SIMULATING AUTONOMOUS OBJECTS IN A SPATIAL DATABASE USING WAVE

Fehmina Merchant(1), Lubomir F. Bic(1), Peter Borst(2), Malcolm Corbin(3),
Michael B. Dillencourt(1), Munehiro Fukuda(1), Peter Sapaty(3)
(1) University of California, Irvine
(2) University of Karlsruhe
(3) University of Surrey
E-mail: fmerchan@ics.uci.edu

ABSTRACT

WAVE is a system based on the concept of mobile objects propagating asynchronously through a network of interpreters. We demonstrate how this computational paradigm can be used to simulate the movement of physical objects through a 2-dimensional space. Our approach maps the simulated space, consisting of a line segment database that is stored using a PM_1 quadtree, onto a collection of interpreters of the WAVE language. The simulated objects are programs carried on messages that physically migrate through the underlying space and carry along their entire behavior. We describe the implementation of several basic behaviors of such autonomous mobile objects e.g., following a given line segment, moving in a certain direction, or searching for information in the object's vicinity.

INTRODUCTION

Recent simulation efforts have recognized the need to permit the human operator to interact with a simulation while it is in progress, thus controlling the process dynamically as it unfolds. A typical example is the Distributed Interactive Simulation (DIS) initiative (DIS 94; Loral 92), whose objective is to permit the creation of realistic synthetic environments to conduct large-scale interactive battle simulations.

The DIS approach is to establish standards such that large numbers of different computer simulations running on a variety of heterogeneous platforms could collaborate in such large-scale operations. The current standard is based on defining a common unit of discourse, called a Protocol Data Unit (PDU), which encodes the necessary information about possible events. All dynamic objects are then required to inform each other of their movements and other events they cause by broadcasting that information using PDUs.

One of the main problems with this philosophy is that there is no correlation between the simulated space and the underlying distributed network. Each participant simply gets a copy of the entire simulated space (geographical database) and is responsible for maintaining all updates locally. The simulated objects do not migrate through the physical network, only through the local simulated space. Consequently, *logical proximity is unrelated to physical proximity*; e.g., two planes flying in close formation could each be running on machines located on different continents, or vice versa. The unfortunate implication of this scheme is that PDUs from any given object must be broadcast to the entire physical net, even though only a small number of participants in the object's logical vicinity may potentially be interested in this information (Multicasting is being explored by the DIS community but no satisfying solutions have yet been proposed.)

In this paper we describe an approach to distributed interactive simulation that is based on a different philosophy. It maps the synthetic space, which is a line segment database stored using a PM_1 quadtree, onto a collection of interpreters of a special language called WAVE such that simulated objects are programs carried on messages which physically migrate through the underlying space.

THE WAVE MODEL

Basic Principles

WAVE is a novel approach to distributed computing developed at the University of Surrey, UK, and the University of Karlsruhe, Germany (Borst 1992; Sapaty 1988; Sapaty 1991). Any WAVE application may be viewed as a collection of nodes communicating with one another via links. Unlike traditional message-based approaches, however, nodes do not contain any application-specific programs. Instead, each contains an *interpreter* for a special WAVE language. Hence the collection of all interpreters, together with their logical links, represents an abstract computational environment—a logical network—within which problems may be solved using WAVE objects.

Each WAVE object is a program carried by a message. It is a mobile object, which may travel and, if necessary, replicate itself through the underlying network of logical nodes (interpreters). The *mobility* of WAVE objects is implemented by *incremental interpretation* of the programs contained in each object. That is, each node receiving a WAVE object typically interprets only a portion of its program, i.e., a part of the object's behavior. Any such program is a sequence of computational and navigational commands. When the receiving node encounters a navigational command, it causes the program to be propagated to another node along a given link. Hence the interpretation is incremental in both time and space. This is repeated until the given program terminates, causing the corresponding object instance to cease to exist.

In addition to their ability to move autonomously through a network, WAVE objects may also *replicate* themselves in that a navigational command may cause the object, including its program and current state, to be replicated and a separate copy sent to multiple nodes connected to the sender via links. This allows the computation to spread asynchronously into the underlying computational network thus "colonizing" it for its use as a computing resource, without needing any *a priori* knowledge of its size or topology. Thus WAVE objects are similar to viruses or worms, the main distinction being that the latter are compiled while WAVE objects are interpreted and may travel only through nodes containing the WAVE interpreter.

Using WAVE for Distributed Simulation

Due to the objects' inherent mobility in the net, WAVE has been shown to offer a more natural formalism than other approaches to satisfy the requirements of distributed interactive simulations (Bic et al. 1994; Borst et al. 1994). In the simplest case, the simulated space is divided into a grid and each subregion is mapped onto a different physical node. WAVE objects, each representing a distinct simulated object of the application, such as an aircraft or ship, then move through the simulated space according to their predefined behaviors or in response to an operator's directions. Once an object reaches the boundary of a subregion, it is handed off to the appropriate neighboring region. Hence objects move not only in the simulated space but actually *migrate* physically through the network. The important implication is that logical proximity of objects is related to their physical proximity and thus information about a given object does not have to be broadcast to the entire net.

The generic behavior of a simulated object, such as an aircraft, can be captured very succinctly by the following WAVE program skeleton:

```
(1) repeat
(2)   local_computation(...)
(3)   (X,Y) = new_position(node_variables, my_variables)
(4)   if (X,Y) outside of current node
(5)     then forward program to neighbor node
(6) until term_condition(node_variables, my_variables)
```

The object traverses the simulated space by periodically recomputing its coordinates. On each iteration, it performs some local computation in step 2. The object then determines its next position (i.e., the value of the variable (X,Y)) in step 3 by applying the function `new_position` to local variables found in the current station node (`node_variables`) and to variables the object carries along as it travels through the net (`my_variables`). If the next position is in an area covered by a different node, the object is transferred to that node. In this way the object keeps moving through the net until the termination condition is satisfied (step 6), at which time it ceases to exist. Note that `local_computation`, `new_position`, and `term_condition`, the three functions that determine the object's overall behavior, are unspecified. They may be different for each object, thus defining objects with different behaviors. Furthermore, they may not all be known at the time the simulation process is started; rather, the user is able to introduce new object types on the fly by specifying these functions as needed in a given situation.

MOVING WAVE OBJECTS THROUGH SPACE

Representation of 2-D Space

The simulated space is represented as a spatial database, with the lowest-level primitive objects being line segments. More complicated objects (e.g., boundaries between regions, coastlines, roads, railways, waterways) are represented by chains of contiguous line segments. We use a PM_1 quadtree (Hoel and Samet 1991; Samet 1990a; Samet 1990b) to store the spatial database.

In this scheme, the 2-D simulated space is recursively decomposed into four subregions of equal size until either (1) each divided subregion contains at most one line segment, or (2) all line segments within a subregion are all incident on a common vertex lying within the subregion. Figure 1 shows the structure of a PM_1 quadtree representing a simulated space. Each node corresponds to one square region of the total space. The size of each region is determined by its distance from the root. In particular, the root, r, corresponds to the entire space, i.e., $0 \leq X < 40$, $0 \leq Y < 40$. The node q2 then corresponds to the north-east quadrant, i.e., the area $20 \leq X < 40$, $0 \leq Y < 20$ (as indicated by the dashed arrow). Each leaf node represents an area that is not decomposed further because it contains a single vertex or an interior portion of a single line segment. For example, the block $0 \leq X < 10$, $10 \leq Y < 20$ contains the vertex (5,15), while the block $30 \leq X < 40$, $0 \leq Y < 10$ contains a portion of a rail road edge.

Each vertex is represented by two fields called XC and YC that correspond to the X and Y coordinates of the point. An edge is implemented with three fields, S, B, and N where S is the slope of the edge, B is the edge's y-intercept and N is the edge's name. Each node in a PM_1 quadtree has seven fields. The first four fields are directions to the node's four children corresponding to the four directions NW, NE, SW, and SE (labeled h in Figure 1). The fifth and the sixth fields specify the size of the subregion corresponding to the node of the PM_1 quadtree. The seventh field indicates whether the node is a leaf node or a non-leaf node. If a node is a leaf node, then in addition to the seven mentioned fields,

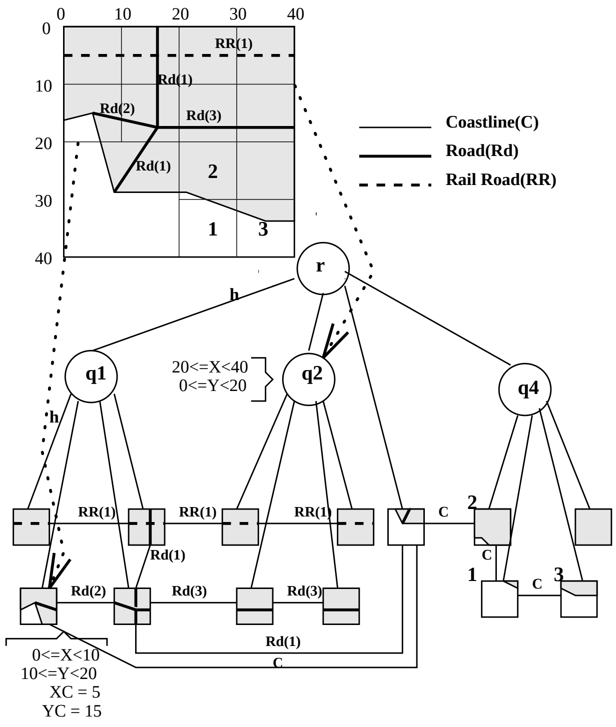

Figure 1: A PM_1 Quadtree

it has information pertaining to an edge or a vertex stored in it. Also, a leaf node is connected by links to other leaf nodes where the line segment(s) stored in its subregion continues (in Figure 1, block 1 is connected to blocks 2 and 3 with C links since the line segment C stored in block 1 continues into blocks 2 and 3).

After the simulated space is partitioned in a PM_1 quadtree, each node of the PM_1 quadtree is mapped to a processor. WAVE objects then move over different subregions by going up and down the quadtree.

Spatial Propagation Primitives

We present three primitives that can be used by WAVE objects to navigate through the simulation space and to search their vicinity for various events.

- *follow_line_segment(L, Term_Cond)*: Follow an existing line segment L until a certain termination condition occurs
- *follow_vector(Direction_Vector, Term_Cond)*: Follow a virtual line segment (i.e., move in a specified direction) until a termination condition occurs
- *search_circle(R, Term_Cond)*: Search vicinity within a radius R until a termination condition occurs

These primitives can be used by WAVE objects to simulate various complex behaviors. For instance, a simulated object, such as a tank, can "look" for some geographical features in its vicinity, e.g., a road junction, using *search_circle* and then move through the simulated space toward it using *follow_vector* or *follow_line_segment*.

Primitive 1 - Follow an Existing Line Segment
This primitive is used by WAVE objects to follow line segments (representing roads, railways, water ways or boundaries of areas) that are present in the simulated space.

```
follow_line_segment(L,Term_Cond)
/* Assume that the object O is at a data block B which
   contains one of the endpoints of the line segment L */
(1) repeat
(2)   local_computation(...)
(3)   (X,Y) = new_position(L,...)
(4)   if (X,Y) outside of current node
(5)     forward program along link L
(6) until term_condition(Term_Cond,...)
```

The program listed above enables WAVE objects to move along line segment L until some criterion is satisfied. Assuming that the object is initially at the node containing one of the end points of the line segment L (a leaf node of the PM_1 quadtree), it repeatedly determines where to go next in step 3. If the new coordinates are outside the current node and the object is moving along line segment L, the object is transferred along link L to the node where the line segment L continues (step 4–5). This process is repeated until an appropriate condition (for instance, certain distance covered or an obstacle encountered), computed by the function `term_condition` is satisfied. The object may then change its behavior (for instance, it might change direction).

Figure 2 illustrates how a WAVE object moves along road Rd(3) for a distance of 20 units using the PM_1 quadtree. Assume that the object is at one of the end point (16,18) of Rd(3) (labeled block 1). It periodically recomputes its new position. When the object's position is outside the current block's (leaf node) space, it propagates along link Rd(3) to block 2 where the road Rd(3) continues. This process is repeated until the object reaches block 3 and has traveled 20 units.

Primitive 2 - Follow a Virtual Line Segment
This primitive allows WAVE objects to move in a certain direction in the simulated space. The WAVE objects, while moving in a particular direction, may check for a certain condition along the way and modify their behavior accordingly.

```
follow_vector(Direction_Vector,Term_Cond)
(1)  repeat
(2)    local_computation(...)
(3)    (X,Y) = new_position(Direction_Vector,...)
(4)    if (X,Y) outside of current node
(5)      repeat
(6)        replicate program up link h
(7)      until ((X,Y) inside current node)
(8)      repeat
(9)        if ((X,Y) inside current node)
(10)         replicate program down link h
```

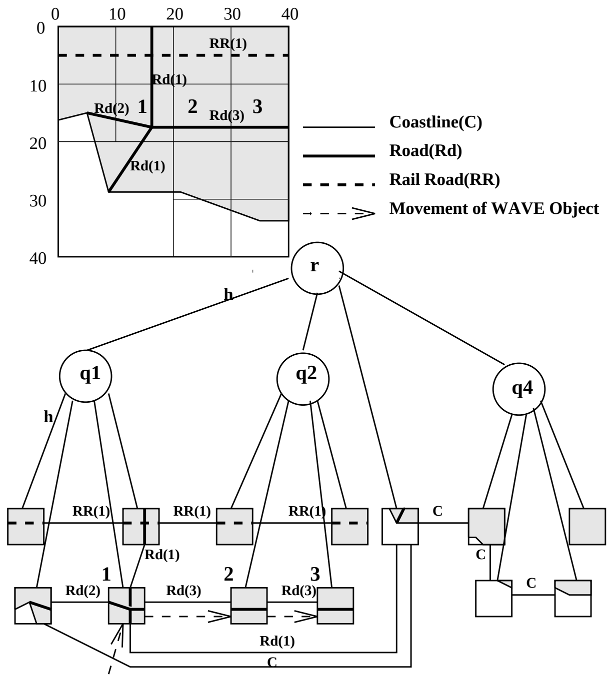

Figure 2: Primitive 1 - Follow an Existing Line Segment

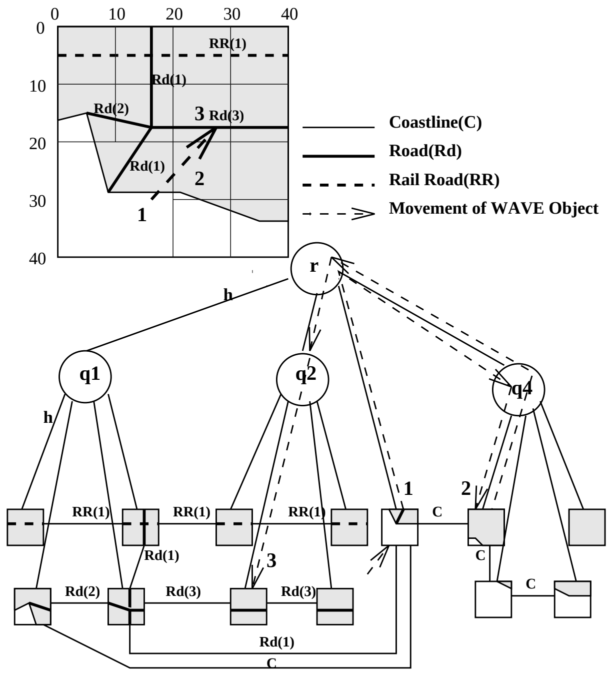

Figure 3: Primitive 2 - Follow a Virtual Line Segment

```
(11)      until (leaf node reached)
(12) until term_condition(Term_Cond,...)
```

The WAVE object periodically recomputes its new coordinates (step 3). If the new coordinates (X,Y) are outside the current node, the object moves up the PM_1 quadtree until it reaches a node that has the new coordinates in its space (steps 5-7). Then, the object moves down the PM_1 quadtree until it reaches the leaf node containing the new coordinates (steps 8-11). In this way the object keeps moving through the simulation space in a particular direction until some condition, computed by function `term_condition`, is satisfied, at which time it can change its behavior by, for instance, changing direction or "looking" for other objects.

Figure 3 illustrates how an object at point P(17,30) (block 1) moves along a line with the slope of -3/2 in the +x direction looking for Rd(3). The object periodically recomputes its position until its new coordinates are in an area covered by block 2. The object then traverses to block 2 by moving up the PM_1 quadtree until it reaches node r and then moving down the quadtree until it reaches block 2. This process is repeated until the object reaches block 3 and finds Rd(3).

Primitive 3 - Search Vicinity
This primitive is used by WAVE objects to search a circular region around them for some information. This may be used to implement *"active sensing"*; that is, instead of each object having to constantly broadcast information about itself—the PDU concept—objects can actively "look" for other object, events, or geographical features, by sending out waves of special sensing objects, which spread into the object's vicinity and report to it information they have been programmed to detect.

```
search_circle(R, Search_Criteria)
/* Object is at point P and information in the circular
   region of radius R with its center at P needs to be
   gathered*/
(1)  repeat
(2)    replicate program up and down link h
(3)    if current node encloses circle(P,R)
(4)      replicate program down link h
(5)    else if current node doesn't intersect circle
(6)      failure
(7)  until (leaf node reached)
(8)  if (leaf node satisfies Search_Criteria)
(9)    gather_information(...)
(10)   repeat
(11)     replicate program up link h
(12)   until (point P inside current node)
(13)   repeat
(14)     replicate program down link h
(15)     if (point P not inside current node)
(16)       failure
(17)   until (leaf node reached)
(18) else failure
```

The WAVE object is at point P and wants to "look" for a certain event or feature in the circular region of radius R around it. Waves are first propagated along all the h links of the PM_1 quadtree (step 2). If a node is reached whose subregion encloses the circle, waves are then propagated only to the node's children (steps 3–4). This process is repeated until waves reach all the data blocks (leaf nodes) belonging to the circular

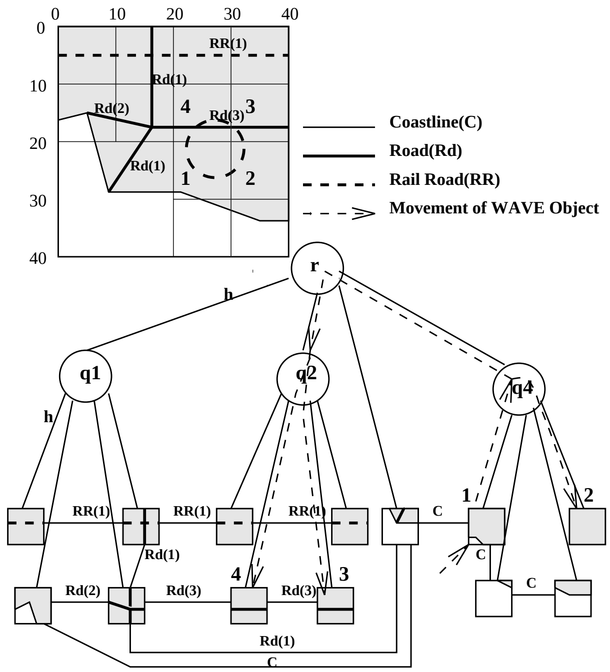

Figure 4: Primitive 3 - Search Vicinity

region (steps 1–7). Waves on arriving at the leaf nodes then determine if the data that is being "looked" for is present in their subregions (`Search_Criteria`). If this is the case, waves carry the relevant information back to the block containing point P (steps 8–18).

Figure 4 illustrates how an object O at point P(28,21) sends waves in its vicinity (circular region with radius of 5 units around it). First, the object O propagates a wave along the h link of the PM_1 quadtree to node q4. Wave on arrival at node q4 determines that the circle to be isolated is not completely enclosed by the node's space and, as a result, sends waves to its parent and children nodes (parent labeled r). Since the r (root) node's subregion completely encloses the circle to be isolated, waves are now propagated only to the root's three children (excluding the child from which the wave was received). Waves on arriving at the leaf nodes (representing subregions intersected by the circular region) then determine if the desired objects are in their subregion and report to object O accordingly.

CONCLUDING REMARKS

We have described an approach to simulation using active objects that are capable of propagating asynchronously through the simulated space. In our scheme the 2-dimensional simulated space, which is a collection of line segments (approximating various geographical entities) stored using a PM_1 quadtree, is mapped onto a number of interpreters of the WAVE language. Simulated objects are then programs written in WAVE, that are carried on messages that physically migrate through the simulated space. We have described three basic primitives that objects can use to move and to gather information about their surroundings: (1) follow an existing line segment, (2) move in a given direction, and (3) search a circular region.

There are two main advantages of this approach, resulting from the underlying philosophy of WAVE. First, there is no centralized control for managing the various simulated objects. Each is a self-contained autonomous entity, capable of moving through the simulated space and making its own decisions based on its inherent behavior and information gathered dynamically, i.e., without any external intervention. This, in turn, supports scalability in the number of potential simulation objects. The second major point is that the resulting system is open-ended in that the functionalities of the simulated objects are not coded as part of any programs stored in the network nodes. These only contain the universal WAVE interpreter and hence it is possible to create and inject objects with new behaviors while the simulation is in progress. Thus the scope of the new functionalities is not bounded by the number of precoded message types but by the expressive power of the interpretive language.

REFERENCES

Bic, L.; P. Borst; M. Corbin; and P. Sapaty. The WAVE Control Protocol for Distributed Interactive Simulation. *11th Workshop on Standards for the Interoperability of Distributed Simulation*, September 1994.

Borst, P.; M. Corbin; and P. Sapaty. WAVE Processing of Networks and Distributed Simulation. *High-Performance Distributed Computing (HPDC-3)*, August 1994.

Borst, P. The First Implementation of the WAVE Systems for UNIX and TCP/IP Computer Networks. *Technical Report 18/92*, December 1992.

Loral Systems Company. Strawman Distributed Interactive Simulation Architecture . *ADST/WDL/TR-92003010*, 1 and 2, 1992.

DIS Steering Committee. The DIS Vision: A Map to the Future of Distributed Simulation. *Institute for Simulation and Training*, 1994.

Hoel, E. and H. Samet. Efficient processing of spatial queries in line segment databases. *Symposium on the Design and Implementation of Large Spatial Databases (SSD)*, pages 235–256, August 1991.

Samet, H. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, Reading, MA, 1990.

Samet, H. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

Sapaty, P. WAVE-1: A New Ideology of Parallel and Distributed Processing on Graphs and Networks. *Future Generations Computer Systems*, 4(1), 1988.

Sapaty, P. Logic Flow in Active Data. In *VLSI for AI and Neural Networks*, J.G. Delgado-Frias and W.R. Moore, eds., Boston, Mass., 1991.