ADAPTIVE DEAD RECKONING ALGORITHMS FOR DISTRIBUTED INTERACTIVE SIMULATION

BU-SUNG LEE WENTONG CAI STEPHEN J. TURNER L. CHEN

School of Computer Engineering
Nanyang Technological University
Singapore 639798

Abstract: This paper describes two adaptive algorithms for dead reckoning in Distributed Interactive Simulation (DIS). The first algorithm is based on the control mechanism of *adaptive adjustment of threshold level* and the second on *automatic selection of extrapolation equation*. Since a fixed threshold cannot adequately handle the dynamic relationships between moving entities, a multi-level threshold scheme is proposed. The definition of threshold levels is based on the concepts of *area of interest* and *sensitive region*, and the levels of threshold are adaptively adjusted based on the relative distance between entities during the simulation. During the simulation, the motion of an entity may also change from time to time. To optimize the performance, the second adaptive algorithm uses a control mechanism that automatically selects an appropriate extrapolation formula according to the entity's motion. Various experiments were conducted. The results show that the proposed adaptive dead reckoning algorithms can achieve a considerable reduction in the number of update packets while maintaining adequate accuracy in extrapolation.

Keywords: Distributed Interactive Simulation (DIS), Dead Reckoning, Adaptive Algorithms.

1. INTRODUCTION

Distributed Interactive Simulation (DIS) is a technology for linking simulations of various types at multiple locations to create a realistic, complex, "virtual world" for the simulation of highly interactive activities [DIS, 1994]. The High Level Architecture (HLA) [DoD, 1998] is a general purpose architecture designed to promote simulation reuse and interoperability. This is achieved through the HLA concept of the federation: a composable set of interacting simulations. Although the DIS standard [IEEE 1278-1993] has been supplanted by the HLA, the DIS community has developed the real-time platform reference federation object model (RPR FOM) [Shanks, 1997] to provide the functionality of the DIS standard within the HLA environment.

Since simulation entities are physically distributed in DIS, for a large scale DIS exercise, updating states of the simulation entities may generate a large amount of communication and thus saturate network bandwidth. To reduce the amount of communication, the *dead reckoning* (DR) technique, a fundamental feature of the DIS standard [IEEE 1278-1993], was developed. Using a dead reckoning model, the position of a simulation entity will be extrapolated. So, instead of emitting a state update packet after each movement of the entity, update packets are only transmitted when the difference between the extrapolated and the true position exceeds a predefined threshold [Pope, 1991].

Previous research work on dead reckoning have been largely focused on the evaluation of extrapolation equations [Foster and Maassel, 1994; Lin, 1994a; Lin, 1994b], and the performance investigation of DR mechanisms (e.g., [Durbach and Fourneau, 1998; Figart et al, 1996]). In their study, for a given simulation entity the extrapolation equation is usually unchanged during the experiments. In addition, the threshold value, used as a parameter in DR algorithms, is also fixed. In this paper, we introduce two adaptive dead reckoning algorithms that reduce the number of state update packets without sacrificing extrapolation accuracy. The performance of the algorithms will also be studied.

In general, dead reckoning algorithms use a fixed threshold, regardless of the relationship between the entities, to control errors in extrapolation. In order to maintain an adequate accuracy, a small threshold is usually used. However, since the exact position is not important for entities that are far away from each other, unnecessary update packets may be generated. To reduce the number of update packets while maintaining adequate accuracy, in the first adaptive

A shorter version of this paper appears in Proceedings of 13th Workshop on Parallel and Distributed Simulation (published by IEEE Computer Society), May 1999, Atlanta, Georgia, USA.

dead reckoning algorithm, the threshold is dynamically changed according to the relative distances between simulation entities. It is also evident that the performance of extrapolation equations depends on the motion made by a simulation entity as well as the threshold value used. To further optimize the performance, the second adaptive dead reckoning algorithm automatically selects an appropriate extrapolation equation according to the current motion of the simulation entity.

Our work is different from that in the area of relevance filtering (e.g., [Bassiouni et al 1997; Bassiouni et al, 1998; Morse, 1996; Rak et al, 1996]), although we have the same goal, that is, to reduce the traffic on the network and to improve the scalability of DIS systems. Relevance filtering is concerned with eliminating the transmission of irrelevant packets, whereas, our work focuses on reducing the number of state update packets caused by the dead reckoning mechanism.

This paper is arranged as follows: Section 2 will give a brief introduction on dead reckoning. The adaptive multi-level threshold algorithm and its performance will be covered in Section 3. The algorithm based on automatic selection of extrapolation equation according to an entity's motion will be explained in Section 4. Finally, Section 5 will conclude the paper and outline our future works.

2. DEAD RECKONING

One of the important aspects in DIS is the ability of each simulator to represent accurately in real-time the state of all simulation entities, including both local and remote, participating in the same DIS exercise. To reduce the number of state update packets, the DR technique is used. In addition to the *high fidelity model* that maintains the accurate position about its own simulation entities, each simulator also has a *dead reckoning model* that estimates the position of all simulation entities (both local and remote). The anticipated position of an entity is usually calculated based on the last (or past) accurate state information of the entity using an *extrapolation equation* (see Table 1). So, instead of transmitting state update packets, the estimated position of a remote simulation entity is readily available through a simple, local computation.

To maintain accuracy, after each update of its own simulation entity, a simulator needs to compare the true position of the entity obtained from the high fidelity model and its extrapolated position. If the difference between the true and the extrapolated position is greater than a pre-defined *threshold*, a state update packet will need to be sent to other simulators. Extrapolation for the entity will then be corrected by all dead reckoning models at all simulators, based on the updated state of the entity.

Current extrapolation equations can be divided into two groups: one-step formulas and multi-step formulas [Lin, 1995]. One-step formulas only use the last state update packet to extrapolate an entity's position, whereas, multi-step formulas use the last two or more state update packets in the extrapolation.

Conventionally, in DIS, first order extrapolation is used for orientation, and second order for position [Katz, 1994]. For easy comparison, only one-step and two-step (both first and second order) extrapolation formulas will be studied in this paper. They are listed in Table 1. For simplicity, only one dimension of an entity's motion is considered in the examples throughout this paper. However, the algorithms discussed in the paper can also be applied to multi-dimension motion.

	One-Step	*Two-Step*
1st order	$x_t = x_{t'} + v_{t'}\tau$	$x_t = x_{t'} + \frac{x_{t'} - x_{t''}}{t'-t''}\tau$
2nd order	$x_t = x_{t'} + v_{t'}\tau + 0.5a_{t'}\tau^2$	$x_t = x_{t'} + v_{t'}\tau + 0.5\frac{v_{t'} - v_{t''}}{t'-t''}\tau^2$

Table 1: Extrapolation Equations

In Table 1, $x_{t'}, v_{t'}$ and $a_{t'}$ represent respectively the position, velocity and acceleration of the entity as found in the last state update packet. Similarly $x_{t''}, v_{t''}$ and $a_{t''}$ are the position, velocity and acceleration of the second last state update packet. τ is the elapsed time from the last update. The formulas are used to extrapolate the position of the entity at time $t = t' + \tau$.

A multi-step formula needs less data from the update packet than the one-step formula of the same order. But, in multi-step formulas, high-order derivatives, such as velocity (or acceleration), are calculated from positions (or velocities). They are not as accurate as that used in one-step formulas. Therefore, to track the movement of the same entity, using multi-step formulas may introduce more errors and generate more update packets than using one-step formulas of the same order for a given threshold.

Threshold is an important parameter in the DR algorithm. It is used to control the accuracy of extrapolation, and affects the number of entity state update packets generated. A small threshold makes the DR algorithm generate state update packets at a higher frequency, but results in higher accuracy in the estimation of the entity's position. On the other hand, the DR algorithm using a large threshold generates fewer update packets, but its accuracy is also lower.

3. ADAPTIVE MULTI-LEVEL THRESHOLD

Generally, DR algorithms are implemented with a fixed threshold. Although it is easy for the simulator to operate the DR model, a fixed threshold may not adequately handle the dynamic relationship between entities. In fact, the value of the threshold is relevant to the distance between entities. In DIS, each entity, like the real object it simulates, may have different interests in other entities around it. For example, when two entities are close to each other, each of them may need to pay more attention to the other's movement. In this case, a small threshold should be used because the accuracy of extrapolation is important. However, when two entities are far away from each other, the position of one entity may become irrelevant to the other entity. Thus, a large threshold will be sufficient under this situation.

3.1. The Algorithm

To determine the levels of threshold, concepts in *relevance filtering* [Morse, 1996] are used. In relevance filtering, *area of interest* (AOI) of an entity, also called *reachability region* in [Bassiouni, 1998], is defined as a circle with a constant radius around the entity. The length of radius is usually defined according to the entity type. Hence, the AOI of all entities (either local or remote) can be easily determined by a simulator. In addition to AOI, an entity also has its *sensitive region* (SR). If one entity moves into another entity's SR, a collision will likely happen.

Level	4	3	2	1
Description	no overlap of AOIs	overlap of AOIs with another entity	in another entity's AOI	in another entity's SR

Table 2: Threshold Levels

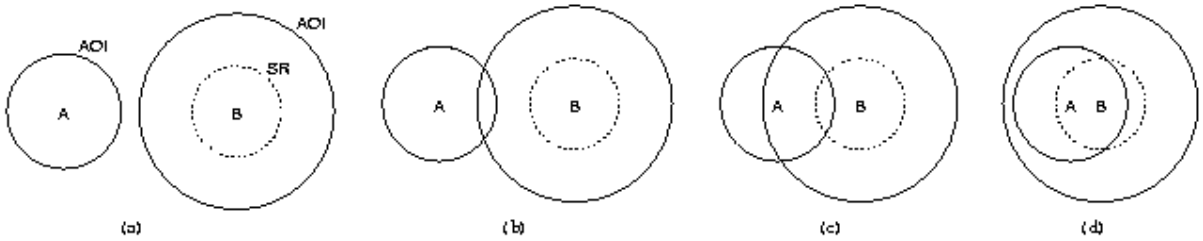

Figure 1: Multi-level Threshold Illustration

By using AOI, SR and the distance between entities, four threshold levels can be defined, with level 1 being the smallest threshold and level 4 the largest. They are listed in Table 2 and illustrated in Figure 1. In the figure, the circle with a solid line represents the entity's AOI, and the circle with a dotted line represents the entity's SR.

When an entity A's AOI is not overlapped with any other entity's AOI (part (a) of Figure 1), the level 4 threshold will be used in entity A's extrapolation. Large errors are allowed in the extrapolation, and update packets will be sent at a low frequency. In this case, entity A's movement is not interesting to other entities, and extrapolation can be less accurate.

When entity A's AOI is overlapped with entity B's AOI, the extrapolation of entity A's movement needs to be more accurate. There are three cases:

- If entity A is in entity B's SR (part (d) of Figure 1), to prevent entity B from making a misjudgment on collision, level 1 threshold (that is, the smallest threshold) has to be used in entity A's extrapolation. In this case, entity A's update packet will be emitted most frequently, but its extrapolation will be the most accurate.

- If entity A is outside entity B's SR but within its AOI (part (c) of Figure 1), level 2 threshold will be adopted. To avoid missing the detection of an approaching entity, entity B needs to have a more accurate position of entity A. However, there is no danger of making a misjudgment on collision. Therefore, a small threshold will be adequate in entity A's extrapolation.

- If entity A is outside entity B's AOI (part (b) of Figure 1), level 3 threshold will be used. Since the two AOIs are overlapped, one entity may

move to another's AOI in a short period of time. The extrapolation of A's position still needs to be accurate, but the requirement is less rigid. Entity A's update packets do not need to be sent frequently because A is not in B's AOI. So, a relatively large threshold can be used in entity A's extrapolation.

In all these cases, extrapolation of A needs to be accurate, though the degree of accuracy required is different.

Threshold values at different threshold levels may be determined by factors in the simulation environment or entity characteristics. For example, the value of level 4 threshold could be limited by an entity's AOI, and the value of level 1 threshold may be defined by the *collision distance* between entities. In a DIS exercise, any entity's movement is restricted by the network delay and the time spent by the simulator to process simulation events. For this reason, judgment of collision of simulation entities is not like that of real objects. Generally, if the distance between two entities is less than a pre-defined collision distance, a collision is considered to have happened in the simulation.

For each local simulation entity e that belongs to a simulator *i*, these four levels of threshold will be used in the extrapolation. If an error is detected at a certain threshold level *k*, an update packet will be sent only to the simulators with a remote entity re whose threshold level is *k* or less. An update packet will be sent to all other simulators only when an error is detected at the threshold level 4. In this way, update packets will be sent only to the relevant entities, and the amount of communication will be reduced.

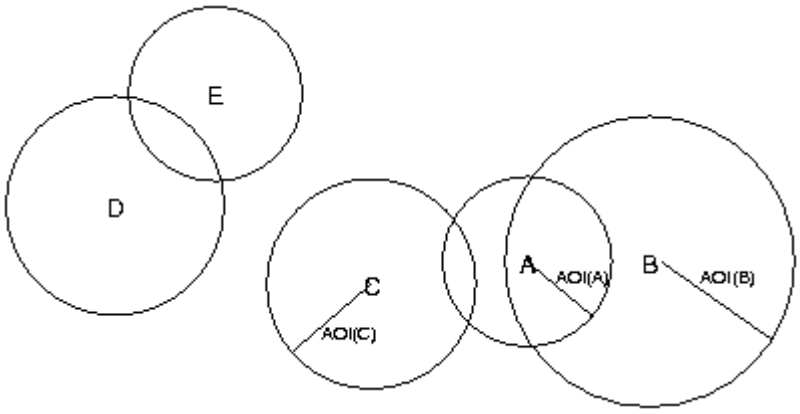

Figure 2: Multi-level Threshold Example

```
/* for simulator i */
/* assume that ∀ e and re, TLevel[e,re] = 4 initially */
for every received packet of remote entity re do
    for each local entity e ∈ simulator i do {
        Dist = distance(e ,re);
        if Dist < AOI(e) + AOI(re) then
            if Dist < SR (re) then
                TLevel[e,re] = 1;
            else
                if Dist < AOI(re) then
                    TLevel[e,re] = 2;
                else
                    TLevel[e,re] = 3;
        else
            TLevel[e,re] = 4;
    }
```

Figure 3: Algorithm to Determine Threshold Level

For a local entity e and a remote entity re, TLevel[e,re] gives the threshold level used to decide when to send re e's update packet. It determines how often re may receive e's update packets. Different remote entities may have different threshold levels. For example, in Figure 2, different threshold levels will be used in entity A's extrapolation. Update packets will be sent only to entity B when an extrapolation error happens at threshold level 2. If an extrapolation error occurs at threshold level 3, an update packet will then be sent to both entities B and C. An update packet will be sent to all other entities (that is, entities B, C, D and E) when an error happens at threshold level 4.

Our adaptive, multi-level dead reckoning algorithm, therefore, consists of two parts:

- For each remote entity re, to determine its threshold level when its update packet is received; and
- For each local entity e, to determine the sending of its update packet when its state changes.

The first part of the algorithm is given in Figure 3, where function *AOI*(e) returns the radius of AOI of entity e, and function *SR*(e) returns the radius of SR. The second part of the algorithm is given in Figure 4.

```
/* for simulator i */
for each state update of local entity e do {
    extrapolate e's position based on the past state information;
    /* send update packet if necessary */
    switch Diff = | TruePosition - ExtrapolatedPosition | {
        case Diff > LevelOneThreshold
            multicast an update packet to the group
                {simulator k | re ∈ simulator k ∧ TLevel[e,re] = 1};
            break;
        case Diff > LevelTwoThreshold
            multicast an update packet to the group
                {simulator k | re ∈ simulator k ∧ TLevel[e,re] <= 2};
            break;
        case Diff > LevelThreeThreshold
            multicast an update packet to the group
                {simulator k | re ∈ simulator k ∧ TLevel[e,re] <= 3};
            break;
        case Diff > LevelFourThreshold
            broadcast an update packet to all other simulators;
            break;
        default
            break;
    }
}
```

Figure 4: Algorithm to Send Update Packets

3.2. Experimental Results

To test the algorithm, two experiments were conducted. In both experiments, a simulated environment was created and a program was written to simulate what should happen in a distributed environment. Statistics, such as total number of update packets generated and average extrapolation accuracy, were collected during the simulation.

3.2.1. Experiment One

In the first experiment, there are two entities: one is a *motionless entity* (ME) at position (110, 0); the other entity (referred to as a *test entity*, TE) is moving along a circle with center (0, 0) and radius 100. The radiuses of ME's SR and AOI are 10 and 50 respectively. The level 1, 2, 3 and 4 threshold values of TE are 5, 10, 20 and 40 respectively. The collision distance is assumed to be 5. The unit used for distance in the experiment is meter.

Figure 5 shows the trajectory of TE's movement (that is, the smooth sinusoid curve) and its extrapolation in the *x* dimension when the fixed threshold values are used. Note that the trajectory of ME is the straight line at 110. The number of update packets (that is, DRNUM) is also shown in the figure. When the threshold is 5 (Figure 5(a)), the extrapolation is sufficiently accurate for ME to make correct collision decisions. When the threshold is 10 (Figure 5(b)), since the extrapolation is less accurate, the ME may make some wrong collision judgments as TE is moving close to it. However, fewer update

packets are generated in this case (14 compared to 19).

Figure 6 shows the trajectory of TE's movement when the threshold value is adjusted adaptively. It can be seen that when TE is approaching ME, the extrapolation error becomes smaller and smaller. Therefore, the extrapolation of TE at the closest point to ME is accurate enough for ME to make the correct judgment. In addition, since a large threshold will be used when TE is far away from ME, the number of update packets generated is also low. In fact, the least number of update packets (that is, 13) is generated in this case. So, by adaptively adjusting threshold values, we can achieve the required accuracy as in the case shown in Figure 5(a) as well as the low generation rate of update packets as in the case shown in Figure 5(b).

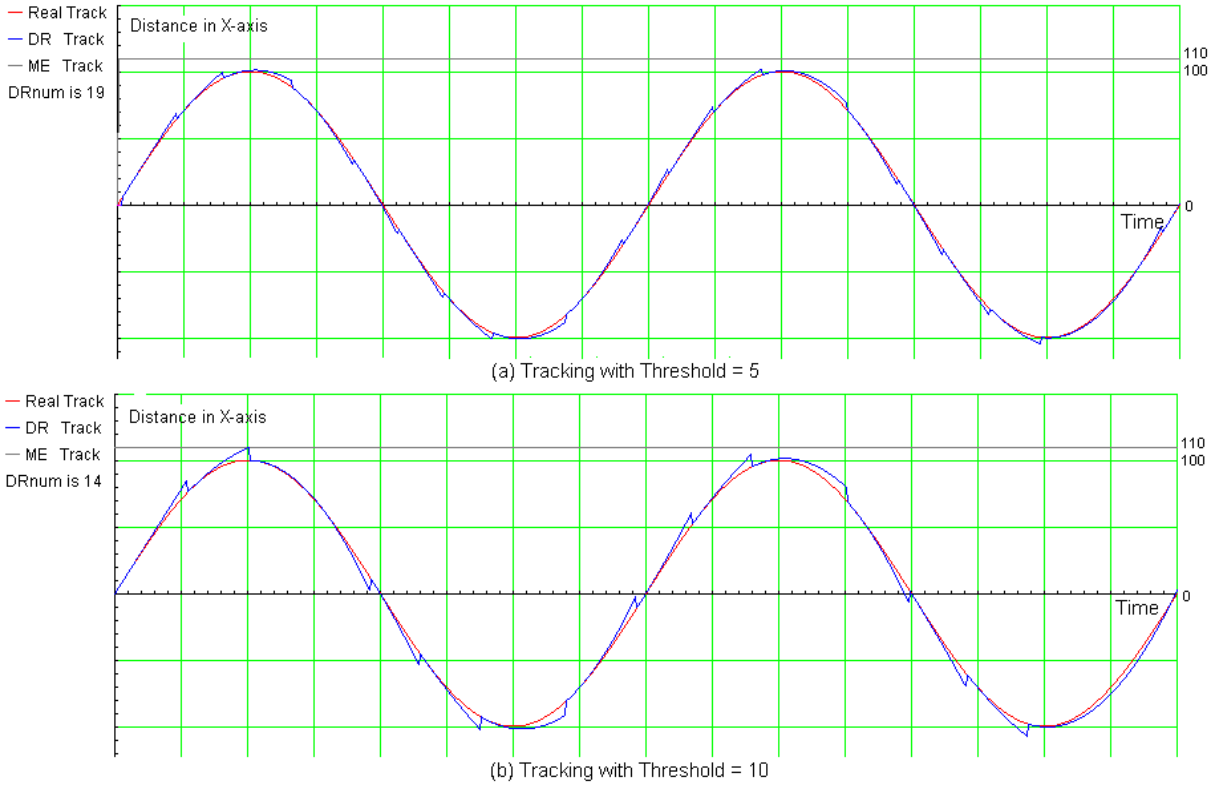

Figure 5: Extrapolation with Constant Threshold

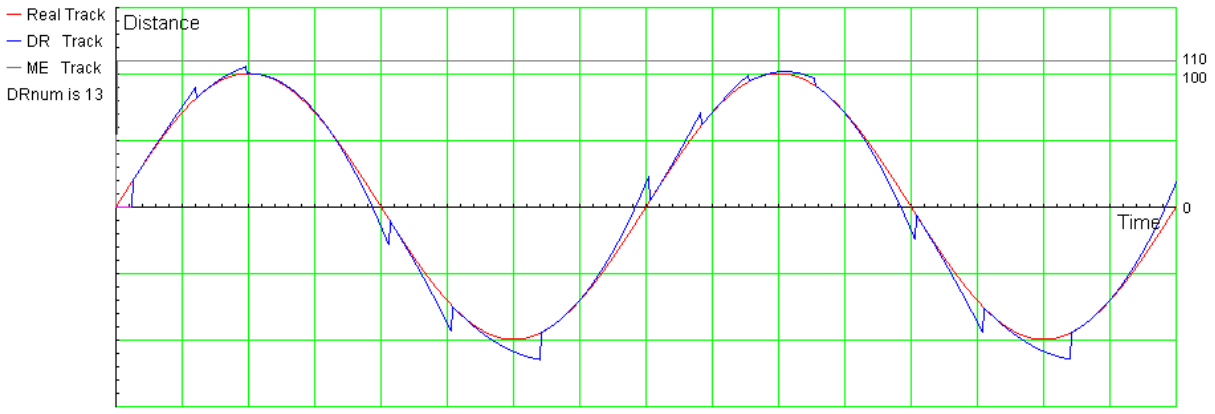

Figure 6: Extrapolation with Multi-level Threshold

3.2.2. Experiment Two

In the second experiment, an entity may move randomly in a 500*m* × 700*m* two dimensional space, and is represented by a circle with radius of 2.5*m*. For this simple experiment, we assume that all entities have the same AOI and SR. The radiuses of an entity's AOI and SR are defined as 32*m* and 10*m* respectively. The number of entities in the experiment varies from 16 to 32, and each entity is simulated by a simulator. The position of each entity is updated every 5 *sec*. At each update, the speed and direction of each entity may change randomly. The maximum speed is 3 *m*/*sec*, and the maximum acceleration is 0.5. The simulation duration is 10 minutes.

```
SumOfError1 = 0;
NumUpdates = 0;
/* (a) calculate the sum of average errors for all updates */
for each position update do {
    NumEntities = 0;
    SumOfError2 = 0;
    /* (b) calculate the sum of average errors for all entities in each update */
    for each entity e do {
        SumOfError3 = 0;
        /* (c) calculate the sum of errors relative to an entity e */
        Se = { e' | e' real position in e's AOI};
        for each entity e' in Se do
            SumOfError3 = SumofError3 + the difference between
                the real position of e' and its extrapolated position;
        SumOfError2 = SumOfError2 + SumOfError3 / SizeOf(Se)
        NumEntities++;
    }
    SumOfError1 = SumOfError1 + SumOfError2 / NumEntities;
    NumUpdates++;
}
/* (d) calculate the average error in AOI */
AvgErrorInAOI = SumOfError1 / NumUpdates;
```

Figure 7: Procedure to Calculate the Average Error in AOI

Threshold	2	8	12	18	24
# of PDUs	4844	3380	3079	2779	2624
Avg Error in AOI	0.5608	2.6838	3.5178	5.5289	7.5277
Avg Error in SR	0.5457	2.1426	3.0143	5.4567	7.1402

16 entities in the simulation

Threshold	2	8	12	18	24
# of PDUs	9705	6796	6203	5638	5230
Avg Error in AOI	0.5645	2.4171	3.0259	5.6765	8.6891
Avg Error in SR	0.5573	2.4637	3.6056	5.1140	6.7640

32 entities in the simulation

Table 3: Fixed Threshold Results

In our experiments, the extrapolation accuracy is defined by *average error in AOI* and *average error in SR*. The average error in AOI is calculated using the procedure shown in Figure 7. For each position update, the average error between the real and the extrapolated positions is calculated. For each entity e, we are only interested in those entities whose real position is in e's AOI (that is, the set S_e). The average error in SR can be calculated similarly if SR is used instead of AOI in S_e's definition.

Table 3 shows the number of PDUs (Protocol Data Units) generated and the average error in AOI and SR when a fixed threshold is used in the dead reckoning algorithm. (The unit of threshold is meter.) The following are some observations obtained from these numbers:

- As the threshold used increases, fewer PDUs will be generated, but the average error in AOI (and SR) increases.

- In most of the cases, there is no large difference between the average error in AOI and the average error in SR, since the fixed threshold is used.

- Except when the smallest threshold (i.e., 2) is used, for all other cases the average errors are large, compared to the entity's size.

# of Entities	16	32
# of PDUs	3461	7875
Avg Error in AOI	1.0130	1.4012
Avg Error in SR	0.4990	0.5414

Table 4: Multi-Level Threshold Results

Threshold Level	4	3	2	1
Value (in Meters)	30	18	8	2

Table 5: Multi-Level Threshold Values

Table 4 shows the number of PDUs generated and the average error in AOI and SR when our multi-level threshold dead reckoning algorithm is used. The threshold values used in different levels are listed in 5.

It can be seen from Table 4 that there is a great reduction in the average error in SR, compared to the average error in AOI. In our algorithm, if entity A is in entity B's SR, a minimum threshold will be used in the dead reckoning so that B will receive A's update packets most frequently. In this case, B will know A's position most accurately.

Compared to the case where the threshold is fixed at the minimum (that is, 2), using multi-level threshold, the reductions on the number of PDUs are 29% and 19% for 16 and 32 entities respectively. The average errors in AOI increase. But, the average errors in SR are unaffected in both scenarios. Compared to the case where the threshold is fixed at 8, the number of PDUs increases slightly when a multi-level threshold is used. The percentage increments are 2% and 14% respectively for 16 and 32 entities. However, the average errors in SR are reduced greatly by about 77% for both situations. The average errors in AOI are also reduced.

In summary, the above two experiments demonstrate the feasibility of our adaptive, multi-level threshold dead reckoning algorithm. The results show that using a multi-level threshold in dead reckoning, adequate extrapolation accuracy can be achieved with a reduced number of state update packets.

4. AUTOMATIC SELECTION OF EXTRAPOLATION EQUATION

An entity typically performs various movements in a virtual environment. However, they can be classified generally into three types, that is, *smooth*, *bounce* and *jolt* move, as shown in Figure 8.

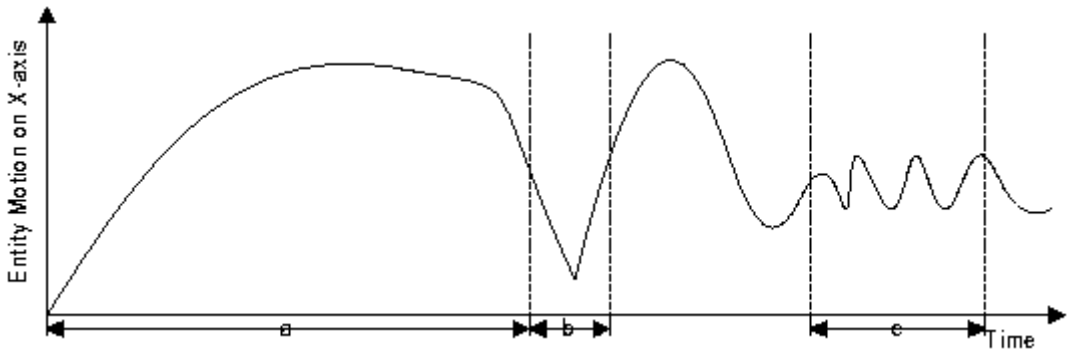

Figure 8: Characteristics of Entity Movement

Figure 8 shows the trajectory of an entity's movement along one dimension (e.g., the *x* dimension). When an entity's velocity changes continuously with no jump variation, we say that the entity is making a *smooth move* (as shown in part (a) of Figure 8). If an entity has a collision, there will be an abrupt, drastic change in the direction in which the entity is traveling. At this moment, the trajectory of the entity is more like a "sawtooth", as shown in part (b) of Figure 8. This kind of movement is referred to as a *bounce*. If an entity has frequent sudden changes of its direction, its trajectory will look like the one shown in part (c) of Figure 8. In this case, we say that the entity is in a *jolt* motion. Frequent bounce will lead to a jolt motion. This type of motion is usually difficult to deal with in DIS because the high frequency of state changes may result in a large number of state update packets.

4.1 Entity's Motion and Extrapolation

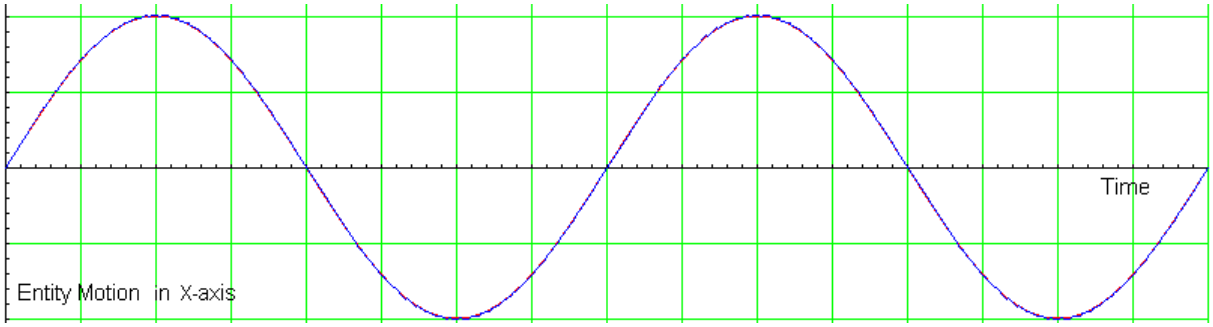

Figure 9: Trajectory of A Smooth Motion

Figure 9 shows a trajectory in the *x* dimension. This smooth trajectory can be produced by an entity that moves around a fixed point in a 2-D virtual environment. Table 6 shows the number of state update packets generated when various extrapolation formulas are used under different threshold values. When the threshold is tight, second order formulas generate fewer update packets than first order formulas. As the threshold value increases, the number of update packets generated decreases for all the formulas. However, first order formulas have a faster decreasing rate than second order formulas. So, when the threshold is loose, both first and second order formulas generate more or less the same number of update packets. Obviously, for smooth motion, second order formulas are the better choice.

Error Tsd	*One-Step*		*Two-Step*	
	1st order	*2nd order*	*1st order*	*2nd order*
2	49	22	64	26
5	29	17	41	20
10	22	14	28	16
20	15	9	20	12
30	13	9	16	9
40	8	9	12	8

Table 6: Results of Extrapolating Smooth Motion

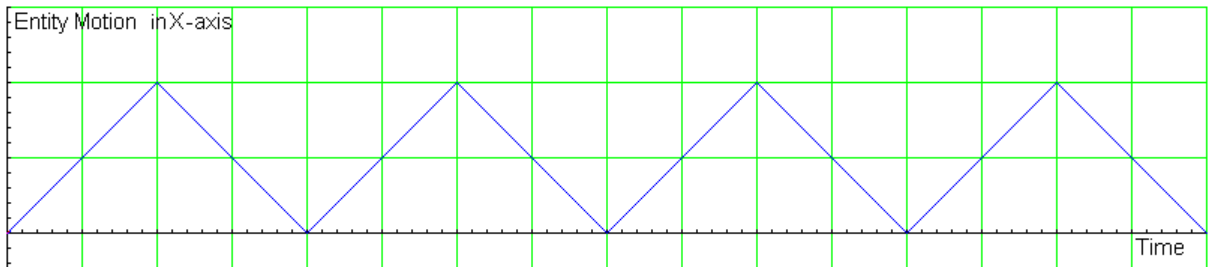

Figure 10: Trajectory of A Bounce Motion

The second trajectory illustrates a bounce motion, as shown in Figure 10. This trajectory can be generated by an entity that moves back and forth at a constant velocity between two fixed points. The performance results of various extrapolation formulas are listed in Table 7, which clearly show that one-step formulas outperform two-step formulas. Extrapolation using two-step formulas may introduce errors at each bouncing point, because the calculation using the positions (or velocities) in the last two update packets may not give the correct velocity (or acceleration). Thus, for extrapolating bounce motion, one-step formulas are more appropriate.

Error Tsd	*One-Step*		*Two-Step*	
	1st order	*2nd order*	*1st order*	*2nd order*
2	8	9	15	15
5	8	9	15	15
10	8	9	15	15
20	8	9	15	15
30	8	9	15	15
40	8	9	15	15

Table 7: Results of Extrapolating Bounce Motion

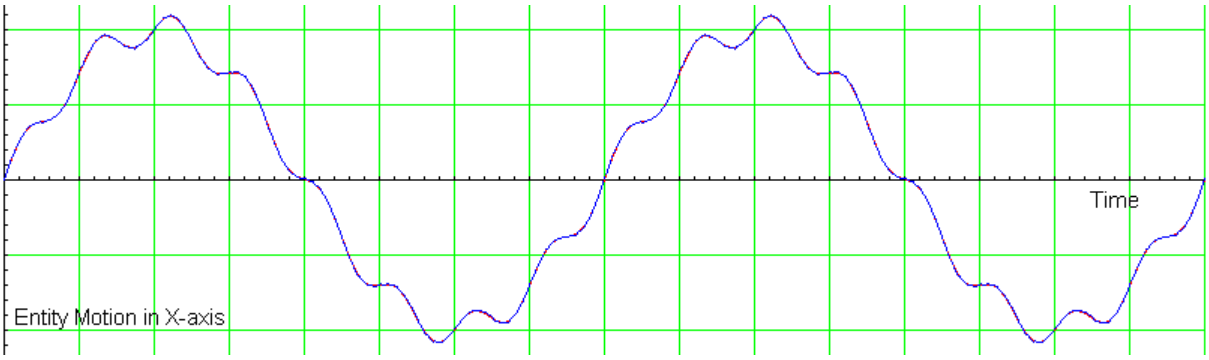

Figure 11: Trajectory of A Jolt Motion

The last trajectory, shown in Figure11, illustrates a jolt motion. An entity that spins itself while moving in a circle around a fixed point may produce this kind of trajectory. Table 8 tabulates the performance results of various extrapolation formulas. The results show that when the threshold is tight, second order formulas generate fewer update packets than first order formulas. However, when the threshold becomes larger, first order formulas will outperform second order formulas. So, for jolt motion, the type of formulas to be used in the extrapolation depends on the threshold values.

Error	*One-Step*		*Two-Step*	
Tsd	*1st order*	*2nd order*	*1st order*	*2nd order*
2	114	67	139	96
5	60	46	81	64
10	39	34	60	34
20	23	33	31	33
30	23	33	20	33
40	16	33	14	33

Table 8: Results of Extrapolating Jolt Motion

In summary, for the smooth motion, second-order formulas (either one-step or two-step) are the appropriate choice, since the entity moves smoothly and its velocity changes continuously. In this case, the velocity and acceleration can be used effectively to predict the trend of the entity's movement. For the bounce motion, one-step formulas are more suitable than multi-step formulas. Because of the abrupt changes of the entity's movement, the information from the old update packets may not be useful in predicting the future position of the entity. As a result, the velocity (or acceleration) calculated based on the information from two or more continuous update packets may not be accurate. For the jolt motion, when threshold is tight, second order formulas should be used. However, when threshold is loose, first order formulas are a better choice. Because of the frequent, sudden changes of the entity's movement and the large degree of error tolerance (i.e., the large threshold value), the use of acceleration in the extrapolation may introduce more errors.

4.2 The Algorithm

As entities perform various activities in a DIS exercise, their motion may change from time to time during the exercise. In addition, to reduce the transmission of unnecessary state update packets, variable threshold values may be used in the DR algorithms (as described in section 3). The results obtained in the last section show that the performance of an extrapolation formula depends on the current motion of the entity as well as the threshold value. This suggests that to optimize the performance of a DR model, the extrapolation formula needs to be adaptively selected during the simulation.

Motion	*Threshold*	*Formula*	*Code*
smooth	any	Two-step, 2nd order	0
bounce	any	One-step, 1st order	1
jolt	tight	One-step, 2nd order	2
jolt	loose	Two-step, 1st order	3

Table 9: Selection of Extrapolation Formula

The algorithm that automatically selects the extrapolation formula according to the current motion of the entity and threshold value is given in Figure 12. The selection strategies used in the algorithm are listed in Table 9. These strategies are made based on the consideration of computation efficiency and the results obtained in the last section. For extrapolating smooth motion, the two-step second order formula is used instead of the one-step second order formula, because it requires less information from the update packets. Similarly, since the first order formula has less parameters than the second order formula, for bounce motion, the one-step first order formula is used instead of the one-step second order formula. For jolt motion, the one-step second order formula is selected for tight threshold, and the two-step first order formula is adopted for loose threshold.

```
/* for simulator i */
/* assume that ∀ e, FormulaCode[e] = 0 and SendFlag = True initially */
for each local entity e ∈ simulator i do
    for each state update do {
        extrapolate e's position based on the past state information;
        /* update BounceTime -- (a) */
        for each ele ∈ BounceTime do
            if ele < (CurrentSimTime - Δt) then
                BounceTime = BounceTime – {ele}
        /* decide which formula to trigger -- (b) */
        if change of direction > BounceAngle then {
            BounceTime = BounceTime ∪ {CurrentSimTime};
            if sizeof(BounceTime) > JoltTrigger then
                if tight(Threshold[e]) then TriggerCode[e] = 2;
                else TriggerCode[e] = 3;
            else TriggerCde[e] = 1;
        } else TriggerCode[e] = 0;
        /* correct extrapolation and send update packet if necessary -- (c) */
        if FormulaCode[e] != TriggerCode[e] then {
            FormulaCode[e] = TriggerCode[e];
            change extrapolation formula and correct entity extrapolation accordingly;
            send an update packet with TriggerCode[e] to other simulators;
            SendFlag = false;
        }
        /* send update packet if necessary -- (d) */
        if SendFlag ∧ ( |TruePosition – ExtrapolatedPosition| > Threshold ) then
            send an update packet with TriggerCode[e] to other simulators;
    }
```

Figure 12: Algorithm for Selection of Extrapolation Formula

To trigger the selection of different formulas, variables FormulaCode, TriggerCode, BounceAngle, JoltTrigger and BounceTime are used in the algorithm. For each formula to be used, a code is assigned (see Table 9). Initially, it assumes that an entity is in a smooth motion, thus FormulaCode is 0. As explained in the last section, an entity is making a bounce move if there is an abrupt, drastic change in its direction. In the algorithm, variable BounceAngle is used to detect whether or not an entity is making a bounce move. Array BounceTime is used to keep track of the number of bounce moves made by an entity in a given period of time (that is, Δt). If an entity makes more than JoltTrigger times of bounce moves during Δt, the entity is in a jolt motion. The classification of tight and loose threshold (i.e., function *tight*( )) will be explained in the next section.

Based on the selection strategies shown in Table 9 and the triggering mechanism described above, TriggerCode will then be decided. If the current FormulaCode is different from the TriggerCode, the extrapolation formula will be changed accordingly. To inform other simulators of the change, an update packet needs to be sent with the TriggerCode (that can be piggybacked to the update packet using two additional bits). Sending additional update packets whenever an entity changes its motion may not introduce additional overhead, since the change in the entity's motion often results in a large error in extrapolation.

An update packet will also need to be sent if the difference between the true and extrapolated position is larger than the threshold. For other simulators, upon receiving an update packet, they will use the TriggerCode to make a correction of the extrapolation if necessary.

4.3. Experimental Results

Figure 13 shows a trajectory of an entity's movement with mixed periodical smooth and bounce motions. In this experiment, BounceAngle is set to 90 degree, JoltTrigger is equal to 3, and Δt is fixed as 1 second. Table 10 shows the results of the experiment. It can be seen that for this mixed motion, second order formulas perform better when the threshold is tight. As the threshold becomes loose, first order formulas will outperform second order formulas. However, for the algorithm based on automatic formula selection, it always has a comparably good performance no matter what threshold value is used. The algorithm will automatically select an appropriate extrapolation equation according to the current motion of an entity

and the threshold value. This will minimize the extrapolation errors, which in turn will result in fewer update packets being transmitted and thus a higher performance.

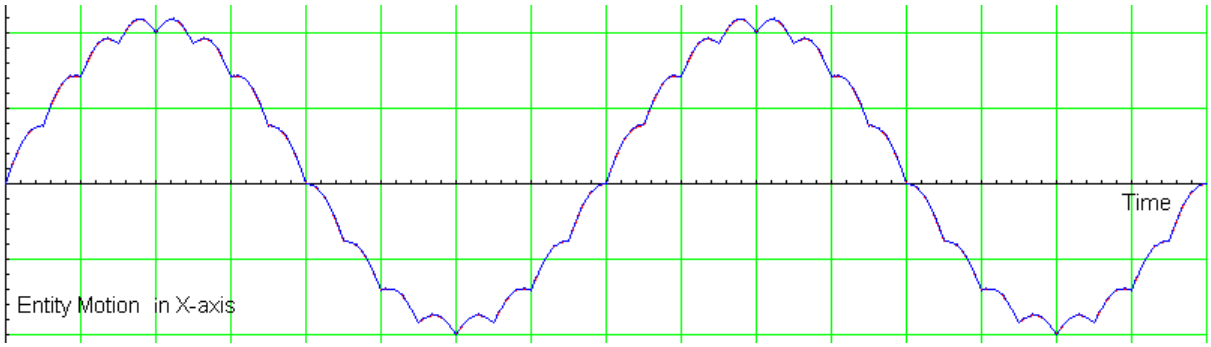

Figure 13: Trajectory of A Mixed Motion

Error Threshold	*Auto Selection*	*One-Step*		*Two-Step*	
		1st order	*2nd order*	*1st order*	*2nd order*
2	98	140	96	186	104
5	71	96	65	96	75
10	55	58	65	41	64
20	22	25	33	20	34
30	18	19	33	16	21
40	14	10	33	13	21

Table 10: Results of Extrapolating Mixed Motion

5. CONCLUSIONS

This paper describes two new adaptive algorithms for dead reckoning in DIS. The first algorithm is based on the control mechanism of adaptive adjustment of threshold level and the second algorithm exploits the control mechanism of automatic selection of extrapolation equation.

Since using a fixed threshold to control extrapolation error may either generate unnecessary update packets (when the threshold is tight) or result in mistakes in the simulation (when the threshold is loose), a multi-level threshold scheme is proposed in the paper. The definition of threshold levels is based on the concepts of AOI and SR, and the levels of threshold are adaptively adjusted based on the relative distance between entities during the simulation. Depending on the threshold level, an update packet is sent only to the relevant entities. Thus, a close by entity may receive more update packets than an entity that is far away. To test the scheme, an experiment was conducted. The results show that the multi-level thresholds can adequately reflect the dynamic relationship between entities. It reduces the rate of transmitting update packets while maintaining adequate accuracy in the extrapolation.

Different motions made by an entity are defined in the paper. Experiments were conducted and the results show that the performance of an extrapolation formula depends on the error threshold as well as an entity's motion. When an entity's motion and threshold value vary, there is no single formula that consistently outperforms other formulas.

An entity may perform various activities in its life time. The motion of an entity may change from time to time during simulation. The threshold value may also be dynamically adjusted depending on the relationships between entities. To optimize the performance, a control mechanism that automatically selects an appropriate extrapolation formula according to entity's current motion and threshold value is also proposed. The choice of the extrapolation formula for a given motion type is made based on a number of experiments. The switching between formulas is dynamically triggered by the change in the entity's movement. The experiment results show that the algorithm always has a comparably good performance regardless of an entity's motion and threshold.

However, many details still need to be worked out to correctly integrate the two control mechanisms. Problems that may arise when using the mechanisms, for example, handling of packet loss, also need to be studied. In addition, to investigate the overhead of the mechanisms and to obtain more significant results, further experiments must be carried out.

REFERENCES

Bassiouni M, Chiu M-H., Loper M. and Garnsey M. 1997, "Performance and Reliability Analysis of Relevance Filtering for Scalable Distributed Interactive Simulation." *ACM Trans. on Modeling and Computer Simulation*, Vol.7, No.3. 1997

Bassiouni M., Chiu M-H., Loper M. and Garnsey M. 1998, "Relevance Filtering for Distributed Interactive Simulation." *Computer Systems - Science & Engineering*, Vol.13, No.1, pp.39-47, Jan 1998.

Department of Defense 1998, *High Level Architecture Programmers Guide*. Version 1.3, April 1998. (http://www.dmso.mil/hla/)

DIS Steering Committee 1994, "The DIS Vision, A Map to the Future of Distributed Simulation." Technical Report IST-SP-94-01, Institute for Simulation and Training, Orlando, Florida. 1994. (ftp://ftp.sc.ist.ncf.edu/SISO/dis/library/vision.doc)

Durbach C. and Fourneau J-M. 1998, "Performance Evaluation of a Dead Reckoning Mechanism." In *Proc. of IEEE Second International Workshop on Distributed Interactive Simulation and Real-time Applications*, pp.23-29, Montreal Canada, July 1998.

Figart G., Slaton C. and Slosser S. 1996, "Using Encumbrance Factors to Improve Dead Reckoning Approaches to Object Regeneration and Modeling." In *Proc. of 14th DIS Workshop on Standards for the Interoperability of Distributed Simulation*, March 1996.

Foster L. and Maassel P. 1994, "The Characterization of Entity State Error and Update Rate for DIS." In *Proc. of 11th DIS Workshop on Standards for the Interoperability of Distributed Simulation*, pp.61-73, Sept. 1994.

IEEE 1278-1993, "Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications." 1993.

Katz A. 1994, "Synchronization of Networked Simulations." In Proc. of the 11th DIS Workshop on Standards for the Interoperability of Distributed Simulation, pp.81-87, Sept. 1994.

Lin K-C. 1994a, "The Performance Assessment of the Dead Reckoning Algorithms in DIS." In *Proc. of 10th DIS Workshop on Standards for the Interoperability of Distributed Simulation*, March 1994.

Lin K-C. 1994b, "Study on the network load in Distributed Interactive Simulation." In *Proc. of AIAA on Flight Simulation Technologies*, 1994.

Lin K-C. 1995, "Dead Reckoning and Distributed Interactive Simulation." In *Proc. of SPIE Conference (AeroSense'95), Orlando Florida*, April 1995.

Morse K.L. 1996, "Interest Management in Large-Scale Distributed Simulation." Department of Information & Computer Science, University of California at Irvine, Technical Report #96-27, 1996. (http://www.ics.uci.edu/~kmorse/survey_paper.ps)

Pope A.R. 1991, "The SIMNET Network and Protocols." Report No. 7262, BBN Systems and Technologies, Cambridge MA, June 1991.

Rak S.J. and van Hook D.J. 1996, "Evaluation of Grid-based Relevance Filtering for Multicast Group Assignment." In *Proc. of 14th DIS Workshop on Standards for the Interoperability of Distributed Simulation*, pp 739-747, March 1996.

Shanks G.C. 1997, "The RPR FOM, A Reference Federation Object Model to Promote Simulation Interoperability." In Proc. *Spring Simulation Interoperability Workshop*, 1997.

BIOGRAPHY

BU SUNG LEE received his B.Sc. (Hons) and Ph.D. from the Electrical and Electronics Department, Loughborough University of Technology, UK, in 1982 and 1987 respectively. He is currently an Associate Professor with the Nanyang Technological University, Singapore. He is a member of the Asia Pacific Advance Network (APAN) and the Application Manager of Singapore Research & Education Networks (SingAREN). He has been an active member of several national standards organization such as the National Infrastructure Initiative (Singapore One) Network working group, the Singapore ATM Testbed. His research interests are in network management, broadband networks, distributed networks and network optimization.

WENTONG CAI is an Associate Professor with School of Computer Engineering at Nanyang Technological University (NTU), Singapore. He received his B.Sc. in Computer Science from Nankai University (P. R. China) and Ph.D., also in Computer Science, from University of Exeter (U.K.). He was a Post-doctoral Research Fellow at Queen's University

(Canada) before joining NTU in Feb 1993. Dr. Cai has been actively involved in the research in Parallel and Distributed Simulation (PADS) for more than ten years. His current research interests include: PADS, Parallel Programming Environments & Tools, Parallel Algorithms & Architectures and System Performance Analysis.

STEPHEN TURNER is an Associate Professor in the School of Computer Engineering at Nanyang Technological University (Singapore). Previously, he was a Senior Lecturer in Computer Science and Director of the Parallel Systems Research Laboratory at Exeter University (UK). He received his MA in Mathematics and Computer Science from Cambridge University (UK) and his MSc and PhD in Computer Science from Manchester University (UK). His current research interests include: Distributed and Parallel Simulation (PADS), Visual Programming Environments, Parallel Algorithms and Languages, Distributed Computing and Agent Technology.

L. CHEN was a Master student at School of Computer Engineering, Nanyang Technological University. He graduated in 1999 and is currently a Senior System Analyst with MobileOne (Asia) Pte Ltd, Singapore.