A Generic Location Event Simulator

Kumaresan Sanmugalingam and George Coulouris

Laboratory for Communications Engineering,
Department of Engineering, Cambridge University, Cambridge, UK
{ks286, gfc22}@cam.ac.uk

Abstract. This note describes a standalone generic location event simulator that has been designed for the visualisation, scalability testing and evaluation of location-aware event-driven middleware and applications.

1. Introduction

Middleware for location-aware applications has recently emerged from several projects [1,2,3,4]. They are designed to manage real-world location events and to provide relevant high-level event streams to location and context-aware applications. The simulator described here has been developed in the context of the QoSDReAM middleware project [1,2].

Scalability and soak testing are difficult to perform in such systems, since the raw events are derived from physical sensor technologies, therefore requiring developers to move about in the manner that they would expect users to, under normal circumstances, in the user's own environment. The frequency, distance and speed with which they move differs greatly and is application domain dependent, meaning there is a need for testing with a costly prototype deployment. Scalability testing also requires the deployment of large numbers of *locatables* – objects whose locations can be sensed. But it is not possible in a typical laboratory with a relatively small number of people, who do not move around as much as the thousands of travellers, staff and vehicles moving though an airport on a daily basis. It would be useful to be able to easily produce evidence that shows our middleware can scale to the specified number of events required by the end user. In addition, application designers often need to perform early evaluations of alternative designs before access to sensor systems is available. Beyond visualisation, analysis of the data would provide operations researchers and transport engineers with the information they need to be able to help in the architectural planning of buildings by predicting problems associated with crowds building up at bottlenecks for crowd control and safety of crowds etc. Thus, there is now a real need to be able to simulate an environment for testing large-scale sentient applications. We have developed an event-based simulator that models the behaviours of locatables in a simple model of a physical space. The simulator generates and sends simulated location events into a generic location event-driven middleware platform.

This section discusses the reasons why such a simulator is needed in large scale ubiquitous computing middleware. Section 2 describes the overall architecture and

the main components that are needed to build it. Section 3 goes in to more detail on the most important part of the simulator, the main components needed in building realistic behaviour in to the objects being simulated. Section 4 presents our results, and some discussion on what the results said about our location-aware middleware. Finally we discuss further work and conclude.

1.1 Background

Hybrid systems are dynamical processes that are heterogeneous in nature, having continuous parts, governed by differential or difference equations, and by discrete parts, described by finite state machines and automata, if-then-else rules, propositional and temporal logic. They switch between many operating modes, with mode transitions triggered by variables crossing specific thresholds (state events), by the elapse of certain time periods, or by external input events.

Our simulator is based on a process-oriented hybrid simulation model of locatables' behaviour in terms of physical movement as well as stoppages at queues, combining a continuous mechanics model, a discrete-event queuing model, a sensor model, and a few other supporting physics models. Each locatable has a mostly continuous behaviour until an event occurs suddenly changing its state (say in velocity or position) at discrete time steps. There has been no work using these techniques in ubiquitous computing, however it is becoming popular in control systems. In our simulator, locatables are modelled as generic vehicles, so we can adopt approaches from standard control theory problems of cruise control, brake-by-wire and steer-by-wire systems. A major project in this area is Shift [5], a Berkeley Path hybrid automata programming language for Autonomous Guided Vehicles. The basic ideas for simulation and queuing theory, such as having a central scheduler and queue array-list, are common across all object-oriented discrete-event queuing simulators [6]. The traffic simulation part of the system (the computer simulation of traffic engineering models) can be classified in to *microscopic* (vehicle-vehicle interactions), *mesoscopic and macroscopic* (a continuous flow, for traffic flow analysis) categories. Most urban transportation problems are network related [7], and a lot of ideas for the queuing model part of the system, for example being able to specify delays on links, came from Berkeley's ns2 simulator [8], however our system required locatables to be able to move freely in all three dimensions of space.

Finally, although their requirements are the opposite of ours, we should mention Ubiwise (WISE/Nexus/Ubisim) [9] and QuakeSim [10] which aim to help demonstrate ubiquitous computing applications on PCs, simulating the user interface in 2D/3D, the sensors, the communications protocols, etc. Both decided to use the Quake III arena engine (as one of their requirements is for a realistic looking GUI), and QuakeSim has integrated the Context Toolkit [4] into itself (to gather, aggregate, interpret and publish the context information). In the conclusion of their recent ACM paper [11] the QuakeSim authors mention that their system is only "for demonstration purposes, and for *small-scale* testing during development" and that "for full-scale system tests, it is necessary to use the real system deployed in its real setting" – however that is not the case with the simulator described here, which has behavioural simulation of locatables.

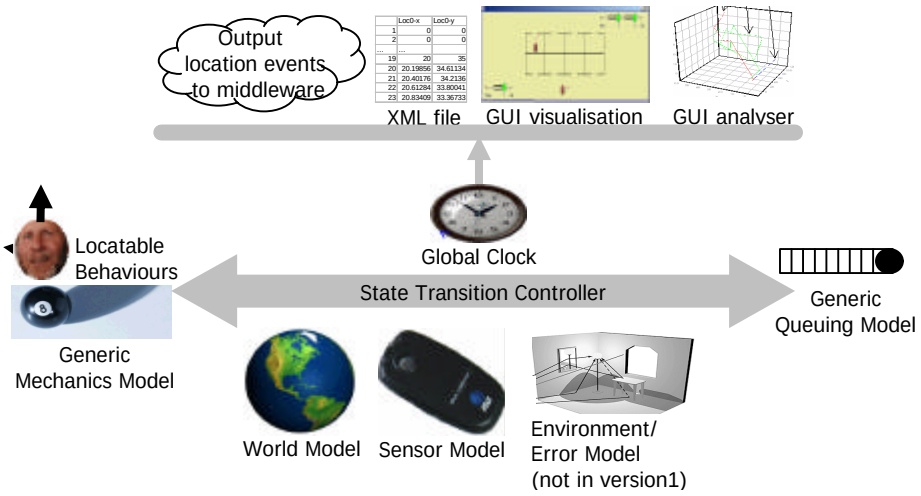

Fig. 1. Architecture of the generic location event simulator

2. Architecture

2.1 Main components

The simulator has a central State Transition Controller which sets the initial positions of all the locatables, assigns tasks to them when their current task has just been completed, and manages the state transitions between the mechanics and queuing models. The Global Clock ensures all processes are running the same time-step, thus governing the overall frequency and consistent global ordering of events being generated, an important consideration when the simulator's threads need to be distributed for scalability reasons.

There are several different models in the system, discussed in more detail in the section on "Pluggable models and behaviours". The Mechanics and Queuing Models help define the behaviour of the locatables, the Sensor and Environment/Error Models simulate the unique and dynamically changing physics present in the room (for example how ambient light can affect an infrared-based sensor such as the ActiveBadge system [12]), and the World Model is a model of the building's geometry.

2.2 Operation

Once all the models are defined, and the simulator is run, the locatables enter and exit the world via the Mechanics or Queuing Model. In the examples implemented, they enter with a certain inter-arrival rate, and exit after a certain period of time, or when certain tasks have been completed.

At the bottom left-hand corner of Fig.2, the locatables enter the simulator's queue1 via the State Transition Controller, at times governed by a set inter-arrival-rate. After waiting in the queue and being served (governed by the queue length and a set maximum serving rate), they exit the Queuing Model and enter the Mechanics Model, again via the State Transition Controller. In this model, they move around the World Model avoiding collisions, either at random or governed by their set tasks, defined in their behaviour modules (such as go to shops area, browse around this area for 60 seconds, enter the nearest shop, wait there for 30 seconds,). Finally at a specified time they move towards that locatable's departure gate, entering Queues 2&3, at the top right-hand corner of Fig.2. This is another set of state transitions, controlled by the State Transition Controller.

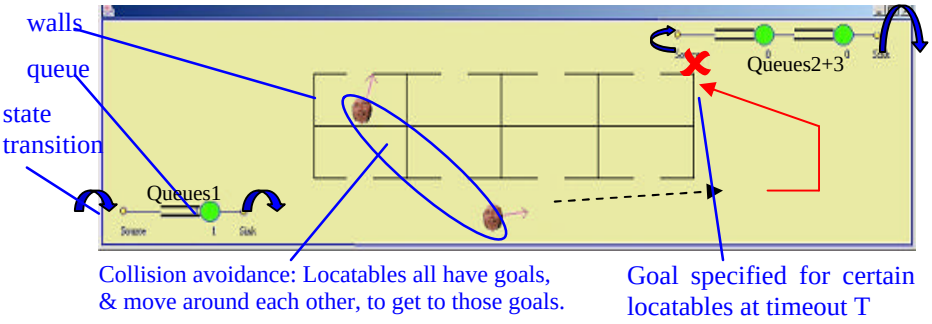

Fig. 2. A screenshot of an example simulation run of the simple airport terminal demo

3. Pluggable models and behaviours

Genericity and pluggability are the most important requirements for this software. The simulator runs as a stand-alone program, so it is very easy to forward location events, to any middleware that needs to be tested, from a completely pluggable sensor layer. Its generic queuing model has pluggable statistical distributions to allow for more complex, customisable inter-arrival behaviour, and its generic mechanics model has pluggable equations to allow for realistic high-level movement-based behaviours of any locatable, so it is possible to extend it to model vehicles moving outdoors. The code will soon be open-sourced, so that developers and researchers will have the ability to plug it in to other middleware, and to encourage further development of plug-ins of sensors and behaviours.

3.1 World model

The simulator generates events, and is intended for use with any middleware that will perform the appropriate calculations on those events. There is a need for a clear world model, to make it easier for application programmers to build worlds (such as an airport, railway station, etc), by simply specifying room geometry, centre-of-door locations etc, that use our simulator to generate locatables in that world.

This model of the real world environment needs to be consistent with the middleware being tested. In the QoSDReAM middleware framework, the model has static *regions* (2D geometric shapes) stored in an object database defining room geometry (walls, doors, stairs, escalators, lifts, etc), and dynamic regions defined dynamically via events from the underlying (simulated or real) sensor subsystem, around itself for every monitored physical object. When an overlap occurs between two significant regions, an event is fired up to the applications that have subscribed to that particular event type (and the necessary event filtering is taken care of automatically).

3.2 Sensors

The simulated event streams have exactly the same format as the events that the real world physical sensors generate, and thus it is impossible for the middleware to tell them apart. It is possible to interchange, or even to mix, real and simulated events in real time, to serialize the time-stamped event stream into a file for graphing, and to replay them afterwards in real-time which is useful in tracking bugs that occur at run-time.

Users have been able to generate a variety of sensor event types, of varying resolution and frequency, because of the regions abstraction discussed above, including sensors which measure geometric position (ActiveBat [14]), symbolic position (the room-contained ActiveBadge [12]), vision (TRiP [13]), proximity (login/keyboard monitors), etc.

3.3 Human behaviour

Simple behaviours are composed into more complex composite behaviours (that can include behaviour based on position, time, locatatorId, locator type, etc), and one of these, including all its state information about time etc, can be assigned to a locatable, or to a set of locatables based on their defined role, as their current task. These roles are similar to defining locatable-types, such that all locatables of those types take on the behaviours specified to that type. The behaviours implemented so far are:

Initialisation and queue behaviour

The queuing model is fairly generic. The positions and characteristics of the queues in the system need to be specified, including their positions (x,y,width,height), λ (the birthrate in a Markov chain) along with its probability distribution for a pseudo-random inter-arrival rate (oscillates between the chosen $1/\lambda$ and 0), μ (the service time of the server), their topology (MM1's in series, MMn, a branched MMn network with specified probabilities for the branch directions, etc), a buffer limit, and to specify the event to be sent to the mechanics model when a locatable has gone through that queue.

Straight-line behaviour

This is the simplest behaviour in the mechanics model, and locatables continue to move in a straight line, until their collision avoidance algorithm changes the path because an obstacle was detected.

(Sequence of) goal-seeking behaviour

A locatable tries to travel, via the shortest-path-route whilst avoiding obstacles, towards a specified goal. An extension to this behaviour is to specify a sequence of bounded random sub-goals, which helps to create believable movement, and goal-timeouts were set to make sure locatables never become stuck within a room. Automated sub-goal calculation, based on an "all-possible-movements-plane", will be implemented as part of an overall framework.

Finally, the basic *collision avoidance* algorithm always exists underneath all the different behaviours mentioned above (however, this too is pluggable), as we assume all locatables will always try to avoid colliding with each other. The algorithm makes changes to the velocity based on differential equations for realistically feasible motion of any generic locatable, with an exponentially increasing slow-down as the locatable gets nearer the obstacle, and with different locatable-types having different turning rates and turn radii – for example, humans can turn abruptly on the spot to avoid moving locatables, and realistically detect *static* objects (walls) earlier, slowly applying only small changes to the path, so the end result are locatables moving *parallel* to walls, which is what happens in real-world corridors.

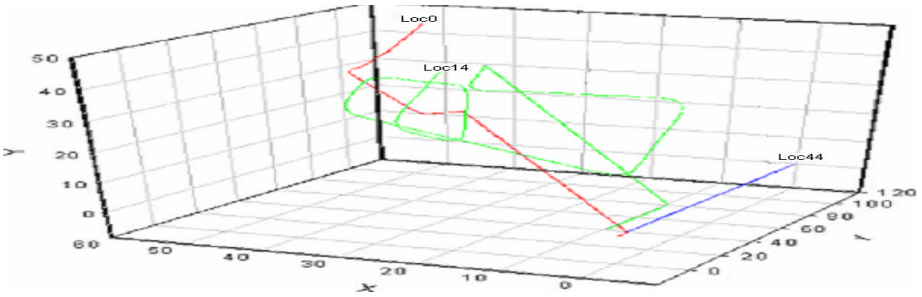

Fig. 3. An example run – graph showing only 3 of the 50 locatable's tracks. The axes are 2D space (x, y) and time (t)

4. Results

The results here consist of microsimulation (mechanics and queuing theory) and macroscopic (fluid flow approximation [15]) models. The simulator can output results to a file whilst running in a GUI-less mode, to speed up execution on low-end computers and to make sure that the most accurate performance results possible are obtained for the middleware being measured. Each locatable runs in its own thread, which allows it to scale to very large spaces with a high number of events, and to also have a distributed implementation. The main limitation to scale is the number of locatables that can fit in a given area at any one time. Obtaining architectural data, we assume average walking speeds of 5 ft/s (≅ *1.5 m/s*), and set the area to that of Atlanta

Airport (529,547m^2). This demo was not tested against real world data, however this is being resolved (section 5).

What can be seen in Fig.3 is that locatables try and spread out to avoid the crowds that inevitably build up in the centre, and to minimise the probability of a collision occurring. They only walk across the centre when executing a shortest path goal-seeking behaviour. Locatables enter the mechanics model (from queue1) at different times, with Locatable44 not even entering (because t=120 is too low for it to enter and exit queue1, although it did arrive at queue1 at t=40), and Locatable14 not getting near to queue2 because it gets blocked by all the other locatables that were nearer at t=120.

In Fig.4, the difference between the two sets of bars indicates the time spent in queue1 for each locatable, which is growing because we specified λ=0.05, μ=0.067.

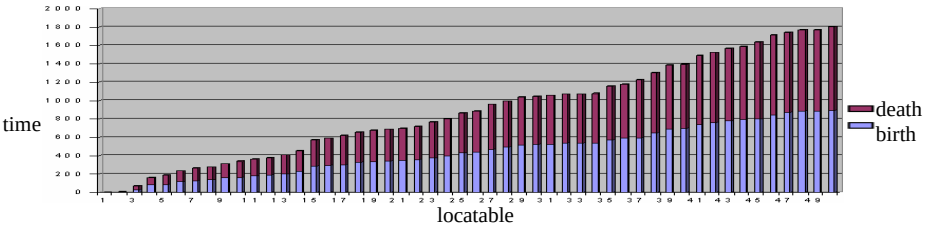

Fig. 4. Analysis of the effect of build ups in the Queue1 of the example run. λ=0.05, μ=0.067

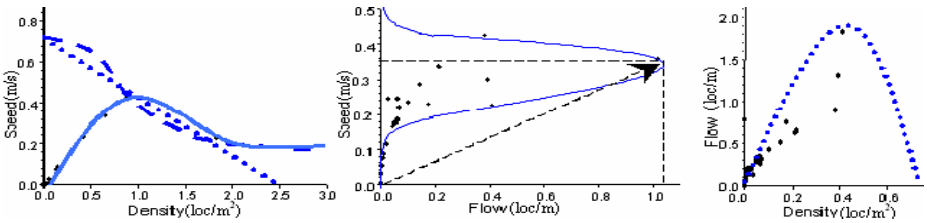

Fig. 5. Macroscopic fluid flow analysis

Fig.5 demonstrates the kind of analysis possible with macroscopic fluid-flow approximation. The first graph of Fig.5 shows that between a density of 0 and 1 locatable/m^2, speed increases from 0 m/s to a maximum (as expected, because with 0 locatables, speed is 0, and with only 1 locatable in the unit square area, there is nothing to slow it down), then decreases exponentially as more locatables crowd the unit square area (the collision avoidance algorithm forces the locatables to slow down). The linear graph (dotted-line) shows a well-known approximation to the speed-density relation, where this slowdown would continue to a complete standstill, like bumper-to-bumper traffic jams in the case of vehicles, at (2.5, 0). The "backward-S" curve (dashed-line) is another well-known graph, based on observations of cars on highways. The second graph clearly shows that there is a maximum flow rate, and we *define* the x-y points being where there is optimum density and speed. The final graph shows that flow increases until it reaches a maximum (at optimum speed and density), then decreases to zero reaching the jam density mentioned above.

5. Conclusions and Future Work

The successful test of the simulator helped convince us that having large numbers of simple pluggable locatable behaviours is the correct architecture for a simulator supporting the design, development and testing of ubiquitous computing applications.

More behaviours are being added by our group and soon by others (currently being developed are “moving target tracking” and “fluid flow following”). The airport demo was not tested against real-world location data, so we are now comparing the latest version of this simulator against ActiveBat[14] data from an office environment (AT&T Labs, Cambridge). Finally, we may work on more advanced control theory, such as model verification of the hybrid system, and more integrated behaviour and feedback loops using adaptive, neural, and fuzzy-control systems.

References

1. Naguib, H. and G. Coulouris, Location Information Management, in Proc. UbiComp01, Atlanta, GA. September 2001 http://www-lce.eng.cam.ac.uk/qosdream/
2. H.Naguib, G.Coulouris, K.Sanmugalingam An Open Framework for the Development of Location-Aware Systems. Submitted to Ubicomp2002.
3. Mike Addlesee, Rupert Curwen, Steve Hodges, Joe Newman, Pete Steggles, Andy Ward, Andy Hopper. Implementing a Sentient Computing System. IEEE Computer Magazine, Vol. 34, No. 8, August 2001, pp. 50-56.
4. A.Dey, J.Mankoff, G.Abowd. Distributed Mediation of Imperfectly Sensed Context in Aware Environments. GVU Tech Report GIT-GVU-00-14. Submitted to UIST 2000.
5. M.Antoniotti,A.Göllü, SHIFT and SMART-AHS: A Language for Hybrid System Engineering, Modelling and Simulation., USENIX Conference on Domain Specific Languages, 1997.
6. R. Jain, "The Art of Computer Systems Performance Analysis" Wiley-Interscience, April 1991.
7. A.Byrne, A. de Laski, K.Courage, C.Wallace, “The American NETSIM from the 1970's”. Handbook of computer models for traffic operations analysis, FHWA-TS-82-213, 1982
8. NS workshops 97-00, UCBerkeley, http://www.isi.edu/nsnam/ns/ns-tutorial/index.html
9. "WISE – A Simulator Toolkit for Ubiquitous Computing Scenarios" Vikram Vijayraghavan and John J. Barton. UbiTools-'01 workshop, part of UbiComp 2001
10. Markus Bylund, Fredrik Espinoza, "Using Quake III Arena to Simulate Sensors and Actuators when Evaluating and Testing Mobile Services". CHI2001, Extended Abstracts, pp 241-243.
11. Markus Bylund, Fredrik Espinoza: Testing and demonstrating context-aware services with Quake III Arena. CACM 45(1): 46-48 (2002)
12. Roy Want, Andy Hopper, Veronica Falcao, Jonathon Gibbons . The Active Badge Location System. ACM Transactions on Information Systems, Vol. 10, No. 1, January 1992, pp 91-102.
13. Diego López de Ipiña, Paulo Mendonça and Andy Hopper, "TRIP: a Low-Cost Vision-Based Location System for Ubiquitous Computing", Personal and Ubiquitous Computing, 2002.
14. Andy Ward, Alan Jones, Andy Hopper. A New Location Technique for the Active Office. IEEE Personal Communications, Vol. 4, No. 5, October 1997, pp. 42-47.
15. D.Helbing, A.Hennecke, V.Shvetsov, M.Treiber, “MASTER: Macroscopic Traffic Simulation Based on A Gas-Kinetic, Non-Local Traffic Model”, Inrets proceedings, in press, 1998.