosts. These three new components of the *Engine* intracommunicate in-order to carry out the task of distributing and managing the execution of simulations.

1.3.2 Novel Features in SWAGES

We now briefly summarize the advantages of distributing the SWAGES architecture and its extended features that allows it to interoperate with the other components.

1.3.2.1 Distributed SWAGES

The distribution of the subcomponents of the *Engine* across a set of hosts provides many performance advantages. As the number of executing *SimClient*s grows, so, too, does the amount of communication back to the SWAGES server. Large amounts of concurrent communication can potentially render a *Client Server* as a bottleneck for a parallel simulation run. Processing these messages can be both a processor intensive operation as well as a bandwidth bottleneck. Distributing the management in a hierarchical manner produces a completely scalable system (e.g., a new *Client Server* can be started on a new host as soon as existing ones reach their capacity limits based on experiment submissions and available simulation hosts).

1.3.2.2 Post Processing

The system provides additional components to process the results returned by the remote simulations. These post-processing tools provide services including data warehousing, statistical analysis and visualization features. Some of the services rely on system-unique resources such as database installation or even graphic card requirements. However, as is the case for all ADE components, these services do not need to run on the same host as the *Engine* or the *Manager* but can run on any host in the ADE environment that meet the requirement of the service.

1.3.2.3 Analyzer/Database

It is not uncommon for a large-scale distributed simulation experiment to evaluate billions of parameter combinations and produce gigabytes (if not terabytes) of resulting data. As Charlot et al. point out, "these data need to be managed, shared and analysed using varied computational methodologies, such as data mining and database management systems" (2007). SWAGES can perform such tasks through the *Analyzer*, which provides the storage and efficient management capabilities to facilitate automated results analysis as well as real-time interactive data mining.

1.3.2.4 Database

Loading all simulation results into main memory for analysis quickly becomes infeasible with large-scale simulations producing terabytes of data; even searching for a desired result in a massive data set (e.g., over one billion simulation results) can be extremely time-consuming if standard sequential file access is used to scan a file system. The *Analyzer*, therefore, utilizes a MySQL[4] database to house the massive number of results produced by SWAGES. The tables are indexed using "B+ trees" (Comer, 1979) which minimize the number of filesystem accesses required to search for a result while also maximizing the usage of the main memory of the system. Users can manage and manually analyze a stored data set by accessing it via a Web-based user interface.

1.3.2.5 Analysis Processes

With a robust data storage and retrieval backend in place, the *Analyzer* can provide automated analysis of large amounts of data. The processes for doing this analysis involve creating and populating a model repository (database), learning topographical functions that best fit the data (automated analysis), executing predictions, and reverse lookups from the data store (interactive analysis).

The process begins when information about the parameter space is received from the *Manager* and a model repository is created to house the data. A model repository is defined in terms of the variables of the parameter space for the experiment. Using additional information provided by the *Manager*, the *Analyzer* then subscribes to the *Engine* and requests all results produced for a given experiment. The received data is parsed and stored into the respective model repository for that experiment. The *Analyzer* then begins to produce a set of potential "hypotheses" about functional relationship between the parameters of the space.

As results are uploaded from the remote simulations, automated analysis of the data begins. The goal of the analysis is to find a function that best describes the topographical form of the data based on the set of function schemas originally generated

[4] http://mysql.com

in the previous process. Multivariate regression analysis, which uses the database directly, is used to evaluate each candidate function. Earlier versions of the *Analyzer* utilized multiple subprocesses running R[5] and MATLAB[6] for statistical analysis. Though effective, memory constraints for very large data sets produced run times an order of magnitude slower than the database statistical package written for the current *Analyzer*. Thus, relying on the database directly versus using external statistics packages has proven to be an efficient way of doing automatic data analysis for large data sets.

As new results flow into the system and the data set grows, the candidate topographical function forms are continuously re-evaluated based on goodness of fit to the current data set. The selection of which functions to evaluate is determined by a greedy algorithm which identifies those function forms that best fit the data set prior to the addition of the new results. The result of this process is that at any moment the *Analyzer* has a set of functions that describe the relationship of the parameters for the experiment.

1.3.2.6 Visualizer

The *Visualizer* provides an interactive three dimensional graphical interface, using the FreeHEP Java3D[7] library extensions (produced by the high-energy physics modeling community), for plotting model parameters against one another to aid the evaluation process. However, visualization, like data analysis, is complicated by the problem of handling vast amounts of data, therefore, the *Visualizer* provides preprocessing mechanisms for data to minimize the amount of data that needs to be sent to the user's three dimensional rendering system. Hence, the *Visualizer* consists essentially of two components: the client-side rendering application and the database plus the preprocessing server-side scripts. This architecture makes the visualization of massive data sets possible and efficient by exploiting the speed of the highly indexed database representation and minimizing the data that needs to be transferred to the client.

1.4 A Case Study in Employing `SWAGES`

In this section we will present an interdisciplinary research project in biology and computer science that demonstrates the use and application of `SWAGES`. We first summarize the research questions addressed in the particular project and briefly introduce the agent-based simulation environment used to explore these questions. Then we discuss the role `SWAGES` played in conducting the simulation experiments

[5] http://cran.r-project.org

[6] http://www.mathworks.com/products/matlab/

[7] http://java.freehep.org

and the subsequent data analysis, showing how large-scale simulation experiments can be defined and executed in a grid environment, and ultimately how the results returned from the parallel simulations can be stored, analyzed, and visualized to provide feedback to the modeler.

1.4.1 Research Questions and Simulation Model

The main research question we explored in the project comes from mate choice in biology: how do females pick their mates? Specifically, how do female treefrogs decide which male treefrog to approach in the swamp at night when the only information about males available to females are the acoustic properties of the males' mating calls? Prior work in theoretical and ecological biology proposed two main competing strategies for females: "pick the closest out of N males" or "pick the closest above some threshold θ". To test the success of and tradeoffs among these two strategies, we developed a social agent-based model that explicitly models male and female frogs as "agents" located in a spatially extended, two dimensional environment. The model included both environmental parameters (e.g., the number and distributions of male and female frogs) and individual parameters (e.g., the call quality of males or the mating strategy employed by females). The goal was to systematically vary environmental and individual parameters to gain an understanding of the various dependencies and tradeoffs among the different dimensions (e.g., how mating success depended on the distribution of males, how time-to-mating depended on the number of competing females, how male-female ratio influenced the performance of the different strategies, etc.). Details about the biological questions and the computational model, have been reported by Scheutz et al. (2010).

1.4.2 The Simulation Environment

The social female choice model was implemented as a discrete-event agent-based simulation in Java. After initial testing of the simulation in a graphical environment to verify that all agents individually behaved as expected, the simulation environment was connected to `SWAGES` to permit the automatic scheduling and running of the millions of simulations required for the parameter space we had set out to explore. The integration of the simulation with `SWAGES` was accomplished by extending a `SWAGES` client-side component, the *JAVAClient*, and overriding some of its methods. The primary method that had to be overridden was the entry-procedure which receives a list of startup parameters used by the simulation to govern its operations (other methods can also be overridden if advanced features such as check pointing or `SWAGES`-level control over the event-scheduler algorithm are desired). The main functionalities provided by the *JAVAClient* are the systematic representation of agent types and the parallelizable, discrete-event scheduler (for details about

an integration that takes advantage of the advanced intra-simulation discrete event scheduling algorithms, see Scheutz and Harris (2010)). These pre-defined algorithms allow for quick model development and verification. Moreover, the social female choice simulation environment used mechanisms provided by the *JAVAClient* for handling initial conditions and specifying event details (e.g., mating events) for data recording during simulation runs. All data is recorded in systematically named files which can then be collected by SWAGES for subsequent data analysis.

1.4.3 Simulations Runs in SWAGES

The data presented in this case study was obtained by operating SWAGES within a standalone HPC configuration where both the server and processing nodes all operated on the BigRed supercomputer.[8] Other phases of the research project used different HPC configurations (e.g., a heterogeneous set of processing nodes consisting of local laboratory machines and compute nodes on a shared resource grid). Regardless of the particular composition of the computing environment, the simulation environment along with resource files defining the parameter space, the simulation's execution details and the resulting output files' data format, need to be submitted to the SWAGES server. SWAGES then initializes simulations on compute nodes as they become available, passing startup parameters to simulation instances and collecting data from simulation output files for subsequent insertion into the SWAGES database.

For the particular experiment set performed on BigRed, three different mating strategies were evaluated: a *minimum threshold* strategy (where the closest mate exceeding some fitness criteria is selected), a *best of N* strategy (where the mate with the highest fitness metric is chosen out of the N closest candidate mates), and a random mating strategy (used as a baseline). Each strategy was tested across the same set of initial starting conditions which varied different environment factors and the characteristics of the frog population. For example, for the random condition to be evaluated across the range of initial conditions, 240 000 simulations were run. More simulations were run for the *minimum threshold* and *best of N* because both strategies included additional strategy parameters (fitness threshold level and size of N, respectively). For the *best of N* strategy, five different values of N were evaluated. This resulted in 1 200 000 simulation runs ($5 \cdot 240000$). For the *minimum threshold* strategy, ten different values for the fitness threshold were tested, resulting in 2 400 000 additional simulation runs ($10 \cdot 240000$).

In order to execute such a large number of simulations quickly, a large number of compute nodes were required. However, at the time of submission the queue on the cluster environment was long for simulations with a large number of compute node requests. Rather than attempting to reserve a large number of nodes which would have caused the requests to sit in the queue for weeks, we decided to let SWAGES

[8] For details on BigRed system, see http://kb.iu.edu/aueo.html.

request compute nodes dynamically and individually (up to a maximum number of 50 node requests at a time each for four hour increments). When nodes became available, SWAGES then scheduled and executed simulations until that node's alloted time expired.

1.4.4 Data Management and Visualization

As a result of employing this dynamic resource request strategy, the entire set of 3 840 000 simulations finished in a little over two weeks from the time they were submitted (roughly 17 days). The simulations generated over 11 gigabytes of summary data (with precomputed statistics) and over 10 terabytes of detailed results in over 35 billion files. Clearly, without tools like SWAGES, the number of simulations and the size of the resultant data would be prohibitive for model exploration of this magnitude. For example, for large datasets like the ones produced in this study, standard file systems are not a practical method for data management when searching for patterns inside the data files requires a slow scan of every file. This is where the automatic insertions of results in a database has proven to be of great use for quickly querying the dataset. Earlier versions of SWAGES did not utilize a database. Instead, custom scripts were generated to search the returned results files for patterns of interest. To appreciate the potential speedup provided by the automatic database creation and usage, consider the time it would take to search through every results file to find the set of parameters that produced the maximum number of mated tree frogs: the process of using results files involved searching each file, which required N file accesses where N is the total number of parameter combinations explored! In the case where the data is stored in an indexed database, the desired value can quickly be found without the need to examine every simulation result.

Once the data is pre-processed, visualizations can be produced. In previous versions of SWAGES, these visualizations had to be generated manually; in the current version, visualizations can also be produced automatically. Figure 1.3 shows examples of the simulation results from the frog study that visualize different effects of mating strategies. These types of graphs are immediately meaningful for the modeler and have been used both for verifying theoretical predications as well as developing new refined computational models.

1.5 Discussion

Computational frameworks and infrastructures that are intended to support large-scale explorations of model parameter spaces face significant challenges as the numbers of usable CPUs and hosts in clusters and grid environments rapidly grow and data sets increase in size by several orders of magnitude. The first question that arises is how an infrastructure will "scale up" in light of this enormous growth in

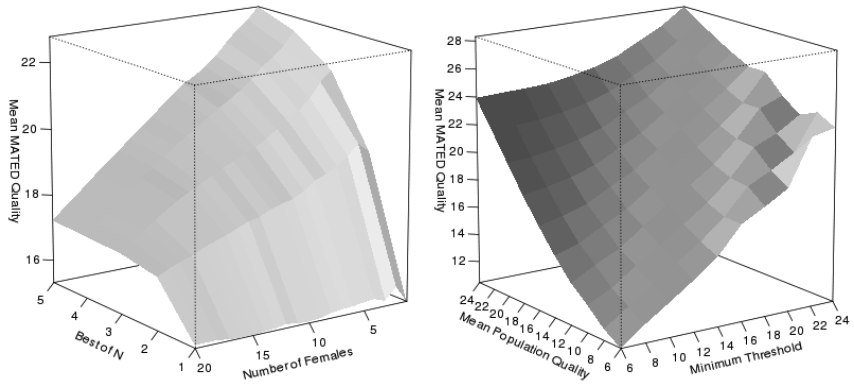

Fig. 1.3 Sample graphs of results generated based on a systematic exploration of the parameter space for two strategies. The left graph shows the dependence of the quality of the mated male frogs on the number of females in the simulation and different parameters for the *best of N* female mating strategy. The right graph shows the dependence of the quality of the mated male frogs on their average call qualities and the value for the *minimum threshold* in the female mating strategy.

both resources and resource demands. A second related question is how the large amounts of data should be managed (i.e., stored, accessed, analyzed, mined, visualized, etc.). And third, as infrastructures become larger and more complex, a technical question arises as to whether and to what extent these infrastructures are able to autonomously handle various types of errors (component crashes, network problems, etc.) that are inevitable in complex computing systems. The goal of the re-implementation of SWAGES in the ADE middleware is to answer all three questions and address the associated challenges.

1.5.1 Automatic parallelization of agent-based models

One feature that makes SWAGES unique among experimentation frameworks is its ability to automatically parallelize and distribute agent-based models. This feature is particularly important for modelers who would like to quickly run a particular model, visualize the results, possibly change a few parameters and run it again. This type of *interactive model exploration* is usually only possible with models that run over a short period of time on a single computer, but not with models that have run-times of tens of minutes or hours (when minutes might already be too long). Since there is a natural limit to how fast a simulation can run on a single computer, the only way to allow quasi-real-time interactions is to parallelize the model. However, parallelization usually requires advanced programming skills *and* support by the modeling environment (neither of which is usually available). SWAGES addresses this problem by removing the burden of parallelization from the modeler.

Specifically, the need for a modeler to manually execute model simulation runs is eliminated by SWAGES with its ability to automatically utilize all computational resources available for parallelization. This is possible for simulation environments that implement an "event horizon", a particular notion of influence a simulated entity can have on other simulated entities within the model simulation (Scheutz and Schermerhorn, 2006; Steinman, 1994). While the event horizon is currently only implemented for (metric) spatial agent-based models, it can be generalized to non-spatial models. For example, it could be used for non-spatial models that are based on interaction graphs that connect every entity in a simulation to those entities with which it can interact.

In addition to parallelization SWAGES also provides mechanisms to support *asynchronous updating and scheduling within the simulation environment*. Parellelization allows simulation instances to run asynchronously on different hosts until one simulation instance needs data from an entity that is updated in another simulation instance. In asynchronous scheduling, simulation entities are not updated cycle by cycle, but scheduled (to the extent possible) based on which entity in one simulation instance will be required to provide information needed by another simulation instance in the future. This anticipatory way of updating entities within each simulation instance in the distributed simulation system can lead to significant overall run-time speedups (Scheutz and Harris, 2010).

In general, the automatic parallelization of simulation models enables fast "model discovery loops" (especially with complex models) while alleviating the modeler from having to parallelize model code manually. Furthermore, parallelized models also facilitate distributed real-time simulations (e.g., of mixed live-virtual-constructive environments) that would otherwise not be able to run in real-time.

1.5.2 Integrated data management

A large amount of data can be produced by even simple agent-based models. For example, models created by Scheutz et al. produced hundreds of gigabytes even though each model run finished within only a fraction of a second (2010). In such cases it is not possible to store simulation data distributed over millions of files in the regular file system (e.g., we have seen simulations produce more result files than a directory could hold given the OS file system restrictions). Rather, data needs to be automatically inserted in a database in a way that aids efficient data access. SWAGES automatically creates a database based on the initial experiment setup and inserts simulation results in an organized way into tables that can subsequently be combined using Structured Query Language (SQL) queries. Moreover, since the database can effeciently access large datasets it is used to preprocess the data for real-time interactive visualizations. The immediate availability of visualizations (if only on sparse data) lets modelers anticipate results and possibly correct simulation setups early on (e.g., if the explored region of the space is not interesting, or if the

results are counter-intuitive) rather than wasting many expensive compute cycles on completing the entire simulation run.

1.5.3 Automatic error detection and recovery

SWAGES comprises several levels of error detection and recovery. At the infrastructure level, ADE provides component supervision, component monitoring, and component restarting in case components fail or crash (for core SWAGES components as well as custom simulation components). Moreover, at the SWAGES level, simulation monitoring, supervision, check-pointing and restarts guarantee that (eventually) scheduled simulations will finish. This includes mechanisms for SWAGES itself to become "dormant" if no host is available (i.e., the *Scheduler* will save its state to disk and schedule a special shell script for execution that will be able to resume the system). At the simulation client level, error recovery comprises restarting of all parallelized clients if any errors prevent a parallelized simulation from finishing. Finally, advanced notification mechanisms are available at each level to inform operators or users about the state of the system at any given time (e.g., through the ADE-GUI or through a Web-based user interface).

1.5.4 SWAGES *compared to other frameworks*

SWAGES shares several features with other grid middleware systems such as Berkley Open Infrastructure for Network Computing (BOINC), Condor, or QosCosGrid. For example, BOINC (Anderson, 2004) also supports the distribution of parameters and input files from a centralized server to client applications running on remote hosts. Architecturally, there is a similarity between the BOINC manager application and a SWAGES *SimClient* in that they both broker communication back to the server and execute the client-side simulation. Furthermore, the corresponding BOINC server-side feeder application responsible for providing initialization information to the client simulation can be likened to the role provided by the SWAGES server-side *Scheduler* and *Simulation Manager*. In SWAGES, the *Simulation Manager* fills the additional function of overseeing all communication with its associated *SimClient* whereas in BOINC there are a series of independent services responsible for these interactions (i.e., the respective feeders, validators, and assimilators). Also, similar to other batch submission systems, SWAGES handles distribution of work across a series of available nodes, and like Condor (Bent, 2005) and BOINC, SWAGES is sensitive to the usage availability of the processing host. SWAGES also shares features with QosCosGrid ProActive. Both are Java-based grid middleware systems and support the parallel distribution of simulation (Kravtsov et al., 2008). Additionally, the ability of QosCosGrid to utilize systems crossing security domains (Choppy et al., 2009) is shared by SWAGES. SWAGES can automatically establish a secure shell

tunnel for systems behind firewalls and permits the use of either shared or system-unique access credentials when establishing these connections. Furthermore, both systems support an XML markup structure for job submission but differ in their implementation: QosCosGrid's QCG Job Profile supports a broader range of simulation execution types while SWAGES is a targeted system for large-scale experimentation and therefore has specialized features to support its workflow.

However, unlike most other popular grid systems, SWAGES supports some features not commonly found elsewhere. SWAGES has been tailored to natively support experimentation through integrated parameter sweep mechanisms. This feature coupled with a Web-based submission capability make SWAGES very easy for a modeler to use. Furthermore, SWAGES supports the unique capability of being able to dynamically parallelize a multi-agent simulation at the client level across a series of hosts. While some extensions to existing simulations add support for distributed computing (e.g., HLA_RePast (Minson and Theodoropoulos, 2004), which uses HLA to distribute simulations based on the *RePast* toolkit (Collier, 2001)), the distribution is not automatic and does not provide advanced distributed discrete-event scheduling that is found in SWAGES. Furthermore, SWAGES modelers do not have to include any provisions for parallelization in their code. Simply adding the keyword "parallelize" to the experimental setup definition is sufficient for SWAGES to attempt parallelization of simulations whenever possible based on the available computational resources.

SWAGES's fine grain parallelization and asynchronous scheduling can lead to a much better use of a large array of computational resources when individual simulations are extremely computationally intensive.

1.6 Conclusions

In this chapter we provided an overview of the latest version of SWAGES. Many attributes of SWAGES make it an easy system to install and use from a modeler's perspective:

- no pre-installed components are required (only secure shell access),
- no fixed pool of compute nodes is required (SWAGES dynamically adjusts to the pool of available hosts in the grid environment),
- simulation supervision and recovery mechanisms are included, and
- simulation experiments can be easily defined through a Web-based user interface, which also allows the modeler to look at results and data visualizations as data becomes available.

And since SWAGES is now implemented in ADE, it provides an open interface that facilitates easy interconnections with other platforms and components, and thus enables easy extensions of SWAGES functionality. In particular, it inherits the flexibility of ADE to use any other component implemented in ADE. For example, existing natural language components (developed in the context of a distributed

robotic architecture DIARC) could be used for user interactions, or action script interpreters for controlling physical agents could be used as scripting engines. Moreover, SWAGES can be easily extended by adding new ADE components based on the standard interface mechanisms provided by ADE (even closed-source commercial off-the-shelf software could be "wrapped" as ADE components and integrated into the system). Finally, the new distributed architecture allows SWAGES to manage large-scale grid simulations that are not possible for a system with a monolithic single server configuration.

References

D. P. Anderson. BOINC: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE Computer Society, 2004.

J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. In *Psychological Review 111*, pages 1036–1060, 2004.

J. Bent. *Data-driven batch scheduling*. PhD thesis, University of Wisconsin, Madison, May 2005.

T. Boyer, M. Scheutz, and B. Bertenthal. Dissociating ideomotor and spatial compatibility. In *Proceedings of the 31st Annual Conference of Cognitive Science*, 2009.

M. Charlot, G. de Fabritis, A. L. Lomana, A. Gómez-Garrido, D. Groen, L. Guylas, A. Hoekstra, M. Johnston, G. Kampis, S. P. Zwart, S. Robinson, M. Strathern, M. Swain, G. Szemes, and J. Villà-Freixa. The QosCosGrid project: Quasi-opportunistic supercomputing for complex systems simulations. Description of a general framework from different types of applications. In *Proceedings of IBERGRID 2007*, Santiago de Compostela, Spain, 2007.

C. Choppy, O. Bertrand, and P. Carle. Coloured Petri nets for chronicle recognition. In F. Kordon and Yvon Kermarrec, editors, *Ada-Europe '09 Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies*, pages 266–281. Springer-Verlag Berlin / Heidelberg, 2009.

N. Collier. RePast: An extensible framework for agent simulation. *Natural Resources and Environmental Issues*, 8:17–21, January 2001.

D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, June 1979.

R. T. Fielding and R. N. Taylor. Principled design of modern Web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, May 2002.

K. A. Gluck, M. Scheutz, G. Gunzelmann, J. Harris, and J. Kershner. Combinatorics meets processing power: Large-scale computational resources for BRIMS. In *Proceedings of the Sixteenth Conference on Behavior Representation in Modeling and Simulation*, pages 73–83, 2007.

V. Kravtsov, A. Schuster, D. Carmeli, K. Kurowski, and W. Dubitzky. Grid-enabling complex system applications with QosCosGrid: An architectural perspective. In

Proceedings of The International Conference on Grid Computing and Applications (GCA'08), pages 168–174, 2008.

R. Minson and G. Theodoropoulos. Distributing RePast agent-based simulations with HLA. In *Proceedings of the 2004 European Simulation Interoperability Workshop*, Edinburgh, UK, 2004.

M. Peschl and M. Scheutz. Explicating the epistemological role of simulation in the development of theories of cognition. In *Proceedings of the Seventh International Colloquium on Cognitive Science*, pages 274–280. Institute for Logic, Cognition, Language and Information, The University of the Basque Country, 2001.

M. Scheutz. The evolution of simple affective states in multi-agent environments. In D. Cañamero, editor, *Proceedings of AAAI Fall Symposium*, pages 123–128, Falmouth, MA, 2001. AAAI Press.

M. Scheutz. ADE - steps towards a distributed development and runtime environment for complex robotic agent architectures. *Applied Artificial Intelligence*, 20 (4-5):275–304, 2006.

M. Scheutz. Model-based approaches to learning: Using systems models and simulations to improve understanding and problem solving in complex domains. In P. Blumschein, W. Hung, and D. Jonassen, editors, *Artificial Life Simulations–Discovering Agent-Based Models*, volume 4 of *Modeling and simulations for learning and instruction, artificial life simulations–discovering agent-based models*. Sense Publisher, Rotterdam, 2008.

M. Scheutz and B. Gibson. Visual attention and the semantics of space: Evidence for two forms of symbolic control. In *Proceedings of the 28th Annual Meeting Cognitive Science Society*, Vancouver, British Columbia, Canada, 2006.

M. Scheutz and J. Harris. Adaptive scheduling algorithms for the dynamic distribution and parallel execution of spatial agent-based models. In F. Fernández de Vega and E. Cantú-Paz, editors, *Parallel and Distributed Computational Intelligence*, volume 269 of *Studies in Computational Intelligence*, pages 207–233. Springer, 2010.

M. Scheutz and P. Schermerhorn. The role of signaling action tendencies in conflict resolution. *Journal of Artificial Societies and Social Simulation*, 7(1), 2004.

M. Scheutz and P. Schermerhorn. Predicting population dynamics and evolutionary trajectories based on performance evaluations in alife simulations. In *Proceedings of 2005 Genetic and Evolutionary Computation Conference*, pages 35–42. ACM Press, June 2005.

M. Scheutz and P. Schermerhorn. Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. *Journal of Parallel and Distributed Computing*, 66(8):1037–1051, 2006.

M. Scheutz and P. Schermerhorn. The limited utility of communication in simple organisms. In *Artifical Life XI: Proceedings of the Eleventh International Conference*, pages 521–528, 2008.

M. Scheutz, P. Schermerhorn, and P. Bauer. The utility of heterogeneous swarms of simple UAVs with limited sensory capacity in detection and tracking tasks. In *Proceedings of 2005 IEEE Swarm Intelligence Symposium*, pages 257–264, Pasadena, CA, June 2005. IEEE Computer Society Press.

M. Scheutz, P. Schermerhorn, R. Connaughton, and A. Dingler. SWAGES - an extendable distributed experimentation system for large-scale agent-based alife simulations. In *Proceedings of Artificial Life X*, pages 412–419, June 2006.

M. Scheutz, P. Schermerhorn, J. Kramer, and D. Anderson. First steps toward natural human-like HRI. *Autonomous Robots*, 22(4):411–423, 2007.

M. Scheutz, J. Harris, and S. Boyd. How to pick the right one: Investigating tradeoffs among female mate choice strategies in treefrogs. In *Proceedings of the Simulation of Adaptive Behavior 2010*, pages 618–627, 2010.

Steinman. Discrete-event simulation and the event horizon. In *PADS '94: Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, pages 39–49, New York, NY, USA, 1994. ACM Press.

Glossary

SWAGES SimWorld Agent-based Grid Experimentation System

ADE Agent Development Environment.

NNSIM Neural Network Simulator.

HPC High Performance Computing.