Multiparadigm Simulations in Modeling Spread of Lyme Disease

Gilbert Chen, Boleslaw K. Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180, USA
Email: {cheng3,szymansk}@cs.rpi.edu

Thomas Caraco
Department of Biological Sciences
University at Albany
Albany, NY 12222, USA
caraco@csc.albany.edu

Keywords

Combined Simulation, Continuous Simulation, Discrete Simulation, Ecology, Differential Equation Solver.

Abstract

This paper discusses techniques for linking different models in simulating complex systems with explicit space representation. For such systems, it is convenient to decompose simulation into interacting parts which are then described by different submodels. Some of these submodels are naturally described as discrete event systems while others are easier to represent as a set of partial or ordinary differential equations. The size of modeled systems necessitates parallelization to achieve acceptable execution time. Hence, techniques for multiparadigm simulation must consider the most challenging case of linking parallel discrete event simulations using optimistic protocols with a solver of partial differential equations in two or three dimensional domains.

In this paper, we discuss first how an existing parallel discrete event simulation can be linked with a partial differential equation (pde) solver to form a single simulation in presence of rollback mechanism. An implementation of the rollback for the pde solver is described. For case of linear subsystems, we designed a fast reverse computation algorithm. For non-linear equations, the cost of reversing computation is larger than the cost of forward computation in the pde solver part.

Next, we describe further functional system decomposition in which a Parallel DES model is divided into subsystems, each describing a different type of entities. We demonstrate that such decomposition results in a different kind of synchronization requirements than the first approach.

We illustrate the new multiparadigm approach by simulating the spread of Lyme disease, a biological process involving several interacting species which differ in size and in the spatial and temporal scales of their population dynamics.

1 Introduction

Combining discrete-event simulation with continuous simulation can be advantageous, since complex systems commonly include both discontinuous and continuously varying processes. A discrete event system consists of many types of entities, each of which has a set of attributes and undergoes a series of transitions that are instantaneous in time, that are referred to as events [6]. In contrast, a continuous system is composed of a set of state variables assigned to a spatial domain and changing continuously in time. The spatial and temporal change of state variables can be represented by a set of partial (or ordinary) differential equations.

As an example, consider a seaport in which management of ship movements and anchoring can be best described by the discrete subsystem. However, at different scale of details, the acceleration and velocity of each ship is influenced by the weather and water currents which are best described by the atmospheric and shallow water continuous subsystems [13]. An example that we use in this paper is the spread of Lyme disease, which we describe in Section 2. Other examples include air traffic control, highway management systems, population dynamics of complex ecosystems, etc. Hence, multiparadigm simulation techniques are important in creating detailed models of complex, multi-scale systems and in linking existing partial models into an integrated model of more complex phenomena.

Although some techniques have been devised for developing combined discrete-event/continuous simulation, mostly in electronic circuit design and traffic control systems [2, 7], most of them focus on continuous systems with finite number of continuous state variables. That is, the state variables are only represented as a function of time, and not of both space and time. In this paper, we concentrate on the systems in which such simplification is not possible and in which the state variables of a continuous subsystem are defined over a continuous space. In the sample

application that we used here, the spread of Lyme disease, space is two-dimensional. However, our techniques will apply equally well to systems with three-dimensional spaces.

In this paper, we concentrate on linking existing models with the minimum change to the underlying subsystem models. Such approach has an added advantage of enabling developing new applications on top of existing software. This accelerates the design of a simulation model and minimizes the cost of software development. More importantly, reusing an existing software that is likely to be repeatedly run and tested for correctness increases reliability of the integrated model.
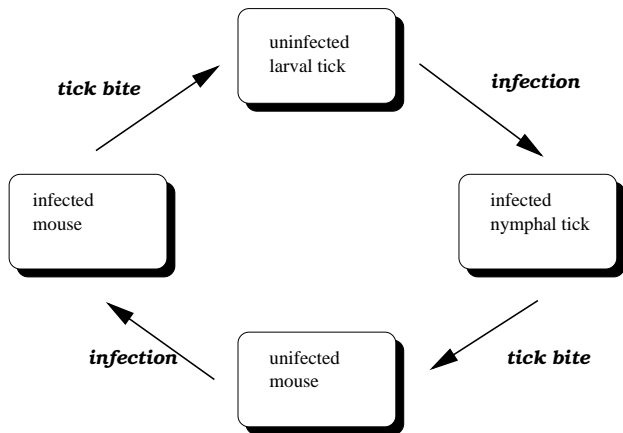
2 Lyme Disease Simulation

Figure 1: The cycle of Lyme Disease

We applied multiparadigm modeling to simulation of the spread of Lyme disease [3] The disease recently has become prevalent in the Northeastern United States [1, 11]. People may acquire the disease after being bitten by a tick infected with the pathogen, a spirochete. The visible incidence of Lyme disease involves humans, but the epidemic is driven by a cycle of infection between ticks and mice. If an infected tick bites a susceptible mouse, the latter becomes infected. The disease then can be transmitted from an infected mouse to the next generation of ticks, competing the natural infection cycle; see Figure 1. The tick life cycle has three stages; they hatch as larvae, transform into nymphs, and mature as adults. Larvae and nymphs prefer feeding on mice. Adult ticks feed on deer, where they mate. The seasonal cycle of infection begins in late spring and lasts 180 days, which sets the simulation duration [4]. Larvae the feed successfully and become infected overwinter; they emerge the next spring as infectious nymphs and the cycle begins again [12].

We previously built a Parallel Discrete Event

Systsm [5], in which mice and deer are modeled as individuals, and space is discretized as a two-dimensional lattice. We treat tick numbers as "background," a feature of each lattice node. Tick numbers are computed independently at each location, so that their densities change discontinuously, even between neighboring locations. We assume that ticks are immobile, but can be carried from one lattice node to another, as a discrete group, by the mobile hosts upon which ticks feed. The simulation ses the optimistic protocol. To reduce the overhead incurred by rollback, it employs Breadth-First Rollback to limit the number of events that need to be rolled back in response to a straggler [5].

2.1 Linking Different Models

A more accurate model requires that tick densities vary continuously in space. since ticks are dispersed by their hosts, the dynamics of tick-stage densities follow a set of partial differential equations. Incorporating a partial differential equation solver directly into the parallel discrete simulation could be extremely difficult, because it changes fundamentally the nature of discrete event simulation; typically, the whole program has to be redesigned.

Alternatively, a separate pde solver can easily be built using either a standard or customized numerical package. Then, the pde solver and the parallel discrete event simulation work collaboratively as a pair of server and client. Since the pde solver is more computationally intensive than the discrete event simulation, it works as a server. To implement communication between pde solver and the parallel discrete event simulation, an adaptor is inserted between the two. Whenever the discrete event simulation needs to access the density of ticks, a request is sent to the pde solver, which can then decide whether or not to respond immediately based on the timestamp of the request.

The Lyme disease application demonstrates an advantage of multiparadigm simulation of biological system. It provides an efficient way of integrating organisms with small body size and huge densities with much large and consequently less densely distributed organisms.

Each of the models can be parallelized by domain decomposition. An interesting property of such parallelization is that each subdomain needs to communicate only with the corresponding subdomain from the other model for linking these two models together. The communication between subdomains of the same model is native to each model simulation (see Figure 2).
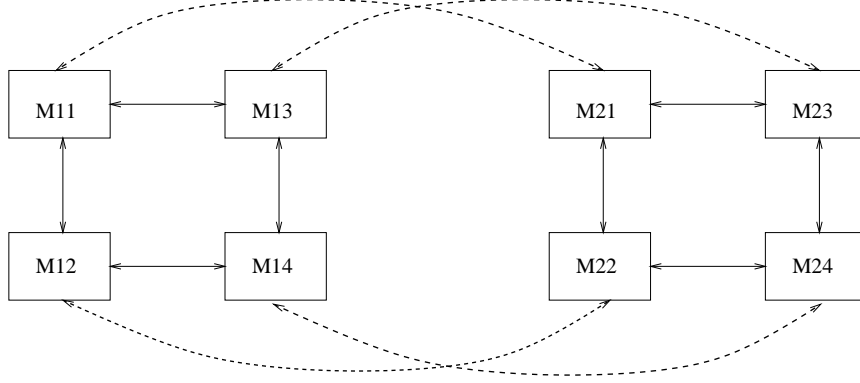
Figure 2: Communication patterns in parallelized multiparadigm simulations. Straight line represent intra-model communications while arches show inter-model communication

2.2   Modeling Ticks

In Lyme disease simulation, five state variables represent five types of ticks as shown in Table 1.

A(x,y,t)	Uninfected adult ticks
a(x,y,t)	Infected adult ticks
N(x,y,t)	Susceptible nymphs
n(x,y,t)	Infectious nymphs
L(x,y,t)	Questing larvae

Table 1: Five types of ticks

To describe the tick population, we use a reaction-diffusion model. A reaction term summarizes spatially localized processes of birth, death, and when applicable, developmental advance and infection transmission. The parameters of the reaction terms are independent of spatial location. Diffusion terms involve the second-order partial derivatives. Diffusion approximates biological dispersal of ticks in response to the overpopulation.

Equations are:

$$\frac{\partial A(x,y,t)}{\partial t} = -\mu_A A + D_A \cdot (\frac{\partial^2 A}{\partial x^2} + \frac{\partial^2 A}{\partial y^2}) \quad (1)$$

$$\frac{\partial a(x,y,t)}{\partial t} = -\mu_A a + D_A \cdot (\frac{\partial^2 a}{\partial x^2} + \frac{\partial^2 a}{\partial y^2}) \quad (2)$$

$$\frac{\partial N(x,y,t)}{\partial t} = -\mu_N N + D_N \cdot (\frac{\partial^2 N}{\partial x^2} + \frac{\partial^2 N}{\partial y^2}) \quad (3)$$

$$\frac{\partial n(x,y,t)}{\partial t} = -\mu_N n + D_N \cdot (\frac{\partial^2 n}{\partial x^2} + \frac{\partial^2 n}{\partial y^2}) \quad (4)$$

$$\frac{\partial L(x,y,t)}{\partial t} = \quad - \quad r(A + a)[1 - c(A + a)]$$
$$- \quad \mu_L L + D_L \cdot (\frac{\partial^2 L}{\partial x^2} + \frac{\partial^2 L}{\partial y^2}) \quad (5)$$

The meaning of parameters is shown in Table 2 It

μ_A, μ_N, μ_L	Mortality rate of ticks
D_A, D_N, D_L	Diffusion coefficient
r	Per capita reproduction
c	Crowding

Table 2: Tick parameters

should be noted that the diffusion equations lack terms for developmental advance and infection. Adults and nymphs only die and diffuse. To represent the flow between stages, we set initial densities of susceptible and infectious nymphs at $t = 0$. At $t = 90$ days, larvae hatches as susceptibles and at $t = 180$ days, they are counted as next year's nymphs.

3   Partial Differential Equation Solver

To solve the above partial differential equations, we chose fully discrete methods which discretizes in both time and space dimensions [8]. In a fully discrete finite difference method, the continuous domain of the equations is replaced by a discrete mesh of points and the derivatives are replaced by finite difference approximations.

Fully discretized method is not the most efficient, but it is well suited for our simulation because it is most amenable to rollback implementation. In the Lyme disease, the entire simulated two-dimensional space is divided into small cells whose size is 20 by 20

meters (this is a home range of a mouse which includes a nest and the foraging area). The ticks are assumed to be dispersed evenly within every cell. If the pde solver discretized space with larger mesh cells than the discrete-event simulator, the larger cells would include discontinuities resulting from the discrete-event simulation. This would make computation by any method difficult. In the fully discretized method we used, convergence to the solution is guaranteed if the time step satisfies the condition: $\Delta t < (h^2/(2c))$, where h is the grid size and c is a coefficient. The local error is of the order $O(\Delta t) + O(h^2)$.

As an example, equation (1) is rewritten as:

$$
\begin{aligned}
A(x,y,t+\Delta t) \;=\;& A(x,y,t) \\
&+\; \Delta t \cdot \{-\mu_A A(x,y,t) \\
&+\; \frac{D_A}{h^2} \cdot [A(x+h,y,t) + A(x-h,y,t) \\
&+\; A(x,y+h,t) + A(x,y-h,t) \\
&-\; 4 \cdot A(x,y,t)]\} \qquad (6)
\end{aligned}
$$

where Δt is the time step and h is the mesh size. The accuracy of the approximate solution depends on Δt and h.

3.1 Rollback of PDE Solver

Main difficulty of linking parallel discrete event simulation that uses an optimistic protocol and the pde solver is a rollback, during which a simulation must be restored to the state representing the simulation time smaller than the current values of the simulation global clock. Linking pde solver to the rolling back parallel discrete event simulation requires that the pde solver must have the corresponding recovery procedure.

Rollback mechanism in a pde solver is also needed when multiple discrete event simulations are linked with one pde solver. For example, in Lyme disease simulation, a pde solver describing tick dynamics can drive several discrete event simulations, each modeling different kinds of species. To achieve both flexibility and extensibility, each discrete event simulation should have no knowledge of others and run independently. Thus, the only way to synchronize them is by forcing a rollback of the pde solver.

The easiest way to implement rollback is saving state variables in the memory. However, pde solvers are very memory-consuming. For large scale system, there might be not enough memory to save even one extra copy of state variables. Saving state variables to disk is very inefficient because of the low I/O speed relative to the speed of the modern processors. Hence, the most efficient solution is to design a rollback algorithm for pde solver that does not require extra memory.

3.2 Reverse PDE Solver

The task of rollback in a pde solver is to calculate $A(x,y,t)$ when given $A(x,y,t+\Delta t)$. One approximate solution could be:

$$
\begin{aligned}
A'(x,y,t) \;=\;& \\
& A(x,y,t+\Delta t) - \Delta t \cdot \{-\mu_A A(x,y,t+\Delta t) \\
+\;& \frac{D_A}{h^2} \cdot [A(x+h,y,t+\Delta t) + A(x-h,y,t+\Delta t) \\
+\;& A(x,y+h,t+\Delta t) + A(x,y-h,t+\Delta t) \\
-\;& 4 \cdot A(x,y,t+\Delta t)]\}. \qquad (7)
\end{aligned}
$$

The value of $A'(x,y,t)$ computed by this equation is not the original value of $A(x,y,t)$ because we replaced all $A(x,y,t)$ terms in the right hand side of equation (6) by terms $A(x,y,t+\Delta t)$. However, if we apply equation (6) again on A'(x,y,t) we obtain:

$$
A'(x,y,t) \to A'(x,y,t+\Delta t).
$$

Since we know the difference between $A(x,y,t+\Delta t)$ and $A'(x,y,t+\Delta t)$, we can refine $A'(x,y,t+\Delta t)$ as follows:

$$
A''(x,y,t) = A'(x,y,t) + [A(x,y,t+\Delta t) - A'(x,y,t+\Delta t)].
$$

Similarly, we can refine $A''(x,y,t)$ iteratively by computing $A(x,y,t)$, with each iteration step essentially as complex (in terms of computation involved) as the computation of $A(x,y,t+\Delta t)$ from terms $A(x,y,t)$. Hence, the cost of the rollback is typically higher than cost of the equivalent forward computation.

4 Fast Rollback for Linear System

To get a good accuracy, reverse algorithm often requires tens of iterations per time step, slowing the progress of the entire simulation. This cost can be avoided if the underlying equations are linear because pde solver can use then a much faster rollback algorithm. We will demonstrate this algorithm on the example of $A(x,y,t)$ in equation (1), but the only property that we use in the demonstration below is linearity of equation (1), so the approach is applicable to any linear partial differential equation set. Suppose that at time $t = a$ an event is executed that changes the solution to $A'(x,y,t)$, such that at $A'(x,y,a) = A(x,y,a) + C(x,y,a)$. After the system progresses to say time $t = b > a$ computing $A'(x,y,b)$, it is notified that this event is being rolled back so this event must be undone. As a result, we need to find the value of $A(x,y,t)$ at time $t = b$, that is, we need to find out $A(x,y,b)$.

Since $A(x,y,t)$ and $A'(x,y,t)$ satisfy equation (1), the function $R(x,y,t) = A(x,y,t) - A'(x,y,t)$ also

Iterations	Reversing without Linear Rollback	Reversing with Linear RollBack	Speedup
5	5.660	3.553	1.59
6	6.643	4.119	1.61
7	7.599	4.715	1.61
8	8.479	5.419	1.56
9	9.441	6.011	1.57
10	10.457	6.439	1.62

Table 3: Execution time of two algorithms

satisfies this equation because it is linear. In more detail, for equation (1) we have

$$\frac{\partial R(x,y,t)}{\partial t}$$
$$= \frac{\partial A(x,y,t) - A'(x,y,t)}{\partial t}$$
$$= -\mu_A A + D_A \cdot (\frac{\partial^2 A}{\partial x^2} + \frac{\partial^2 A}{\partial y^2})$$
$$+ \mu_A A' - D_A \cdot (\frac{\partial^2 A'}{\partial x^2} + \frac{\partial^2 A'}{\partial y^2})$$
$$= -\mu_A R + D_A \cdot [\frac{\partial^2 R}{\partial x^2} + \frac{\partial^2 R}{\partial y^2}]$$

Of course, the initial condition of $R(x,y,t)$ is $-C(x,y,a)$ and it obeys equation(1). Thus, it can be obtained in the following way:

1. first, set the $R(x,y,a) = -C(x,y,a)$, i.e., to the negation of the effects of the event that needs to be undone;

2. next, run the same forward computation as used for equation (1) until $t = b$, obtaining $R(x,y,b)$;

3. finally, compute $A(x,y,b) = A'(x,y,b) + R(x,y,b)$.

Note, that the only equation (5) is nonlinear in our system. There are three state variables in it: $A(x,y,t)$, $a(x,y,t)$ and $L(x,y,t)$. Thus, we can divide the system into two subsystems: one consists of equations (3) and (4), and the other of equations (1), (2) and (5). The fast rollback can be applied to the first subsystem.

The pde solver rollbacks will run faster if we apply the breadth-first rollback [5] and notice that it is not always necessary to run the backward computation on the entire spatial domain. In fact, a discrete event impacts immediately only a limited area and only later it spreads with a limited speed through the space. Hence, knowing the speed of the species involved and the difference between the current simulation time and the rollback time, the affected area can be found and the rollback computation applied only to this subdomain.

The speedup of this fast rollback algorithm is shown in Table 3.

5 Synchronization

Synchronization is a central issue of the multiparadigm simulation. Inappropriately designed synchronization protocol often leads to poor performance, or even compromises the fidelity of the simulation. Multiparadigm simulation has attracted a certain amount of interests, especially in the area of electronic circuit simulation. Peter Frey et al. have proposed two synchronization protocols that are capable of linking an optimistic discrete event simulation with a continuous simulation [9, 10]. However, these two protocols quickly run into difficulty when applied to a spatially explicit simulation in which typically the average number of discrete event occurring during one execution step of continuous simulation is large.

In other words, the discrete event simulation contains a large number of objects each of which undergoes frequent activities. If there are few events occurring during one step of continuous simulation, then it is not necessary to decompose the simulation into different models because the operations associated with processing each discrete event are relatively inexpensive, thus there would be a severe imbalance between the different simulators.

Consider the case in which the number of discrete events is much larger than the minimum number of steps of continuous simulation that can produce sufficient accuracy. If each time processing an event sent from discrete event simulation to continuous simulation requires the continuous simulation clock to be advanced exactly to the timestamp of that event, the amount of computation would become enormous. Therefore, we propose an approximated synchronization protocol where the size of step in continuous simulation is independent of the discrete event simulation (however, it could be fixed or adaptive). When processing a discrete event, an approximated version of the state variables in the continuous simulation is ac-

Number of processors	Execution time before improvement	Execution time after improvement	Speedup
1	1330	835	1.59
2	1082	572	1.89
4	886	420	2.11
6	884	380	2.32

Table 4: Effects of Improvement

cessed. In our implementation, the latest version with a timestamp that is smaller than the timestamp of the discrete event is chosen.

This synchronization protocol doesn't reduce the accuracy significantly because the space domain has already been approximated in all spatially explicit simulations. Neither does it compromise the fidelity of the simulation if and only if the correct order of events is preserved.

5.1 Further Improvement

In our first implementation, we used a somewhat inefficient but simple synchronization scheme: each time the pde solver wants to progress from time t to time $t + \Delta t$, it must wait until all the discrete events earlier than $t + \Delta t$ have been processed.

The improvement becomes possible when we noticed that we always keep two copies of all state variables: one corresponds to time t, the other corresponds to time $t + \Delta t$. The forward computation only changes the state variables at $t + \Delta t$, while the processing of incoming discrete events is based on the state variables at t. Therefore, in the new synchronization scheme, the forward computation begins as soon as the simulation time is advanced to time t. In the meantime, the pde solver still keep receiving the events from the discrete event simulation and processing them. After the forward computation is finished, the forward computation is redone on locations changed by the received events.

The performance of this scheme greatly depends on the number of discrete events that arrive during a time step of the pde solver. If there are too many events so almost all locations have to be recomputed, we cannot benefit from this improvement. Fortunately, the number of events in our discrete simulation during a step of continuous simulation is much smaller than the number of locations. Thus, this scheme improves the performance by a factor of 2, as shown in Table 4.

6 Future Direction
6.1 Multi-DES Model Approach

The modeling approach that we described so far uses an existing parallel discrete event simulation, which models both mice and deer. However, there are no direct interactions between mice and deer. So, it is advantageous to consider representing these two species in separate models. One advantage of such representation is the model extensibility. New species can be easily added to the simulation as long as there is no direct interaction between new and current species. Another advantage of this approach is a possibility for a new parallelization strategy. In the first approach, multiple processors work on different fragments of space. In contrast, in multi-DES model, the different processors can be employed to process different species. As a result, the synchronization between continuous simulation and discrete event simulations changes. In the first approach, two events coming from different processors can be processed simultaneously, because they are independent of each other. However, in the multi-DES approach, an event might be dependent on the event from another processor, if these events happen near each other in space.

The simplest scheme to address this problem is to process only events with smallest timestamp, but this will introduce unnecessary synchronization between processes, thereby defeating the the effects of parallelization. A better solution it to send the event to the pde solver as soon as it enters the event queue on the discrete event simulation side. Although this method increases the communication overhead, the pde solver has the information about future events. Based on such information, the pde solver will know whether it is safe to respond to incoming events from other processors (to a limited extent though, because it is still possible that event will arrive with a large delay that can cause a rollback in the pde solver).

6.2 Mobile Federated Simulation Approach

The reusability issue of simulation programs is becoming more and more critical as the number and quality of available systems increases. Simulations are hard to verify, so the existing programs which have been used and tested for years are more valuable and trusted than the new ones. Therefore, it is beneficial to adopt a federated simulation approach to create
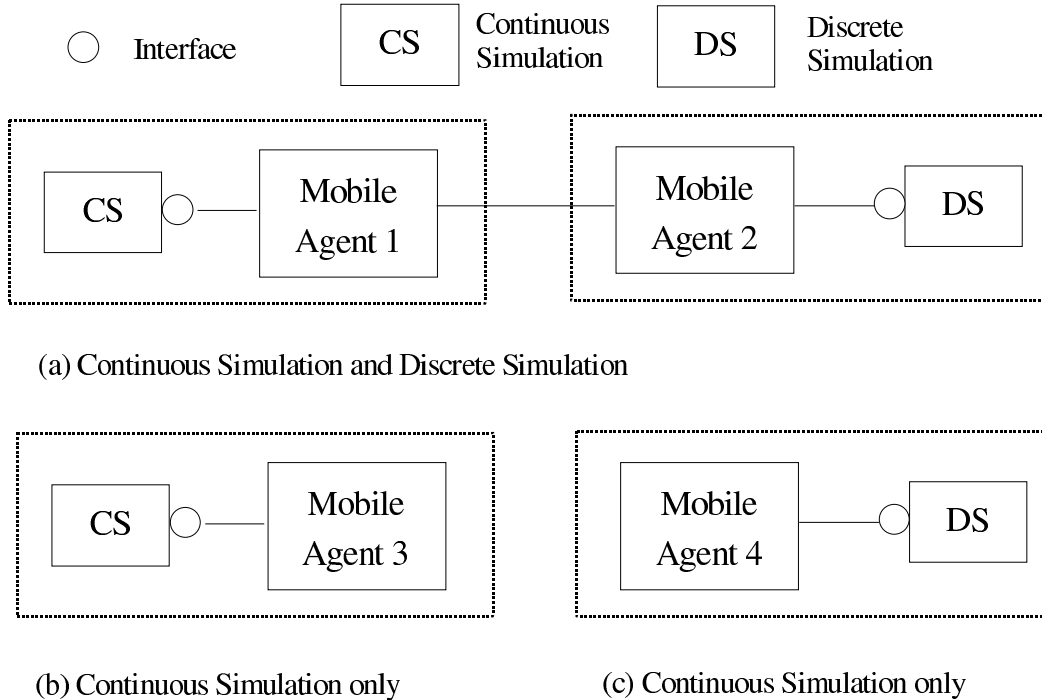
Figure 3: Mobile Federated Simulation Approach

large scale simulation by integrating several component simulations.

We propose a new federated simulation approach that is different from High Level Architecture (HLA). In our approach, every simulation is required to provide a federation interface. For an existing simulation, this interface must be provided by a program that wraps the original simulation. An interface consists of definitions of methods and events. By accessing the interface, the designer of the federated simulation is able to control the component simulations and exchange information with them.

This approach doesn't need to be built on top of CORBA or DCOM because the interface is only allowed to be accessed locally. An interface can be implemented in different ways. For instance, on a shared-memory architecture machine, an interface using direct memory access can be employed instead of the one based on TCP/IP in order to maximize its efficiency.

The interfaces are only open to mobile agents, which are responsible for federating the simulation. Simulation programs are difficult to move freely across heterogeneous platforms because portability of large programs is difficult to attain. Thus, one way to greatly improve the reusability is to utilize mobile agents to interact with component simulation.

In this paper, we illustrate our mobile federated simulation approach on the example of Lyme Disease Simulation, shown in Figure 3. In case (a), the interaction between mice and ticks are of interest so two agents are built to link two simulations. In case (b) and (c), only individual simulation models are studies, requiring a single agent in each case.

This approach demonstrates a more flexible way to federate simulations than HLA. No standards on how to federate are required therefore the mobile agents can choose the best method to accomplish it. In addition, it upgrades existing simulations into simulation servers which would help to build large scale simulation.

7 Conclusions

In this paper we have studied a multiparadigm approach that links a Parallel Discrete Event Simulation with a Continuous Spatial Simulation. We have shown that the rollback of the continuous simulation is necessary. A reversing algorithm for a pde solver proves to be useful if there is no enough memory for a state-saving approach. Also, a fast algorithm that utilizes the property of linear system is described, which is able to speed up Lyme disease simulation by a factor of 1.6. For a pure linear system, this algorithm can be very fast.

To reduce the overhead of linking two models, we

are considering a technique based on collocation of the corresponding parts of the models to the single parallel processor. It should be noted, however, that such collocation is either impossible or difficult to apply if there is no access to the source code of the component submodels. Hence, this method, although potentially more efficient, is also less general than the techniques described in this paper.

Acknowledgments

References

[1] A. Barbour and D. Fish. The biological and social phenomenon of Lyme disease, *Science*, **260**, pp. 1610-1616, 1993.

[2] Wan-Tch Chang, Soonhoi Ha and Edward A. Lee. Heterogeneous Simulation - Mixing Discrete-event Models with Dataflow *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, **15** (1-2), 1997.

[3] Ewa Deelman, Boleslaw Szymanski and Thomas Caraco. Simulating Lyme Disease Using Parallel Discrete Event Simulation, *Proceedings of the Winter Simulation Conference*, 1996.

[4] Ewa Deelman, Thomas Caraco and Boleslaw K. Szymanski. Parallel Discrete Event Simulation of Lyme Disease, *Pacific Biocomputing Conference*, 1996, pp. 191-202.

[5] Ewa Deelman and Boleslaw Szymanski. Breadth-First Rollback in Spatially Explicit Simulations, *Proceedings of the Workshop on Parallel and Distributed Simulation*, 1997.

[6] Richard M. Fujimoto. Parallel Discrete Event Simulation, *Communications of the ACM*, **33** (10) pp. 31-53, 1990.

[7] Gollu, Aleks; Kourjanski, Mikhail. Object-Oriented Design of Automated Highway Simulations Using the SHIFT Programming Language, *Proceedings of the IEEE Conference on Intelligent Transportation Systems*, 1997, Boston, MA.

[8] Michael T. Heath. *Scientific Computing: An Introductory Survey*, The McGraw-Hill Companies Inc., 1996.

[9] Peter Frey, Radharamanna Radhakrishnan, Harold W. Carter, and Philip A. Wilsey. Optimistic Synchronization of Mixed-Mode Simulators, *Proceedings of the International Parallel Processing Symposium, IPPS-1998*, 1998.

[10] Peter Frey, Harold W. Carter, and Philip A. Wilsey. Parallel Synchronization of Continuous Time Discrete Event Simulators, *Proceedings of International Conference on Parallel Processing, ICPP-1997*, 1997.

[11] G.L. Miller, R.B. Craven, R.E. Bailey and T.F. Tsai. The epidemiology of Lyme disease in the United States 1987-1998, *Laboratory Medicine*, **21**, pp. 285-289, 1990.

[12] R.S. Ostfeld, K.R. Hazler and O.M. Cepeda. Temporal and Spatial Dynamics of *Ixodes scapularis* (Acari: Ixodidae) in a rural landscape, *Journal of Medical Entomology*, **33**, pp. 90-95, 1996.

[13] Linda Wilson, George Cybenko, David Lynch, Bruce Cushman-Roisin and Boleslaw K. Szymanski. *KDI: Next-Generation Agent-Based Distributed Simulation* http://www-nml.dartmouth.edu/KDI.

Biography of Presenting Author

Dr. Boleslaw K. Szymanski is an Associate Dean for Information Technology and a Professor of Computer Science. Since joining Rensselaer in 1985, he has been engaged in a multidisciplinary research that focuses on large scale parallel scientific computation, simulation of biological and computer system and in networking. In the past he was also affiliated with the University of Pennsylvania, Aberdeen University (U.K.) and Warsaw Polytechnic (Poland).

Dr. Szymanski received Ph.D. in Computer Science from National Academy of Sciences in Poland in 1976. He is an IEEE Fellow, former ACM National Lecturer, Editor-in-Chief of *Scientific Programming* and a member of editorial board of another journal. Dr. Szymanski co-organized several international conferences. He edited and contributed chapters to three books and authored over 150 research papers in journals and conference proceedings.

His current research focuses on pro-active network management for Next Generation Internet, computational models of evolution, epidemics and biomedical systems, as well as in program optimization for parallel scientific computation. In the past he has been also involved in econometric modeling, algorithm design and information retrieval.