Joslyn, Cliff A; Rocha, Luis and Marathe, Achla: (2000) "Development Environments and Systems Architectures for Hybrid Agent-Stochastic Event Models of Socio-Technical Organizations", Los Alamos Technical Report LAUR = 01-4693

Development Environments and Systems Architectures for Hybrid Agent-Stochastic Event Models of Socio-Technical Organizations

Cliff Joslyn, Luis Rocha, and Achla Marathe
Computer Research and Applications Group (CIC-3)
Los Alamos National Laboratory
{joslyn,rocha,achla}@lanl.gov
http://www.c3.lanl.gov/~{joslyn,rocha,achla}

January, 2001

Abstract

This is a public version of the report delivered in February, 2000 from the Los Alamos National Laboratory team on the Decision Structures (DS) project to the Physical Science Laboratory (PSL) of the New Mexico State University (NMSU). It includes a proposed simulation environment for hybrid agent stochastic-event modeling; a proposed experimental paramaterization for agent models in general; and a consideration of various platforms for the development of agent models, with special attention on Swarm and DEVS/HLA.

Contents

1 Introduction **2**

2 A Hybrid Simulation Environment for the DS Project **2**
2.1 Hybrid Simulation Environments 2
2.2 Specific Architecture 5

3 Experimental Parameterization of Agent Simulations **7**

4 Development Environments **9**
4.1 Requirements 9
4.2 Primary Development Environments 10
4.2.1 DEVS/HLA, University of Arizona [5] 10
4.2.2 Swarm, Santa Fe Institute [18] 13
4.2.3 Comparison and Recommendation 13
4.3 Secondary Development Environments 14
4.3.1 Mozart/OZ (European academic consortium) [12] 14
4.3.2 AgentSheets [1] 14
4.3.3 The Paracell Programming Environment (Flavors Corp.) [13] 15
4.3.4 Strictly Declarative Modeling Language (SDML) (Centre for Policy Modeling, Manchester Metropolitan U.) [17] 15
4.3.5 Ascape (Center on Social and Economic Dynamics, The Brookings Institute) [3] 15
4.3.6 Repast (University of Chicago's Social Science Research Computing) [14] . . 16
4.3.7 Logic for Ecological Modeling (LEM, Serguei Krivov, Jawaharlal Nehru University) [11] 16

1 Introduction

This document concludes the work of the Los Alamos team on the Decision Structures (DS) project with PSL/NMSU with FY99 funding. It covers the period from November 1999 through February 2000, and includes discussion of:

- A proposed simulation environment for hybrid agent stochastic-event modeling developed out of discussions at the November 1999 project meeting.
- A proposed experimental paramaterization for agent models in general, attempting to develop an experimental framework for discovery of dependencies among independent factors in agent models.
- And finally, a consideration of various platforms for the development of agent models, with special attention on Swarm and DEVS/HLA.

2 A Hybrid Simulation Environment for the DS Project

At the November, 1999 project meeting we decided as a group on the proper path forward to integrate the work on data analysis and agent simulation with existing sources of data. It was evident that acquiring real data would be next to impossible in the short- to medium-run, and therefore it was decided that simulated data would have to used instead. A stochastic-event simulator was to be made available to us for this purpose.

We thus entered into constructing a complex architecture, where data analysis would be performed on one set of simulated data, and agent models providing a different set of simulated data, all interacting with the data analysis modules. At this point, it became clear that some explication of the issues involved in such an undertaking would be useful. In this section we first address these issues in general, and then describe the details of the specific architecture decided upon which would instantiate this general scheme.

2.1 Hybrid Simulation Environments

We begin by addressing some ambiguity which has recently been perceived to exist concerning the relations between the three main thrusts of the project: the signal analysis, the problem definition, and the agent simulation.

Our perspective on this begins with our position on modeling and simulation in general, which can be summarized using the **modeling relation** shown in Fig. 1.

Given a real system, certain data is measured from that system, and a simulation of certain aspects of that system is constructed through the modeling task. The simulation in turn generates simulated data, which is compared to the measured data. If there is a high degree of corroboration (validation) between the simulated and measured data, then this is inductive evidence to support the hypothesis that the simulation has accurately captured the particular features of the real system which were selected by the modeling task.

Of course, in any particular application, the entire process is dependent on the appropriate choice of measured variables, aspects of the system to be simulated, and a technologically sound simulation process. Note also that the simulation must produce simulated data whch is the same *type* as the measured data, and then an appropriate corroboration measure must be chosen based on that type.

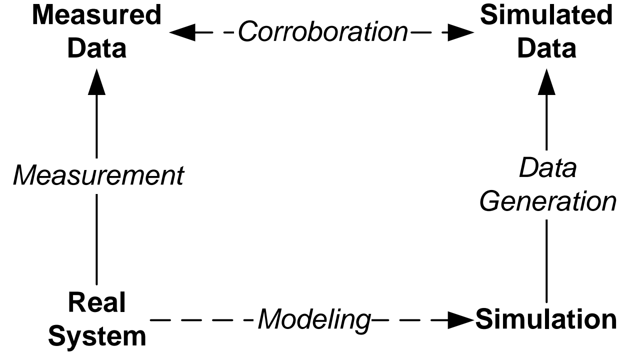

Figure 1: The modeling relation.

This is the general pattern of traditional simulation. In our project, at least two components are attempting to do a task substantially different from traditional simulation. In Fig. 2, the traditional simulation problem is shown in the bottom. The Signal Analysis group is attempting to analyze data, whether measured data from a real system or simulated data from a traditional simulation, in order to develop high-level representations of that data in terms of dynamical systems concepts and the related research thrusts, namely combinatorial homotopy and Chu spaces. Simultaneously, the Agent Modeling group is attempting to develop relatively abstract high-level models of the general class of target systems, rather than high-fidelity simulations of particular examples of the target system.

In effect, we are recapitulating the traditional modeling relation, but at a higher level of abstraction. It follows that now, it is rather the high-level representations produced by the Analysis Group and the data generated by the high-level simulation which must be of the same type, so as to be able to have appropriate corroboration (validation).

Also, note that in this context, the Ping model [4] is actually playing the same role as the Agent model. In other words, it is a high-level simulation which is attempting to capture appropriate abstract properties of systems of the type of the target system, and produces data comparable to that which will eventually be produced from the high-level analysis. Thus our work in particular should involve a close consideration of the Ping model, either as a point of departure for our models, or at least as providing guidance for their design.

As stated in Joslyn's August deliverable [7] this stance has both advantages and disadvantages. In particular, we gain generality and flexibility, but lose fidelity and precision. However, in this project where we are attempting to gain insight into the emergent structures in systems of this type, and wish not to embed too much initial structure into our models, this is an appropriate decision.

To summarize, our view of the proper relations among the project components is:

Signal Analysis: Develop and validate high-level representations.

Agent Model: Develop high-level simulations which appropritately capture target system properties and generate data for the Signal Analysis team.

Problem Definition: Work with low-level simulations sufficiently to provide guidance to the other teams about appropriate forms of abstraction.

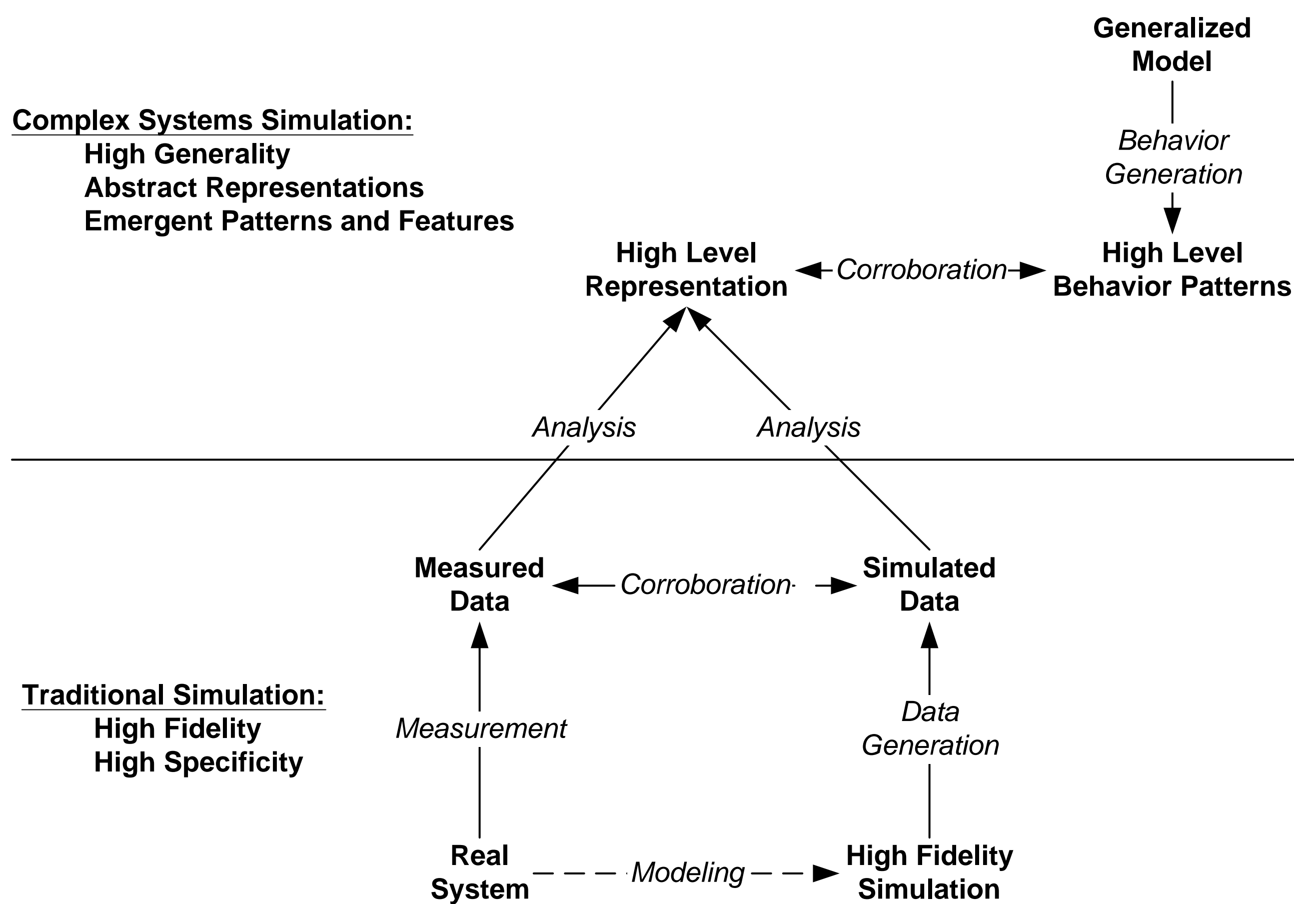

Figure 2: Simulation in complex systems modeling.

2.2 Specific Architecture

The argument above results in a proposal for a specific architecture for our hybrid simulation problem. It is illustrated in Fig. 3 and explained below.

1. **Stochastic Event Simulator:** A detailed, high fidelity simulation, which is standing in for reality for us.

   (a) The **vignette** as a scenario is used to instantiate the simulator proper.

   (b) The **simulator** runs a simulation of the vignette, at PSL.

   (c) **Message logs** and/or an **event logs** would provide source data.

2. **Data Analysis:** A variety of tools to construct high-level representations from the data log output, and provided by the signal analysis researchers on the PSL team. We cannot speak to the specifics here, but this would include at least:

   (a) **Influence Chains:** From Laubenbahcer's combinatorial homotopies.

   (b) **Chu Representations:** From Gherke's work.

   (c) **Other Output:** From work developed by Reinfelds.

3. **High Level Features:** Various representations of high level, abstract features of the DR of the target organization.

   (a) **Induced** features are derived from data analysis of the data logs of the detailed simulation, as specified above.

   (b) **Constructed** features are used to instantiate the agent model (below). They are derived initially by hand directly from abstracting the original vignette (e.g. a highly limitted number of units, unit types, unit capabilities, message types, etc.). It is also intended that later on the induced features will provide a source for constructed features.

   (c) **Generated** features are produced directly from the agent model (below).

   (d) **Corroboration** must be performed to check the "distance" between the features generated from the agent model and those induced from the high fidelity simulation. Much is hidden in this arrow. What will ultimately be included is highly dependent on the nature of the derived features and the level of abstraction of the agent model, and therefore cannot be specified now.

4. **Agent System:** The agent system is intended to capture the high level, abstracted features of the DS of the target organization within a simulation environment.

   (a) The **agent model** is a high level abstraction of features of a command and control organization.

   (b) The **agent shell** is the basic engineering platform in which the agent model in implemented. Multiple platforms were considered (see Sec. 4). At the time this design was completed, the platforms under consideration had the following statuses:

   Swarm: Installed at both LANL and PSL. PSL would have provided support for Objective C Swarm, and LANL was to have developed a JAVA capability.

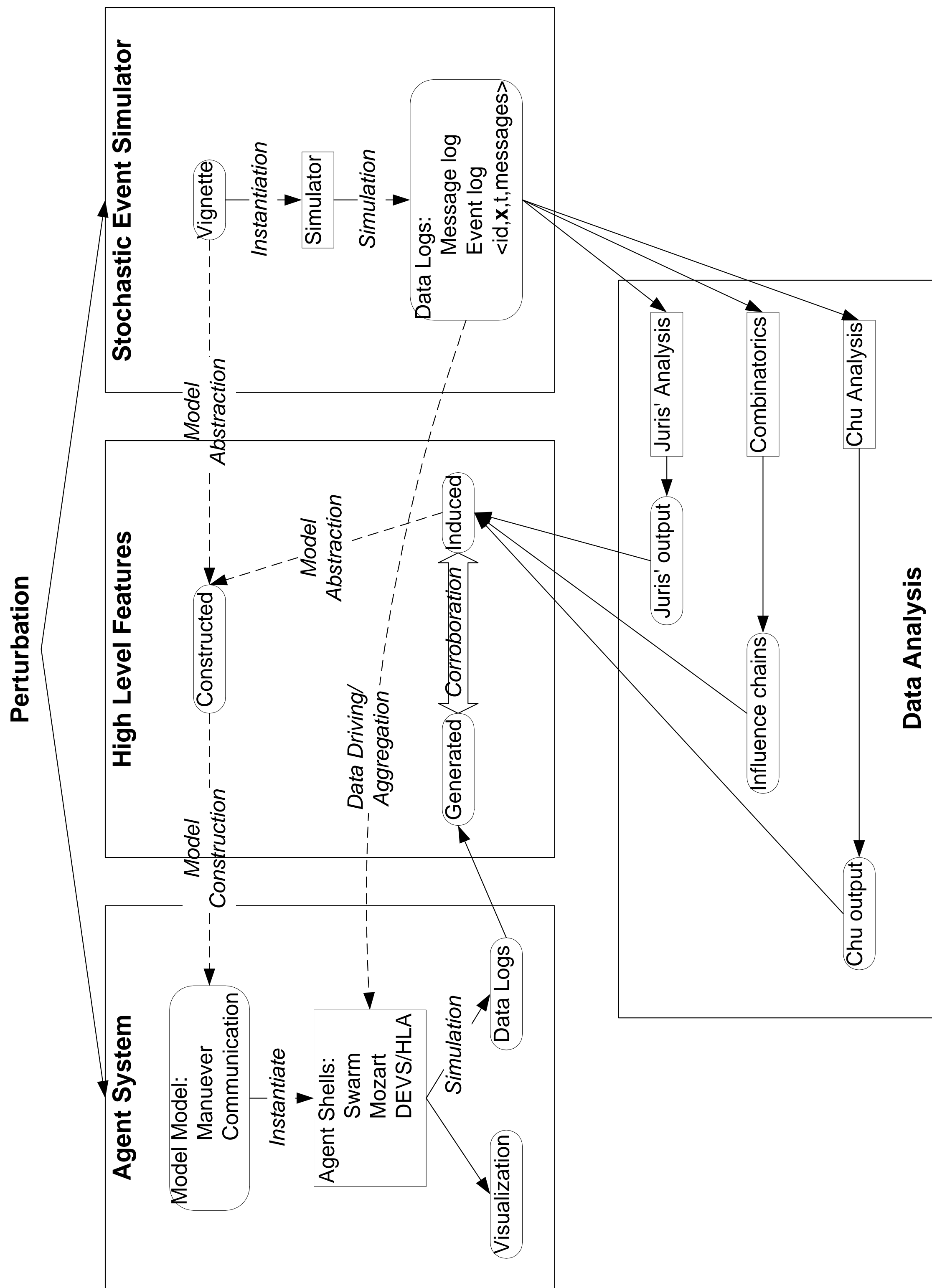

Figure 3: Proposed testbed architecture for hybrid simulation environment.

Mozart/Oz: Being pursued by Reinfelds at PSL.

DEVS/HLA: Considered in detail by Joslyn and Marathe (see Sec. 4.2.1). At the date of writing, this platform had just been made available to the LANL team, and was installed there.

(c) **Data logs** produced from the agent simulation will supply (directly or indirectly) the generated features.

(d) **Visualization** includes the simulation environment (grid, terrain), atomic unit positions, center-of-mass and uncertainty distribution of aggregated units, and "lightning bolts" of communication events among units at multiple echelon levels.

(e) **Data Driving/Aggregation:** Initially the agent shell was not to be used to generate simulated data, but rather its visualization was to have been driven directly from the simulator data logs.

5. **Perturbation:** The ultimate goal is to simulate a perturbation in both the stochastic event simulator and the agent model. In the simulator this will be done by altering average message delay time and loss rate. Similar capabilities, at an appropriately abstracted level, must exist in the agent model. Given a perturbation to either the simulator or agent models or both, appropriate changes should be measurable at the corroboration step.

The February 2000 demo was intended to include:

1. Running the simulator with log capture.

2. Display of data analysis output.

3. Data driven aggregation and visualization in the agent shell.

The Los Alamos team was to have been responsible for 3 implemented in either Swarm or DEVS/HLA. PSL was to have provided appropriate engineering support, beginning with the specification of the data logs. Between November 1999 and February 2000, this did not occur, and in fact, there was very little communication to the LANL team from PSL. As we were preparing to execute our responsibilities, we were informed that funding would not be renewed past February 2000, and that this report would be our final deliverable.

3 Experimental Parameterization of Agent Simulations

In a series of published papers [8] and delivered reports [6, 7, 15] within this project, and others published outside of this project [9, 10, 16], Rocha and Joslyn have developed the conceptual basis for semiotic agent-based modeling and simulation of socio-technical organizations. In doing so, we have been impressed by the complexities involved in even the simplest semiotic agent systems, which, if complete, necessarily involve at least the following three components:

1. Agents interacting with a virtual physical environment.

2. Agents interacting with each other through both communication actions and actions into the "relative environments" (those including other agents).

3. And some sort of even simple knowledge or endo-model structures interior to the agents.

We have emphasized that each of these three factors induces constraints on the freedom of action and decision which systems must necessarily possess in order to be identified as agents or semiotic agents.

Here we would like to present one result of some of this analysis. In particular, we have in mind a series of staged, controlled experiments through which particular aspects of agent simulations could be included, excluded, or combined, in order to gauge the sensitivity to these factors for various particular problem types. Some of these factors are qunatitative, in that they admit to degrees; others are qualitative, meaning that they are either present or absent.

Virtual Environments: The first set of factors involve the nature of the virtual "physical" environment in which the agents interact.

- **Strength of physical laws:** The properties of the (virtual) physical laws manifested in then artificial environment may be more or less constraining on agent behavior. In one limit, it is determining, meaning that the agents have no autonomy of action. In the other, there are no such constraints, meaning an absence of physical laws. This parameter describes the amount of coupling between agents and their environments.
- **Size of the environmental state space:** Simiarly, the number of indepdent variables (dimensions) in the environment can be small or large. Also, the number of different values (states) each dimension might have can vary. Together, they yield the size of the overal state space of the artificial environment.

Individual Agent Properties: The second set of factors involve the inherent properites of the agents themselves, in interaction with the environment.

- **Number and kinds of agent sensory or action modalities:** The modalities by which the agents can sense their environments, and then in turn act back into them, are, of course, crucial. There can be either more or less of these, and more or less different from each other. For example, some agents might be able to perceive both position and color of other agents, or affect both position and orientation.
- **Amount of memory:** We have argued that semiotic agents should be based on an architecture of state machines with memory. We have also observed that learning (for example, in iterated prisoner's dillema problems) can be highly sensitive to the amount of memory available within the agent. Thus this is a crucial quantitative parameter. One limit (no memory) should recover classical collective automata systems, while there need not effectively be any upper limit on the amount of memory.

Collective Agent Properties: The final set of factors are the most important and most interesting, and involve the nature of the interaction among agents in the community. Below it is important to recall that in our approach we have decided to distinguish actions proper which agents take into their environments from semiotic actions of creating and passing communicative tokens.

- **Degree of agent interdependence vs. autonomy:** Just as agents can be more or less coupled to their virtual environments, they can also be more or less coupled to each other. This describes the degree to which agents are either interdependent or autonomous, due either to proper or semiotic actions.

- **Semiotic tokens:** Qualitatively, while we assert that all agents must engage in proper actions with respect to their environments, they may or may not engage in semiotic actions. If so, then to other factors come to bear.
  - **Token passing:** Tokens may or may not be passed to other agents.
  - **Token storage:** Tokens may or may not be stored in memory.
  - **Presence of syntactic relations among tokens:** Tokens may or may not be able to be combined to create new tokens at a higher level of syntax.
- **Common knowledge structures:** Understanding knowledge as semiotic tokens which are interpreted, there may be interpretations held in common by different agents.
  - **Cultural vs. genetic transmission:** Knowledge can be shared "horizontally" with cultural transmission among agent contemporaries, or (assuming reproduction of agents) passed "vertically" with genetic transmission between agent generations.
  - **Amount of shared knowledge:** As we have noted, research has indicated that the amount of knowledge of the world shared among multiple agents (however transmitted) can be a crucial factor in the robustnes of their behavior.

4 Development Environments

We have recognized many senses and conceptions of the agent modeling paradigm [7, 15]. The primary schools are based either in Artificial Intelligence (AI) or Artificial Life (ALife). There are also many programming languages and software modeling environments to support them. We note that most existing platform are based on the AI approach, while ours is more rooted in the ALife approach.

In this section we do not provide a comprehensive survey of the field, referring the reader to other public sources of information on agent modeling environments for that [2]. Rather, here we review a few platforms of particular interest. These have either a special interest for the ALife approach to agent modeling, or have been brought to our attention for other reasons.

Below we first consider our general requirements for a development platform. We then move on to describe and compare the primary development environments we considered, namely Discrete Event SystemS/High Level Architecture (DEVS/HLA) from the University of Arizona, and Swarm from the Santa Fe Institute. We focus especially on the technical details of DEVS/HLA, as it is not familiar to the PSL team. We conclude by describing a number of other environments which are also available for developing agent-based models.

4.1 Requirements

We are motivated by the following requirements:

Rapid Prototyping Environment: Relatively light, flexible environment, without a strong committment to a particular mathematical methodology, and ability to swap in and out different agent internal structures, environmental properties, and agent interaction and communication.

Instrumentation: Ability to get real-time output of agent state variables, behavior, etc., with accompanying display.

Graphics: Support for graphical display of instrumentation, agent environment, etc.

Hierarchical Support: Support for agents embedded in multiple hiererchicaly spatial and temporal scales.

Environment: Support for embedding agents in discrete and continuous spatial and temporal environments.

Programming: An industry-standard programming language environment is preferred, although specialized languages can provide some advantages.

4.2 Primary Development Environments

We now describe the principle feature of the two primary development environments we examined. We deal especially with the details of DEVS/HLA, as it is both new to the PSL team, and highly appropriate for our project.

4.2.1 DEVS/HLA, University of Arizona [5]

DEVS (Discrete EVent Systems) is a long-standing approach to discrete event modeling based on mathematical systems theory. HLA (High-Level Architecture) is an IEEE and DMSO standard for producing inter-operable and distributed simulations. The advantange of DEVS as a discrete modeling environment is its generality, derived from its elegant rooting in mathematical systems theory, its close connection to formal automata theory, and its scalability.

DEVS Formalism Following Zeigler [20, 21], we have (Fig. 4):

Definition 1 (Atomic DEVS) Let $\mathcal{D} := \langle X, Y, S, t, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda \rangle$, where:

- X is a set of external events;
- Y is a set of outputs;
- S is a set of states;
- $t : S \mapsto [0, \infty]$ is the time advance function, where $[0, \infty] := \mathbb{R}^+ \cup \{\infty\}$;
- $\delta_{\text{int}} : S \mapsto S$ is the internal transition function;
- $\delta_{\text{ext}} : Q \times X \mapsto S$ is the external transition function, where

$$\begin{aligned} Q &:= \{\langle s, e \rangle : s \in S, 0 \le e \le t(s)\} \\ &= \bigcup_{s \in S} [0, t(s)] \subseteq S \times [0, \infty] \end{aligned}$$

  is the total state set; and

- $\lambda : S \mapsto Y$ is the output function.

These functions are interpreted as follows. Assume that $\mathcal{D}$ is in state $s \in S$. If an event $x \in X$ arrives after a duration $e \le t(s)$, then $\mathcal{D}$ transits to state $\delta_{\text{ext}}(\langle s, e \rangle, x)$. If no event arrives before $t(s)$, then s is said to expire, and $\mathcal{D}$ transits to state $\delta_{\text{int}}(s)$. Finally, at all times, $\mathcal{D}$ produces output $\lambda(s)$.

This basic formalism has been extended recently to handle concurrency.

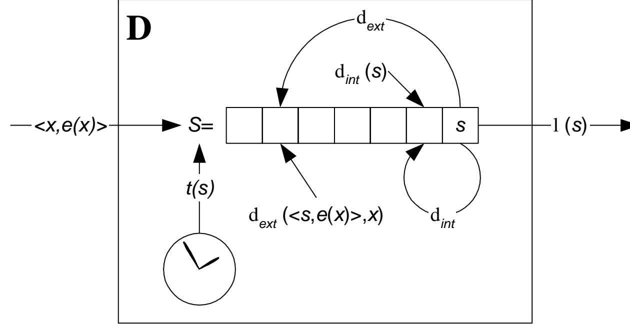

Figure 4: The basic DEVS formalism.

Definition 2 (Extended Atomic DEVS) Let $\mathcal{D}^+ := \left\langle X, Y, S, t, \delta_{\text{int}}, \delta^+_{\text{ext}}, \delta_{\text{con}}, \lambda \right\rangle$, where X, Y, S, t, λ, and δ_{int} are as before, but:

- $\delta^+_{\text{ext}}: Q \times X^b \mapsto S$ is the extended external transition function, where Q is as above, but X^b is the set of bags (ordered collections possibly with duplicates [19]) over X; and
- $\delta_{\text{con}}: Q \times X^b \mapsto S$ is the confluent transition function.

The use of bags here allows the representation of concurrency: any bag with more than one element represents the simultaneous occurrence of these events. The extended external transition function represents the case when all the events are input events. The confluent transition function represents the case when some input events occur simultaneously with the expiration of the time advance function.

Finally, given DEVS atomic models, whether basic or extended, coupled models can be constructed to create a hierarchically structured system, here presented in the classical (non-parallel) version:

Definition 3 (Coupled DEVS) Let $\mathcal{D}^C := \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, \text{select} \rangle$, where X and Y are as before, but:

- $D = \{d\}$ is a set of indices of component models;
- $\forall d \in D$, the M_d are component DEVS (whether atomic or coupled);
- $\forall d \in D \cup \{N\}$, the $I_d \subseteq D \cup \{N\}$ are the influencer set of d, where N indicates the environment of $\mathcal{D}^C$;
- $\forall d \in D$, $Z_{i,d}$ is the i to d output translation with:
  - $i = N \rightarrow Z_{i,d}: X \mapsto X_d$, mapping external inputs to component inputs;
  - $d = N \rightarrow Z_{i,d}: Y_i \mapsto Y$, mapping component outputs to external outputs;
  - $d \neq N \neq i \rightarrow Z_{i,d}: Y_i \mapsto X_d$, mapping component outputs to component inputs.
- select: $2^D \mapsto D$, with $\text{select}(E \subseteq D) \in E$ is a tie-breaking function to arbitrary the occurrence of simultaneous events (note this is not necessary in the parallel version).

The most important thing which follows from this is:

Theorem 4 (Closure Under Coupling) Every coupled DEVS $\mathcal{D}^C$ can be transformed into an extended atomic DEVS $\mathcal{D}^{+'}$ in some appropriately generalized space $\langle S', X', Y' \rangle$.

Thus it is garaunteed that DEVS can be connected in hierarchical patterns of arbitrary complexity, and always yield another DEVS.

Implementations The current primary implementation is DEVS/Java. We have installed DEVS/Java in a Solaris environment, and have made some preliminary explorations. We have also received the DEVS/Java Pursuer model, a 2-D grid-based combat model which has properties similar to DS applications. It was developed by UA for the Joint Measure project for the Navy. Regretably, we received this software at the very end of the performance period.

Furthermore, we have identified staff at LANL who are very interested in DEVS/Java, not only for potentially participating in our effort, but also for applying it to other projects in computer systems simulation currently ongoing at LANL.

The advantages of DEVS/JAVA include its explicit support for hierarhical agent models, the HLA layer, the fact that DEVS is a general (if particular) methodology, and that it is JAVA-based. Its disadvantages are that it is a rather poor development environment, and that DEVS is a particular (if general) methodology.

We will now review these specific points.

Parallel DEVS Formalism: The extensions to the basic DEVS formalism to include bags of input events allow simulation of concurrency. This has been motivated by the requirements for parallelism necessary for distributed simulation environments. This extended formalism has also been proved to be closed under composition and hierarchical construction. They are available in the current implementations.

Variable Structure Modeling: DEVS implementations rely on a $\langle \text{port}, \text{value} \rangle$ structure for inputs and outputs, where "value" can be entity, in particular another DEVS. This facilitates much recent academic work on DEVS for variable structure modeling, or the ability of a simulation to change its structure while running. The current implementation supports adding and deleting components and moving components within the model hierarchy.

Development Environment: DEVS/Java provides a large Java class library to support the basic modeling methodology. Particular models extend these classes, and a number of computer-engineering-based basic models are provided which demonstrate these capbilities. Container libraries are also provided.

External Interfaces: It is important to consider what portions of an overall simulation effort should be included within the DEVS umbrella and what support services should be considered as outside of it; or, in other words, should *everything* be a DEVS component? and if not, how do we know? The answer is that generally, those components which handle autonomous scheduling should be considered candidates to be made into DEVS components. For example, the UA team has developed simple DEVS display components which make Java graphics calls, since these conincide with model events. Batched file input, on the other hand should be regarded as "synchronous", and thus written in native Java called from a DEVS component.

Overall Model Support: DEVS models are trees, with atomic leaves combined into higher-level coupled models. There is currently no support for working on this graph overall, as opposed to within each node. There is a tree-structured documentation tool, but no way to view or modify the overall hierarhy.

HLA and Corba Layers: An important new development is the introduction of external interfaces to DEVS/Java. One is to the High-Level Architecture (HLA), which is a DMSO-approved interoperability standard for simulations. Both DEVS/Java and DEVS/C++ now have an HLA layer, which is useful for protection for both secure and proprietery information, and, of course, for interoperability.

DEVS is a powerful and flexible discrete event modeling environment with many attractive features with respect to agent modeling. In particular, it may provide a superior platform to Swarm with respect to basic model scheduling, model composition and hierarchical structures, and variable structure modeling, all of which are important for the DS project. Furthermore, the HLA layer can prove to be an important asset as our agent models are connected to other simulation packages, e.g. our stochastic event simulator.

Where DEVS/Java is weak is on the support provided by the development environment, especially graphical elements. On the other hand, such capabilities are increasingly available in the Java environment in which DEVS now lives. Furthermore, the UA team is able to provide examples of graphical packages which they have developed for projects similar to ours, in particular the Joint Measure project with the Navy within which the HLA layer was developed. Beyond that, UA is able to provide substantial support for continued enhancement and development of the software platform. Such assistance is available at student rates, and can be flexibly negotiated.

4.2.2 Swarm, Santa Fe Institute [18]

Swarm is a well established object-oriented agent modeling platform developed by the Santa Fe Institute for Artificial Life agent-based simulation.

Swarm is a set of Objective C libraries, now supplemented with a Java layer. We have installed and have available demo applications for both versions.

It has been the assumed default platform for the Decision Structures project. Since members of the DS project are quite familiar with Swarm already, we do not it in detail here. Rather, we focus on pointing out some particular aspects of Swarm and how they compare with DEVS for our application.

4.2.3 Comparison and Recommendation

Swarm supports virtually all of the requirement above, especially the hierarchical architecture, instrumentation (called "probes"), and graphical and environmental support. On the other hand, Swarm is definitely a programming environment in which to develop applications, and is thus not the lightest prototype environment possible.

Furthermore, unlike DEVS, Swarm has no unifying mathematical framework for discrete event simulation. Rather, it is a collection of programming structures and practices which allows efficient construction of different agent models. Basically, DEVS provides a solid platform with a long track record and, most importantly, provable analytical properties. Swarm is a programming environment which support systems development in which systems properties can be demonstrated and experimented on.

For example, in Swarm scheduling is supported within hierarchical models by merging atomic schedules to higher and higher levels, eventually constructing a global schedule for the application. In DEVS there can no global schedule and no global clock. Instead, the only global information propogated up the model hierarchy is the time of the next event.

This is both an advantage and a disadvantage for DEVS. The DEVS methodology is a particular one, while Swarm allows the implementation of *any* mathematical methodology. However, the DEVS methodology, while particular, is also very *general*, falling out from the history of mathematical developments in mathematical systems theory as the *minimally complex* system necessary to perform discrete event simulation. It has been proved that classical simulation methodologies from differential equations to cellular automata are all expressible as DEVS [21].

This can be expressed (here with little explanation) by:

Theorem 5 (Universality of DEVS) Every DEVS-like system is a homomorphic image of a DEVS I/O system.

Theorem 6 (Uniqueness of DEVS) Every DEVS-like system has a canonical realization by a DEVS I/O system.

Here "DEVS-like system" is referring to systems with DEVS interfaces, which can be understood as any system with an input-output interface, which is a shockingly broad class of systems.

Thus one quite interesting suggestion we would like to make is that the Swarm programming environment be used, but to *implement* DEVS systems and methodology. This is one way to approach the problem of capturing the best of both worlds: use a general-purpose programming platform which supports agent programming and simulation in general, but implement these agents consistent with a proven mathematical methodology.

4.3 Secondary Development Environments

We now very briefly survey a few other packages which we have been made aware of to support agent simulations.

4.3.1 Mozart/OZ (European academic consortium) [12]

Mozart-Oz is a programming language and a program development system that is well suited for the implementation of agent based simulations in particular and distributed computations in general.

Mozart-Oz is a mature system that has been under development for about 10 years in a collaboration between German Institute of AI, Univ of Saarbruecken, Univ of Louvain and the Swedish Institute of Computer Science. Its web page points to a very impressive list of papers on its structure, implementation and applications.

Two papers on it will be presented at the Internat Logic Prog Conf 1999 in Las Cruces NM Nov 29 - Dec 3, 1999. The paper on agent related simulations (see above URLpapers) seems particularly relevant to our work.

Mozart is a programming environment that permits seamless transfer of Oz program execution from one machine to a cluster of many machines. Oz is a programming language that combines the best features of Logic, functional, imperative. and object oriented programming into a powerful and elegant programming language, much like UNIX once combined the best features of then current mainframe operating systems into a new and elegant operating system that we now like so much.

4.3.2 AgentSheets [1]

This is a very interesting spreadsheet-oriented approach to generalized agent modeling, which would be appropriate for rapid prototyping. Graphical support and probing are implicit. Too bad it runs only on Mac! From the main web page:

Agentsheets is an authoring environment developed by Alex Repenning at the Center for Lifelong Learning and Design (L3D) located in the University of Colorado at Boulder. It features a versatile construction paradigm to build dynamic, visual environments for a wide range of problem domains such as art, artificial life, distributed artificial intelligence, education, environmental design, object-oriented programming, simulation and visual programming. The construction paradigm consists of a large number of autonomous, communicating agents organized in a grid, called the agentsheet. Agents can use different communication modalities such as animation, sound and speech. To view a sample of Agentsheets applications select here.

The construction paradigm supports the perception of programming as problem solving by incorporating mechanisms to incrementally create and modify spatial and temporal representations. In a typical utilization of Agentsheets, designers will define the look and behavior of agents specific to problem domains. The behaviors of agents determine the meaning of spatial arrangements of agents (e.g., what does it mean when two agents are adjacent to each other?) as well as the reaction of agents to user events (e.g., how does an agent react if a user applies a tool to it?).

4.3.3 The Paracell Programming Environment (Flavors Corp.) [13]

Flavors is the former leader in knowledge systems programming, object-oriented LISP. Paracell is a massively parallel programming environment. Agents are implemented as simple single rule-based components in a uniform topology and architectures.

Paracell is mostly used for manufacturing process control, job-shop scheduling, etc. It is more appropriate for hardware development applications than the agent environments we are considering.

4.3.4 Strictly Declarative Modeling Language (SDML) (Centre for Policy Modeling, Manchester Metropolitan U.) [17]

SDML is a declarative modelling language based on rulebases and databases and implemented in Smalltalk. Models can be constructed from many interacting agents, and hierarchical agents are supported. Although the primary paradism is forward and backward chaining, partial object-oriented facilities are available, as well as hierarchical temporal composition. Primary applications have been in economic and market modeling.

4.3.5 Ascape (Center on Social and Economic Dynamics, The Brookings Institute) [3]

From the main web page:

> Ascape is a software framework for developing and analyzing agent-based models. In Ascape, agent objects exist within scapes; collections of agents such as arrays and lattices. These scapes are themselves agents, so that typical Ascape models are made up of "collections of collections" of agents. Scapes provide a context for agent interaction and sets of rules that govern agent behavior. Ascape manages graphical views and collection of statistics for scapes and provides mechanisms for controlling and altering parameters for scape models.

Ascape is written in Java. Primary applications are economic and market modeling.

Marcus Daniels, the primary engineer of the Swarm Development Group, says "Ascape has strong features for doing lockstep/temporally-flat, CA-like simulations. It includes several fairly interesting/complex models" [Swarm-support email list, 4/13/00].

4.3.6 Repast (University of Chicago's Social Science Research Computing) [14]

Java classes to support agent-based modeling. From the web page:

> It provides a library of classes for creating, running, displaying and collecting data from an agent based simulation. In addition, Repast can take snapshots of running simulations, and create quicktime movies of simulations. Repast borrows much from the Swarm simulation toolkit and can properly be termed "Swarm-like." In addition, Repast includes such features as run-time model manipulation via gui widgets first found in the Ascape simulation toolkit.

From Daniels:

> Repast doesn't quite seem to be as responsive to me as Ascape. Of course, that's pretty much a useless remark. But, hey, so long as we're cloning each others' GUIs, I guess it is in some sense it is appropriate.
>
> Repast has a more general scheduling engine than Ascape. It's a lot like Swarm's, but uses some different idioms. And of course, Repast and Ascape are 'pure java', which in principle means code portability is less of a problem, i.e. it is easier to make demos on the web, migrate code, and that sort of thing.
>
> Finally, Repast has a nice multilayer raster display, where you can turn different layers on and off. (In heatbugs, for example, you can see the bugs, the diffusion space, or both.)
>
> Repast is a respectable Java-based simulator implementation with interfaces highly similiar to many in Swarm.

4.3.7 Logic for Ecological Modeling (LEM, Serguei Krivov, Jawaharlal Nehru University) [11]

An interesting and flexible package, but too specifically tailored to ecological models. From the web page:

> LEM supports interactive development of individual based ecological models. There is a special language for specification of models. No knowledge of computer languages is required for development of new models, and it seems that LEM is the first general purpose individual-based modeling tool that could completely relieve ecologists and artificial life researchers from programming work. LEM is a forward chaining rule based system. The behavior of agents and reaction of environment on the actions of agent are specified by the sets of production rules. In evolutionary models each agent has its own rule base that also serves as a set of chromosomes. In simple ecological models the rule base is shared by the specie of agents and it should be preprogrammed. There is a special rule table that defines the reactions of the environment to the actions of agents. LEM runs under Windows 95/98/NT. It could be ported to OS/2. It has professional quality visual interface comparable with the best examples of Windows programming. LEM comes with the set of tools that facilitate development of individual based models.

References

[1] *AgentSheets*, http://www.cs.colorado.edu/homes/ambach/public_html/agentsheets_html/agentsheets_page.html

[2] *UMBC Agent Web*, http://agents.umbc.edu/

[3] *Ascape*, http://www.brook.edu/es/dynamics/models/ascape/

[4] Coombs, Mike; Birx, Don; and Jarrah, A et al.: (1999) "Combinatorial Time Series Analysis of Influence Structures in Complex Information Networks",

[5] *DEVS/HLA*, http://www.acims.arizona.edu/DEVS_HLA/devs_hla.html

[6] Joslyn, Cliff: (1999) "Agent Modeling from a Semiotic Perspective", http://www.c3.lanl.gov/~joslyn/nmsu/semagent.pdf

[7] Joslyn, Cliff: (1999) "Semiotic Agent Models for Simulating STOs", http://www.c3.lanl.gov/~joslyn/nmsu/semagentp.pdf

[8] Joslyn, Cliff: (2000) "Virtual Environments as Constraints on Decision-Making in Agent Models of Socio-Technical Organizations", in: *2000 Workshop on Virtual Worlds in Simulation*, ed. K. Bellman and C. Landauer, http://www.c3.lanl.gov/~joslyn/vwsim00/vwsim00abs.pdf

[9] Joslyn, Cliff and Rocha, Luis: (2000) "Towards Semiotic Agent-Based Models of Socio-Technical Organizations", in: *Proc. AI and Simulation 2000*, ed. HS Sarjoughian *et al.*, pp. 70-79, http://www.c3.lanl.gov/~joslyn/ais00/ais_final.pdf

[10] Joslyn, Cliff and Rocha, Luis: (2000) "Semiotic Agent-Based Modeling", in: *Society for Computer Simulation Transactions on Simulation*, in preparation

[11] *Logic for Ecological Modeling*, http://geocities.com/SiliconValley/Cable/3109/LEM.html

[12] *The Mozart Programming System*, http://www.mozart-oz.org

[13] *The Paracell Programming Environment*, http://www.flavors.com/HTML%20Presentation%20folder/index.htm

[14] *Repast*, http://repast.sourceforge.net/

[15] Rocha, Luis M: (1999) "Review of Agent Models: Encounters, Strategies, Learning, and Evolution", http://www.c3.lanl.gov/~rocha/psl/agent_review.pdf

[16] Rocha, Luis M and Joslyn, Cliff: (1998) "Simulations of Evolving Embodied Semiosis: Emergent Semantics in Artificial Environments", in: *Proc. 1998 Conf. on Virtual Worlds in Simulation*, pp. 233-238, Society Comp. Sim., San Diego

[17] *SDML: A Strictly Declarative Modelling Language*, http://www.cpm.mmu.ac.uk/sdml/

[18] *Swarm*, http://www.swarm.org

[19] Yager, Ronald R: (1986) "On the Theory of Bags", *Int. J. of General Systems*, v. **13**:1, pp. 23-38

[20] Zeigler, BP: (1985) *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, San Diego

[21] Zeigler, BP; Praehofer, Herbert; and Kim, Tag Gon: (2000) *Theory of Modeling and Simulation, 2nd Edition*, Academic Press