A Simulation Framework for the Investigation of Adaptive Behaviours in Largely Populated Building Evacuation Scenarios

Daniele Gianni, Georgios Loukas, Erol Gelenbe

Department of Electrical and Electronic Engineering
Imperial College London
SW7 2BT, London, UK
{gianni, gl1, e.gelenbe}@imperial.ac.uk

Abstract. In an emergency scenario, civilians and emergency personnel have to continuously adapt their behaviour and make quick decisions to tackle unpredicted developments. Determining the optimal decisions and devising viable operational plans, while adapting to world changes, require systematic and accurate investigation of such systems. To effectively carry out such investigations in largely populated scenarios, we need a software framework that allows (i) *reproducibility of the experiments*, (ii) *extendibility* to diverse and unforeseen scenarios and (iii) *distributed operation* to allow the simulation of largely populated scenarios. We achieve all three requirements by developing an agent-based discrete-event simulation framework, and then building on top a *Building Evacuation Simulator* (BES), according to modern software engineering practices.

Keywords: Building Evacuation, Simulation, Software Framework, Adaptation, Multi-Agent System.

1 Introduction

The characteristics of the individuals involved in an emergency and the way their behaviours adapt in response to changes of their surrounding conditions, have a dramatic impact on the outcome of a rescue operation [1]. Specifically, in a building emergency scenario, the routes through which civilians will be directed towards the external points of collection, and the allocation of rescuers and firemen, are some of the critical decisions that must be made in real-time. These decisions will have to continuously be adapted to unpredicted developments, such as the spreading of a fire, the delayed propagation of information due to faulty communication infrastructure, and the congestion in stairwells. Techniques for quick and optimal decision making have been developed for these purposes, such as [2]. In this paper, we tackle the challenge of designing a simulation tool that combines reproducibility of experiments with a high level of flexibility for a broad scope of models and scenarios. The *Building Evacuation Simulator* (BES) allows easy configuration of the simulated model and rapid incorporation of new components that can be locally developed and

then transparently deployed in either local or distributed environment, in close analogy with the simulation framework *SimJ* [3] [4].

The remaining of the paper is organised as follows. We start with a quick summary of existing state-of-the-art contributions and identify their differences with our work. We continue with the description of the simulated model and the details of the simulator framework. Then, we show how to incorporate new models in our framework and present a preliminary validation of the BES with a simple evacuation scenario in a building with four floors and three stairwells. We conclude with a summary of our contributions and the work we plan to carry out in the near future.

2 Related Work

The application of agent technologies to the study of emergency situation is not new. DrillSim [5], the simulator of human and social behaviour in emergency evacuation [6], and the simulator for crisis management [7], all present state-of-the-art simulation platforms for building evacuation in emergency situations.

DrillSim [5] differs from the BES mainly at its scalability and reusability. The *scalability* is reduced because the simulator is based on a centralised simulation engine that runs only in local environment. Differently, the BES enables the use of distributed environment, and therefore offers a higher degree of scalability. DrillSim also presents a reduced *reusability*. This is due to the structure of the simulation engine and the configurability of the actors in the scenario. The simulation engine includes the simulated geographic space, the evacuation scenario, and the agents. By such design approach, the switching to another implementation simulation engine that might perform better for the given simulation workload, requires considerable reworks. The agents also present limited configurability in terms of decision, motion and health models because their characteristics can be specified only through the parameters of the hard-coded models. Differently, the BES adopts a very modular architecture. It is horizontally based on the layered architecture SimArch and the principle of separation of concerns which maintains separated the parameters that do not affect the behavioural logic of the agents, from the logic itself. By this approach, the BES gains: i) the transparent use of the different simulation engine with no extra effort, such as distributed in the place of a local one, ii) the development and testing of new agents locally before the deployment in the distributed environment, and iii) the testing and parameterisation of new models and optimisation algorithms.

Finally, both DrillSim and BES are augmented reality simulators but in different manner. We have already presented such results and comparisons in [8].

The framework presented in [6] differs from the BES for the grid-based modelling approach of the space, which does not scale very well for large areas, for the execution in local environment only, and for the 3D visualisation that our simulator does not currently provide. Moreover, the two contributions have complementary scopes. Indeed, the work presented here provides a general framework within which one can implement all the modelling aspects defined in [6].

Finally, the authors of [7] concentrate on the modelling of the social interactions, while we focus on the architecture of the simulation framework. In addition, their contribution uses rule-based reasoning and decision-making, which cannot scale in a distributed environment and are computationally demanding even in a local environment. Thanks to its modular architecture, BES can effectively incorporate the reasoning and decision-making modelling that they used in [7].

3 Simulated Model

In a typical building evacuation scenario, the actors involved are the civilians who evacuate the building, the rescuers who collect injured civilians and the firemen who try to extinguish the fire. Following the agent paradigm [9], they need to be independent and intelligent enough to individually decide which resources to use, and how to cooperate or compete for their use. These decisions depend on both their internal objectives and the state of their external world. The human agents are provided with their own personal view of the world, and with their own decision, motion, and health models, which collectively describe their status. As an extension to the typical scenario, one may add an ongoing, dynamic threat which may exist for the agents. Such threats are handled as hazard agents, which do not occupy physical space, but affect the conditions of the simulated world, both the human agents and the actual building.

The different types of agents operate on the simulated physical world, which we model using a set of graphs. The nodes of such a graph represent the physical "*Points of Interest*" (PoI), while the edges represent the available paths between them. PoI may be, for example, the physical location of a fire extinguisher, a door, a desk, or the intersection point between evacuation paths. When a path between two locations is blocked, this is simply represented by the loss of the corresponding links or by a prohibitive increase of the movement cost on them. We followed this approach, because our focus is on largely populated scenarios, the simulation of which should not be slowed down by the modelling of every little detail of the physical world that does not effectively influence the evacuation. Apart from the reduced computational demands, we also benefit from the several existing algorithms for known graph theory problems. For example, most actors that are familiar with a building will use the shortest path to reach their destination, while the rescuers will have to visit all areas of the building in the shortest possible time.

Further to the global graph, which represents the whole of the simulated physical world, we use sub-graphs to define local regions. An agent is always fully aware of all changes that occur in the local region it belongs to. For example, the nodes and the edges modelling a room all belong to the sub-graph of the same local region, because all the agents in the room perceive every change that is taking place in it. We use sub-graphs also for the modelling of dynamic hazards, such as the spreading of fire in the building. Of course, each node can belong to more than one sub-graph, and the nodes of the physical world may be connected with different links on their various sub-graphs, depending on what these represent.

Finally, an important issue that is covered effectively with our graph-based modelling is the effect of congestion in the choke points of the building, such as doors, narrow corridors and staircases. Each node of the physical world graph is modelled as a server with limited capacity and a single queue. When a human agent arrives at a choke point, it either finds it free and is immediately served (crosses the node), or finds it busy and queues up. This approach allows for the integration of more complex queuing models that have been developed in the literature, for human beings moving through congested areas [10].

4 Simulator Framework

The simulator framework is built according to modern software engineering techniques that tend to enable a model-driven approach to the development of the simulators [11]. It is based on the architecture *SimArch* [11] (Figure 1), which organises the simulator software in four different layers: Simulation Model Layer (4), Simulation Components Layer (3), Discrete Event Simulation Layer (2), and Distributed Discrete Event Simulation Layer (1), which is built on top of the *IEEE HLA standard* [12].

Layer 4 is the layer where the simulation model is defined through the declaration of the agents involved in the simulated scenario. Such agents are provided by Layer 3, which in turn uses Layer 2 for communication and synchronisation transparently in both the local and distributed environment. Layer 1 provides a *Discrete Event Simulation* (DES) abstraction on top of the distributed computing infrastructure conventionally identified by Layer 0. This bottom layer does not belong to SimArch but provides the basic services to operate in a distributed environment.

4.1 SimJADE

SimJADE is a simulation framework that extends the popular agent-based JADE framework [13] by introducing an innovative formulation of discrete event simulation systems in terms of multi-agent systems [14]. The most widely adopted DES paradigm is process interaction [15], which presents many affinities with MAS. It is based on independent simulation entities that communicate and synchronise their logical time to carry out the simulation according to the properties of causality and reproducibility. A discrete event simulation system can therefore be modelled as an agent society with a defined ontology, composition, and interaction protocol.

The simulation ontology, named *DES-Ontology*, defines the *DES concepts* (simulation time) and *actions* (DES and simulation life cycle management services) that are used as semantic base for the communications among the simulation agents. The concepts defined by the *DES-Ontology* are:

- *AbsoluteSimulationTime*
- *RelativeSimulationTime*

with “relative” having default semantic “respect to the current time”. Although these two concepts are related by a simple linear transformation, the definition of a relative time concept is included in the ontology because in the simulation community is common practice to use it as parameter type in several DES services.

The *actions* included in the ontology are of two types: the *simulation management services* and the *DES services*. Belonging to the first category are the actions to manage the simulation life cycle:

- *Register agent*: to request the permission to join the simulation society;
- *Registration successful*: to acknowledge the permission in response to a Register Agent request;
- *Remove agent*: to resign the society;
- *Simulation end*: to inform that the society objective has been reached.

Whereas the second group includes:

- *Conditional hold time*: to request an hold for a given simulated time unless any even notification before it;
- *Hold time*: to request an unconditional hold for a specified simulated time;
- *Notify time*: to inform that the specified time has been reached;
- *Notify message*: to inform that the specified event was requested to scheduled for the receiving agent, at the current time;
- *Send message*: to request the delivery of the specified event at the specified time to another simulation entity agent;
- *Wait message*: to request to be wake up when a simulation message is to be notified.

The simulation agent society is composed of two types of agents: simulation entity

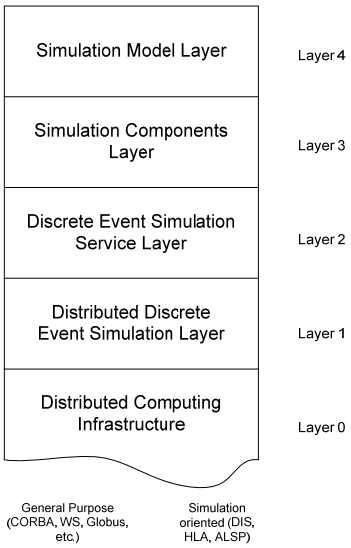

Figure 1 SimArch Architecture [11]

and simulation engine. The *simulation entity agent* incorporates the simulation logic, i.e. the sequence of internal operations and DES service requests, and provides to the developers with discrete event simulation versions of the conventional JADE services, such as doWait and receiveMessage. With this approach, SimJADE brings a model-driven development of the simulation system, since developers are provided with a uniform interface and are not involved with the details of the communication and synchronisation, local or distributed [3], standard agent-based or simulated agent-based [14].

The *simulation engine agent*, which may be unique within the society, collects the request and orchestrates the society according to the properties of causality and reproducibility. It is available in two transparently interchangeable versions, local and distributed. The distributed version is implemented in close analogy with the framework *SimJ* and is based on a HLA-based implementation of layer 1 of the *SimArch* architecture (Figure 1) [11].

The *interaction protocol* is composed of a populating phase and a serve-and-process cycle. In the populating phase, the simulation entities register into the society to be included in the synchronisation mechanisms. The serve-and-process represents the main cycle of the simulation where the communication and synchronisation requests are collected and processed. It relies on the fact that some requests are blocking other are not; and it assumes that while performing a blocking request, the entity agent does not perform other request and waits for a respective simulation message notification. While collecting the requests, the simulation engine schedules proper simulation event handlers to deal with such a request. When the handler is processed because the relative simulation time has reached, it unblocks the entity agent that has requested it.

4.2 Agents: Dynamics and Models

The agents, including their dynamics and the parameterisation, are defined at Layer 3 of the above architecture. Their design is based on the key principle of *Separation of Concerns* [17] that suggests designing components with a minimised in such a way that presents features computer programs into distinct features that overlap in functionality as little as possible. Each agent is defined by a behavioural logic, which specifies the interaction with the external world, and a set of parameters that do not affect the pattern of the logic, according to the design outlines in [18]. By such approach, the cohesion of each agent is maximised to make it reusable across the several values the parameters might assume. A straightforward, but effective, methodology to individuate the candidate parameters comes from the analogy with the physical agents. A *ResourceManager* manages the world model, and the active actors, *human* and *hazard agents*, use and affect the condition of such resources.

The *ResourceManager* coordinates the access to the node and supports the management of the world updates for each agent. Its dynamics are structured in two phases, a wait for an event and the processing of it. The expected types of events are: *WantToMoveTo* and *FinishedMove*. When receiving a *WantToMoveTo* a node, the manager checks whether the node is already occupied by some other human agents. If

the node is busy, it enqueues the request according to a FCFS policy, otherwise it immediately warrants the access by sending an *AuthorisedToMove* event. When receiving a *FinishedMove* event, the manager checks whether other human agents have requested to move on the node. In the affirmative case, the manager authorises the movement of the first agent in the queue. In addition, the *ResourceManager* regulates the updates for the agents. Knowing the position of the each human agent, it determines if a change in the world should be reflected into the individual perception of the world. Similarly, when a human agent moves to a different group of nodes or edges, it provides the updated condition for all the nodes and edges concerning that group.

The *human agents* are provided with their own personal view of the world, and with their own goal, motion and health models which describe their status. The human agents, which move and occupy space in the physical world, present a behavioural logic that is mainly composed of movements because they can only interact with the surrounding parts of the world. Therefore, they must reach the point in the space they want to interact before performing any other type of actions. The basic dynamic is described by the state diagram in Figure 2.

After positioning on the initial node, the human agent state diagram proceeds with a cycle of movements that eventually lead to the final state, which is either a dead agent or the accomplished ultimate goal, such as reaching the point of collection.

After having decided the goal (where to go) and how to reach it, the agent enter the *Reaching the Node* state in which it sends a *WantToMoveTo* event to the ResourceManager through the underlying layer services. It thus enters the state *Waiting for Movement Authorisation* and remains there until the node's ResourceManager authorises the movement on the node. The manager sends the *AuthorisedToMove* event immediately, if the node is free, or when it receives a finished movement message from the agent currently on it. The simulation time between the notification want to move and the reception of the authorisation is the queuing time at the node. The agent authorised to occupy the node holds such position for some simulation time before either moving to another node by starting over the movement cycle or terminating its life if it has reached its ultimate goal – which generally is the external point of collection. However, some of the movements might not be completed in certain conditions of the world because they might require an amount of time greater that the remaining life time of the agent. To include this case, transitions from the states *Reaching the Node* and *Waiting for Movement Authorisation* to the end state are included.

The standard human agent dynamics might incorporate custom sub-dynamics for the specific type of agents present in the simulated scenario. Such sub-dynamics, however, take place when the agent reaches the node and terminates with another movement act, which leads the agent on a different node. For example, a rescuer, who wants to collect an injured civilian on a floor, first reaches the specific point, and then starts the specific dynamics, with which takes charge of the injured civilian, and finally moves towards the exit.

From the above description, it is easy to infer which the parameters that compose the static structure of the human agent because do not affect the dynamics pattern are: *1)* the decision of on which nodes it wants to move, *2)* the travelling time between

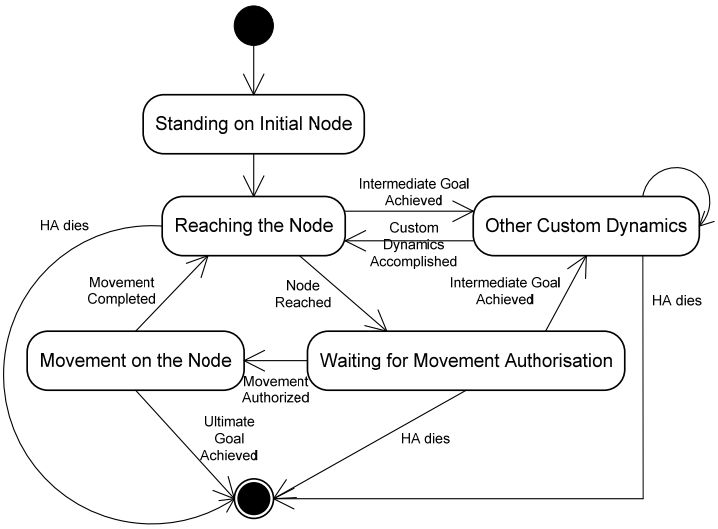

Figure 2 State Diagram of the Human Agent Basic Dynamics

two nodes and the occupation time on a node, and *3)* the life time. Such parameters are therefore kept separated from the agent behavioural logic and can be specified at the time of agent instantiation by passing implementation at configuration time, as in [18]. For each of them, we defined a hierarchy of classes that can be immediately used.

Parameter 1 depends on the agent's personal goal and the strategy adopted to achieves it. From the software architecture point of view, this parameter is specified by the goal "*reach a node in the graph*", which includes a destination node and a decision model that instructs the agent on how to reach it. The goal is defined by the hierarchy show in Figure 4. It can be of type *Simple* or *Composite*, with the latter being sub-classified into *Sequence of Goals* or a set of *Concurrent Goals*. The decision model gives the directions on how to reach. The model is also provided with an update function that operates as observer of the WorldModel and reflects the changes of it on the internal structure, according to the respective design pattern [19]. Such modular structure allows a quick modelling of the different types of human agents in a possible realistic scenario. For example, civilians are provided with a *SingleGoal*, which is reaching the point of collection, while avoiding dangerous paths (with fire for example) or congested path. Differently, rescuers might have *ConcurrentGoals* (e.g. reaching an injured civilian on floor 3, or floor 10) and might have different strategies to reach them depending on their knowledge of the world and the anti-fire protections they wear.

Parameter 2 determines the time duration characteristics of the movement time on the edges and on the nodes. It is defined through the specification of the speed values on both elements as a function of the agent state, the agent characteristics and the physical conditions of the node or edge, on which the movement is occurring. These values can be constant, as in evacuation trainings, or be function of the perception of danger in the event of real emergencies, and can depend on individual panic condition due to fire or other physical factors.

Parameter 3 affects the outgoing transitions between *Reaching the Node* and *Waiting for Movement Authorisation*. If the life time is smaller that the time needed for the movement, the agent dynamic reaches the end state, otherwise it proceeds according to the above specification. This parameter is also maintained updated through the exposition time in adverse physical conditions.

While moving the agent receive updates of the world, which may change its health status, its goals and the ways it reaches. To reduce the number of the event and the complexity of behavioural dynamic, the updates are sent during the crossing of an edge and when completing the movement on a node. In a general, the time spent on the nodes is negligible compared to the time spent on edges and the rapidity in the variations of the world conditions, so that the changes are delayed for human agents in movement on a node.

The *hazard agents*, such as fire-spreading and smoke-spreading, affect the conditions of the simulated world, but do not occupy physical space. They present a simpler yet different simulation dynamic since they do not compete for the access to the nodes. For their peculiarities, they constitute an independent group. The simulator is currently provided with a fire agent, which behaviour can predetermined, with a manual description through XML configuration files, or probabilistic. In either case, the fire intensity on each node and edge is represented as a number between 0 and 1000, and propagates on an extended world model that inherits the structure of the plan and adds edges between physically adjacent nodes. In real scenarios, for example, the fire may propagate not only through doors and along corridors, which can be traversed by human agents, but also through walls and ceilings. The probabilistic model is built according to the guidelines in [20] with further adaptations

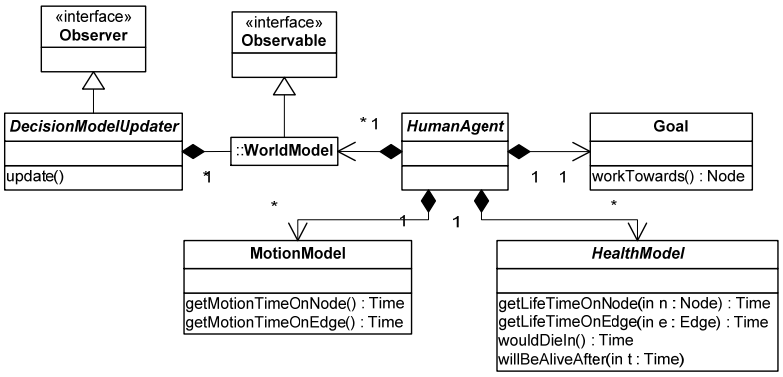

Figure 3 Composition of human agents

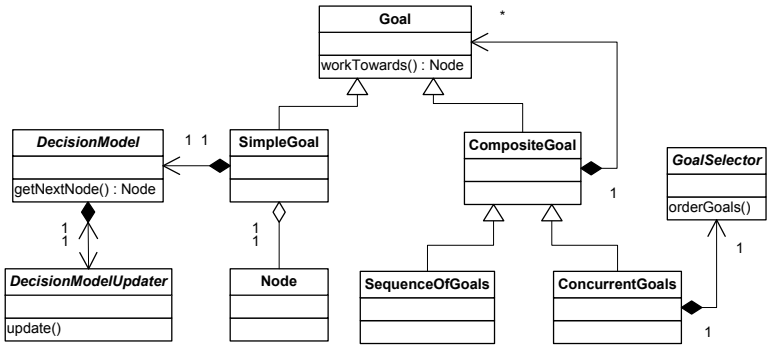

Figure 4 Goal's hierarchy

to include the fire intensity in the spreading dynamics. For further details, please refer to [8].

4.3 Going Distributed

The simulation of a realistic scenario involving thousands of civilians, tens of rescuers and firemen needs computational resource that are at least of polynomial order of the number of simulated agents. Such resource might not be available on a single host, and therefore a distributed environment is needed to effectively carry out behavioural and adaptation studies in such systems. By the use of the SimArch architecture [11], the simulator gains a transparent deployment of agents in either local or distributed environment. This can be indeed done by switching the local version of the simulation engine agent with the distributed version by the use of a HLA-based implementation of SimArch's Layer 1, available from previous works [4] [11].

Despite of the transparent deployment of the agents in either environment, the distributed execution raised new modelling issues. To optimally use the distributed environment, two major issues are to be deal with: how to partition the simulated model over the available computational resources and how to improve the simulator performance through model refinements that do not heavily affect the simulated model.

The model is partitioned in order to exploit the intrinsic parallelism of independent physical subsystems, while meeting the memory constraints on each host and minimising the network workload. For instance, the events happening within a floor or along stairs loosely affect the rest of the system; therefore the simulated world is allocated on independent single area simulator that can be either floor or stairs, each running on a separate host. In addition, since the stairs constitute critical evacuation paths which are going to be traversed by all the agents escaping the building, they might become overcrowded with the number of agents. In that case, a further partitioning could be necessary in order to meet the memory requirements.

A key factor for the performance of the simulator is the amount of data exchanged between the separate simulators. In order to reduce such data, the world model is locally stored and cloned for each incoming agent and then enriched with agent's personal knowledge. In addition, considering that the agent executing locally can interact only with the local world, a condensed view or the remaining world is adopted as compromise strategy to achieve scalability of the simulated world and accuracy decision making process.

5 Incorporating New Models

Separating the behavioural dynamics from the specification of the decision, motion and health models brings good reusability of the human agent basic dynamics, which can easily incorporate new and possibly adaptive models. According to the above schema, we can define the following parameterisation for the civilians. They are provided with a simple goal that consists in reaching the external point of collection. The goal is achieved by computing the best path between the current position and the destination associated on a graph-based decision model. Initially, the graph is labelled with the physical distance between the nodes; however, changes in the world affect the weights according to the decision model updater function. In the case of civilian, this function is defined by the following expression:

$$weight(edge) = \begin{cases} edge.physicalLength & if\ (edge.fire = 0) \\ +\infty & if\ (edge.fire > 0) \end{cases}$$

The civilians' motion model is defined as function of the world and individual motion characteristics. For the floor, the speed on the edges is uniformly distributed in 145 – 155 cm/s [21], and the speed on the nodes is constant and equal to the average 150 cm/s. For the stairs, the speed on the edges reduces to 60 – 80 cm/s, whereas the speed on the nodes remains the same.

The civilians' health model life time is defined as step function. The civilian survives until the physical conditions of the world cross the individual threshold. Currently, the function is taken as dependent only on the fire level on the node or edge being traversed:

$$lifeTime(edge) = \begin{cases} +\infty & if\ (edge.fire < 300) \\ 0 & if\ (edge.fire > 300) \end{cases}$$

A possible extension of such human agent's parameters could include the modelling of other phenomena occurring during the evacuation of a building. For example, to model the competition for the use of the intersection PoIs, the motion model could be defined according to a different function, which could be, for instance:

$$motionTime(node) = \begin{cases} nodeTime & if\ (node.lengthOfQueue = 0) \\ node.lengthOfQueue/10 + nodeTime & if\ (node.lengthOfQueue > 0) \end{cases}$$

Such function indeed models the fact that the traversal of an intersection node deteriorates when there are collision phenomena [6].

The rescuers are defined in a similar way. Their goal is generally of type "concurrent type", since there can be more injured civilians at the same time. The goal selector can be based on physical proximity criteria or decentralised optimisation techniques based on neural networks [2].

The decision model updater is defined according a different function because the rescuers might wear protections, and therefore might able to traverse also more hostile sections of the building. An example of such function is:

$$weight(edge) = \begin{cases} edge.physicalLength & if\ (edge.fire = 0) \\ edge.physicalLength \times (1 + edge.fire/250) & if\ (edge.fire \in (0;250)) \\ +\infty & if\ (edge.fire > 250) \end{cases}$$

The motion model is defined with 170 – 180 cm/s, on the floor edges, and 70 – 80 cm/s on the stairs edges, when reaching injured civilians; and 100 – 120 cm/s on the floor edges, and 50 – 70 cm/s on the stairs edges, when rescuing civilians. The speed on the nodes is maintained the same as for the civilians, since this parameter affects the performance of the simulator and do not significantly influence the statistical results of the simulator.

6 Preliminary Validation

The validation of emergency simulators is generally not possible with direct comparison of data from the real world, because emergency metrics, such as the total or average individual evacuation time, for a specific building often do not exist until some disaster happens; when they happen the priority is not collecting statistics. However, a preliminary validation can be carried out by properly setting the simulator parameters in a verifiable scenario.

We can assume that in a public building with not particularly complex structure and populated only by employees, who are familiar with its layout, evacuees use the shortest physical path to the main exit. For our validation scenario, we consider a public building with four floors, three stairwells and a main exit, which is the external point of collection for evacuees. The building is populated with 80 civilians uniformly distributed over the four floors. The simulation is performed in a distributed manner over eight federated simulators, one for each floor and stairwell, and one for the external point of collection.

For this preliminary validation we do not consider the social behaviour of the evacuees, but we assume that their motion is regular and tidy, without non-adaptive crowd behaviour, and with queuing at choke points being the only reason for them to be delayed. The civilians' motion parameters are set according to the average values provided in [21]. The edge crossing time each time is therefore given by l / s, where l

is the physical length and *s* is the speed of each actor. We assume a global traversing time of 0.3s for each node.

The results of our experiment after several runs show that the average total evacuation time is about 87s. This value is reasonably close to the ideal value of 76s that we computed using the optimistic mathematical model presented in [21].

As a further validation step, we tested the system behaviour of the simulator by measuring the average evacuation time for different allocations of the 80 civilians in the four floors (Figure 5). In the first experiment (Exp. 1) we place 35 civilians on floor 1, and 15 civilians in each of the other three, while in Exp. 2 we have 35 civilians on floor 2 and 15 in the others, and so on for Exp. 3 and 4. As we expected, the closer the majority of the civilians are to the external point of collection, the lower the average evacuation time, despite the effect of increased queuing times at congested areas.

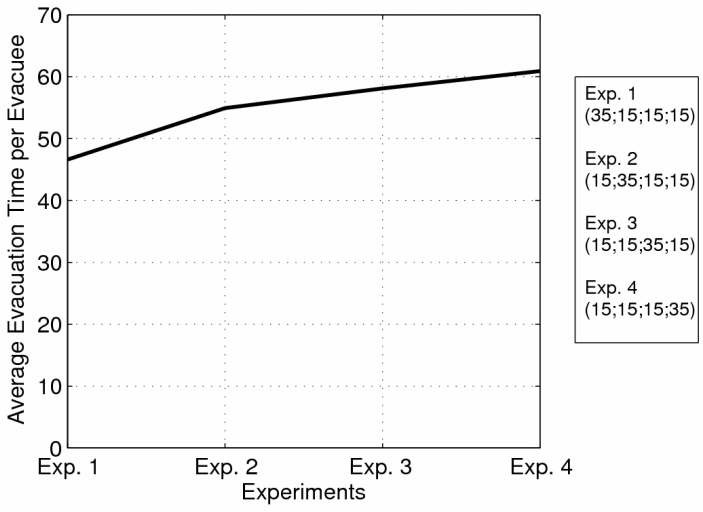

Figure 5 Variation of average evacuation time for different distribution of civilians over the four floors

7 Conclusions

The field of disaster management and emergency response can benefit greatly from the use of computer simulation, both to evaluate evacuation plans, standard policies, and decision mechanisms, and also to suggest optimal actions even during an emergency. However, the systematic investigation of the adaptive behaviour of agents and complex interaction with their physical surroundings in disaster scenarios requires a software framework that allows *reproducibility of experiments* and rapid *extendibility* to new models and scenarios. In this paper, we presented a simulation

framework which meets these requirements, and additionally provides transparent deployment in both local and distributed environments.

The *distributed operation* of our software allows the simulation of largely populated scenarios, and also gains better fault-tolerance and integration with other simulators. As future work, we plan to make use of the latter by integrating the *Building Evacuation Simulator* (BES) with a simulator of larger scope, such as the excellent work on the Robocup Rescue (RR) [22], which deals with disasters at city-level. We envision the agents using the RR simulator to travel across the city and using the BES whenever they enter a building, so as to effectively simulate a disaster at different levels of microscopy.

In this paper we presented the design of a distributed agent-based simulation framework geared towards evaluating adaptive decision mechanisms in building evacuation scenarios. In our future work, we will present such adaptive mechanisms with the use of this simulation framework.

Acknowledgements

This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Systems) project and is jointly funded by a BAE Systems and EPSRC (Engineering and Physical Research Council) strategic partnership (EP/C548051/1).

References

1. C.W. Johnson, "Lessons from the Evacuation of the World Trade Centre, September 11th 2001 for the Development of Computer-Based Simulations", *Journal of Cognition, Technology and Work*, vol. 7, n. 4, Nov, 2005, Springer London, pp. 214 – 240.
2. E. Gelenbe and S. Timotheou, "Random Neural Networks with Synchronised Interactions", *Neural Computation*, accepted for publication.
3. A. D'Ambrogio, D. Gianni, and G. Iazeolla, "SimJ: a Framework to Distributed Simulators", *Proceedings of the 2006 Summer Computer Simulation Conference (SCSC06)*, Calgary, Canada, 2006, pp. 149 – 156.
4. D. Gianni and A. D'Ambrogio, "A Language to Enable Distributed Simulation of Extended Queueing Networks", *Journal of Computer*, Vol. 2, N. 4, July, 2007, Academy Publisher, pp. 76 – 86.
5. V. Balasubramanian, D. Massaguer, S. Mehrotra, and N. Venkatasubramanian, "Drillsim: A simulation framework for emergency response drills", *Proceedings of the 2006 Conference on Intelligence and Security Informatics (ISI 2006)*, May, 2006.
6. X. Pan, C.S. Han, K. Dauber, and K.H. Law, "A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations", *AI & Soc.*, vol. 22, n. 2, Oct, 2007, Washington, DC, pp. 113-132.
7. Y. Murakami, K. Minami, T. Kawasoe, and T. Ishida, "Multi-Agent Simulation for Crisis Management", *Proceedings of the IEEE Workshop on Knowledge Media Networking*, Jul, 2002, IEEE Computer Society, pp. 135 - 139.

8. A. Filippoupolitis, L. Hey, G. Loukas, E. Gelenbe, and S. Timotheou, "Emergency Response Simulation Using Wireless Sensor Networks", *The First International Conference on Ambient Media and Systems (Ambi-sys08)*, February, 2008, Quebec City, Canada.
9. N.R. Jennings, and M. Wooldridge, "Application of Intelligent Agents", *Agent technology: foundations, applications, and markets*, Springer-Verlag, 1998, pp. 3 – 28.
10. D. Helbing, I. Farkas, and T. Vicsekm "Simulating dynamical features of escape panic", *Nature*, n. 407, Sept., 2000, pp. 487 – 490.
11. D. Gianni, A. D'Ambrogio, and G. Iazeolla, "A Layered Architecture for the Model-driven Development of Distributed Simulator", to appear in *Proceedings of the First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems* (SIMUTOOLS08), March, 2008, Marseille, France.
12. IEEE 1516, Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules.
13. F. Bellifemine, G. Caire, and D. Greenwood, "Developing Multi-Agent Systems with JADE", Wiley (2007).
14. D. Gianni, "Bringing Discrete Event Simulation Concepts into Multi Agent System", *Proceeding of the 10th International Conference on Computer and Simulation (EuroSim-UK08)*, IEEE Computer Society.
15. Richard E. Nance, "The time and state relationships in simulation modeling", *Communications of the ACM*, vol. 24, n. 4, April 1981, pp. 173-179.
16. E. Gelenbe, E. Seref, and Z. Xu, "Simulation with Learning Agents", *Proceedings of the IEEE*, vol. 89, n. 2, February, 2001, pp. 148 – 157.
17. T. Mens and M. Wermelinger, "Separation of concerns for software evolution", *Journal of Software Maintenance*, vol. 14, n. 5, Sept, 2002, pp. 311 – 315.
18. D. Gianni and A. D'Ambrogio, "A Domain Specific Language for the Definition of Extended Queueing Networks Models", to appear in *Proceedings of the IASTED International Conference on Software Engineering* (SE 2008), February, 2008, Innsbruck, Austria.
19. E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley (2000).
20. D.G. Elms, A.H. Buchanan, J.W. Dusing, "Modelling the Fire Spread in Buildings", *Fire Technology*, vol. 20, n. 1, 1994, pp. 11 – 19.
21. J. Pauls, "Calculating Evacuation Time for Tall Buildings", *Fire Safety Journal*, n. 12, 1987, pp. 213 – 236.
22. Robocup Rescue, http://www.rescuesystem.org/robocuprescue/.