Advance Access publication on February 22, 2010
doi:10.1093/comjnl/bxq012

Distributed Building Evacuation Simulator for Smart Emergency Management

NIKOLAOS DIMAKIS*, AVGOUSTINOS FILIPPOUPOLITIS AND EROL GELENBE

Intelligent Systems and Networks Group, Department of Electrical and Electronic Engineering, Imperial College, London SW7 2BT, UK
**Corresponding author: nikolaos.dimakis07@imperial.ac.uk*

We describe a distributed simulation tool which addresses the unique needs for the simulation of emergency response scenarios. The simulation tool adopts the multi-agent paradigm, so as to facilitate the modelling of diverse and autonomous agents, and it provides mechanisms for the interaction of the entities that are being simulated. It operates in a distributed fashion to reduce the simulation time required for such large-scale systems. The simulation tool represents the individuals that need to be evacuated, the resources that contribute to the evacuation including human rescuers, and other active resources and entities which may include robots and which can autonomously interact with the environment and with each other and take individual or collaborative decisions. We illustrate the tool with an application and compare the results for both centralized and distributed execution. Our results also show the significant reduction in execution time that is achieved for different degrees of distribution of the simulator on multiple servers.

Keywords: ALADDIN special issue; distributed building evacuation simulator; smart emergency management

Received 11 August 2009; revised 25 November 2009
Handling editor: Nick Jennings

1. INTRODUCTION

The organization and coordination of emergency operations is usually placed under the responsibility of an incident commander who needs to efficiently allocate the available resources, communicate information to all entities and take decisions regarding the planning and execution of the response. However, the severe time constraints related to managing a response implies that, as far as possible, many of the possible courses of action related to a specific form that an emergency may take must have been designed and evaluated well in advance through modelling, simulation and emergency drills. Simulation is therefore a key technology for the preparation of the response to emergency events. However, if simulation tools are sufficiently fast and flexible, they can potentially also be used to evaluate alternative courses of action when emergencies actually take place. This paper focuses on the distributed building evacuation simulator (DBES) which is a tool that we have designed to support the evaluation of alternative emergency courses of action in confined environments such as buildings or ships. The setting we consider is the evacuation process itself, and the need to provide guidance to evacuees and to emergency personnel. The simulation environment that we have built is distributed, in that it can be run with different components installed on different servers linked via a local network so as to accelerate the execution time of the simulation. The layout of the building or other space is provided as an input via appropriate graph models, allowing for the evaluation of numerous evacuation scenarios in areas such as buildings, ships, oil platforms, small campuses etc. and the emergency itself (e.g. a spreading fire or a series of explosions) can be represented within the simulation framework under changing conditions. The simulation tool can then be used to design and test different evacuation scenarios and the impact they will have on the success of the evacuation procedures.

The rest of the paper is structured as follows. Section 2 presents related work in evacuation simulation and in distributed simulation, as well as in movement decision support systems. The details of the DBES are presented in Section 3 while Section 4 describes a decision support system for building evacuation and evaluates its performance. In Section 5 we illustrate the

DBES in the context of some specific scenarios, and discuss the role of the DBES distributed architecture as a way to reduce the simulation execution times.

2. RELATED WORK

In this section we review the literature related to our problem. We begin with an overview of simulation tools on emergency response and we continue with a description of various distributed simulation approaches. Finally, we present work related to movement decision support systems and algorithms.

2.1. Emergency response simulation

The most widely used emergency response simulation models are EXODUS [1, 2] and SIMULEX [3]. In EXODUS an agent-based approach is adopted that uses a coarse grid to map the geometry of a building or vessel. The grid contains nodes that can be connected with eight other neighbouring nodes through arcs. The simulation calculates shortest distances to the exits and stores that information in the environment. SIMULEX is an agent-based model that also uses a coarse grid geometry. It is a partial behaviour model that relies on inter-person distances to specify the walking speed of individuals, allowing for overtaking, body rotation, sideways stepping and small degrees of back stepping. Other simulation models are EXIT 89 [4], EVACNET+ [5] and EVACSIM [6]. For a thorough comparison of these models, the reader should refer to [7].

There are also other systems that can be used for emergency evacuation simulation. In SGEM [8] the authors model the building as a graph consisting of a series of nodes, each of which represents a building region and holds its own configuration. The direction in which evacuees need to go is determined by a function of the zone in which they are, the adjoining zones and the distance to the final exit. In [9] the authors use an integrated approach, which includes a fine grid network that models the areas of the building that are of particular interest, and a coarse network that models the remaining parts of the building. This integrated approach is advantageous over the coarse network approach because it gives detailed evacuation information for the critical areas. The multi-agent paradigm is used in [10, 11] to simulate human behaviour, such as (competitive) queueing, congestion and herding. A collaborative training simulator for the purposes of emergency preparedness and response is presented in SimSITE [12]. The coordination among the simulation entities and their real-time interaction is achieved using high-level architecture (HLA) [13]. In [14], information theory and graph theory are used to develop an evacuation algorithm in comparable sets of non-identical floorplans, each represented by a unique k-node rooted tree. Finally, RoboCupRescue [15, 16] is one of the most common approaches that brings together many institutions across the world to develop a platform for simulating emergency response algorithms that can be applied to scenarios such as a fire erupting in a city.

All of the above approaches, although some of which provide an infrastructure for distributing components of the simulation such as RoboCupRescue, require that a single application control and orchestrate the simulation. This approach has a number of advantages as follows: (i) all simulated objects are simple references in memory which in effect introduce a very limited overhead to the simulation; (ii) interaction among the objects and the central application is minimal as they are both physically located on the same processor; (iii) the central application constantly has a full view of the simulated environment, allowing it to identify potential 'shortcuts' in the simulation before these appear; and (iv) the central application has full knowledge of the simulated objects' strategies and goals, and thus it is able to pre-calculate and store information, significantly reducing the processing requirements of the simulation.

This centralized approach, however, has characteristics that are not realistic. For instance (i) it uses customized interaction protocols that are specific to the simulation, prohibiting the incorporation of additional features; (ii) before the simulation begins, all of the objects of interest have to be physically located on the same processor; (iii) the strategies and goals for each simulated object must be known to the central application, and this may not be possible for a number of reasons, including that of security and privacy; (iv) security and fault tolerance is minimal, as the central application is *the* single point of failure; and (v) performance can only be improved by either using scenario-specific approaches or high-performance libraries.

Thus our approach alleviates some of these problems via a new distributed simulation platform, each of whose components maintain only a subset of the complete state space. In addition, each simulated object is a dedicated active entity that does not necessarily need to be in the same physical computer location as any other of the entities. Furthermore, all aspects of the simulation (i.e. strategy, goals, internal data structures) are all privately held by each simulated object and this enhances flexibility and security. Finally, our distributed simulation platform adopts a paradigm in which certain parts of the simulation are simulated independently. In the context of building evacuation (which is our main area of study), we consider that simulating the evacuation of a three-storey building can be segmented in three individual concurrent parts, where the events of each floor are independent of one another.

Our earlier work on emergency response simulation has resulted in the development of a simulation tool for building evacuation [17], which is a centralized discrete event simulation framework that models the evacuation of a multi-storey building. A later version [18, 19] used an HLA-based approach to distributed simulation in which the evacuees are modelled as agents and the physical world is represented through the use of a graph. In disaster scenarios that are simulated, a hazard (such as a fire) spreads in a building and the evacuees try to leave using

the best possible exit. Apart from evaluating the evacuation procedure in terms of total evacuation time and civilian injuries, this simulator also represents the behaviour of the evacuees and there is an attempt to take into account their detailed behaviour.

These simulation tools have also been applied to evaluate other specialized applications and smart systems for emergency situations [20]. In other work, we have studied the use of mobile robots equipped with wireless devices that search a disaster area for injured civilians and provide them with connectivity to a wireless sink node [21–23]. Here it is assumed that each evacuee carries a wireless device and the goal of the robots is to maximize the evacuees' connectivity so as to support the evacuation. Moreover, we have also designed a distributed decision support system that operates inside a building during a disaster and provides its occupants with directions regarding the best exit [24]. A brief description of this system is also given in Section 4.

2.2. Distributed simulation

There are two main approaches regarding distributed simulation: *conservative* and *optimistic*. Both perceive the simulation task from different angles and are characterized by their own advantages and disadvantages.

The term 'conservative' refers to the fact that this approach ensures causality of the simulated events by exchanging time-stamped messages among the distributed agents, prior to every event execution, thus introducing no errors. All agents negotiate for the earliest event, and the appropriate event is executed. One of the pioneering work was done by Chandy, Misra and Bryant [25, 26]. Other early conservative approaches [27] are the blocking table algorithm [28], SRADS [29], appointments [30], feed-forward [31], conditional knowledge [32] and bounded lag [33].

The optimistic simulation scheme on the other hand makes a very light use of the communication channels between the distributed agents, exploiting the intrinsic parallelism of different simulation parts. This approach is significantly faster than traditional conservative ones, however, it is much more prone to errors, as each agent is operating independently without prior knowledge of other potentially erroneous events. To alleviate this inherent disadvantage, the optimistic simulation approach requires synchronization schemes and rollbacking mechanisms to correct errors when they appear.

2.2.1. *Time Warp*

Time Warp [34, 35] is one of the most prominent protocols in optimistic distributed simulation. One of its key features is that it employs a rollback mechanism to correct errors in the sequence of the simulated events. If an event that refers to the past is received, then the agent rolls back to the latest error-free saved state in the current simulation history, and notifies the other agents, as this error might affect them as well. After rolling back, the simulation continues from that point. To achieve this, each agent has to save its operational state at regular intervals so that it will be able to roll back to it.

In the original Time Warp algorithm, as soon as an agent identifies a causality error, it immediately initiates the rollback sequence to the simulation network, a behaviour called *aggressive cancellation*. Lazy cancellation [36] tries to improve this by estimating if the notification to all agents is necessary, thus reducing the number of communication messages. To achieve this, the agent that determined the error alone initiates the rollback procedure, until the local clock reaches the latest valid time-stamp prior to receiving the out-of-order message. If the re-simulation after the rollback produces the same virtual time, then no further action is needed. This avoids the unnecessary cancelling of correct events but also increases the memory consumption. Lazy re-evaluation [37] tries to minimize the extent of re-simulation after an error, by comparing the intermediate states during a re-simulation with the state just before the error appeared. If such a case is found, then the agent 'jumps' forwards in time, disregarding the remaining events. However, this scheme has increased memory and computation requirements compared to the original one.

The work in [38] attempts to further reduce the amount of messages that are generated during an error, by requiring only the agent that determined the error to initiate the rollback process. This approach eliminates the problem of cascading rollbacks and 'echoing', and results in faster simulation. The authors in [39] present an approach that uses reinforcement learning techniques, also known as simulation-based dynamic programming. Instead of assuming an optimal control strategy, the goal of reinforcement learning is to find the optimal strategy through simulation by utilizing a value function that captures the history of system feedbacks, requiring no prior knowledge of the system.

Other types of optimizations include [40] which investigates adaptive mechanisms for saving the current state of an agent, taking the potential rollbacks into account. Also, [41] explores an adaptive state saving policy to reduce the complexity of a rollback when an error arises. Another approach is to use message aggregation techniques in an effort to optimize the communication process among the agents [42]. In [43], the authors discuss the option of using consistent global checkpoints to synchronize the processes of a distributed simulation during the rollback procedure, allowing to improve the simulation performance and to carry out a more suitable memory management. They present a new optimistic protocol that uses consistent global checkpoints. Finally, in [44], the authors analyse the performance of the Time Warp protocol in cases of limited memory.

2.2.2. *Modern standards*

In addition to the distributed simulation protocols that were presented in the previous section, there has also been significant effort in producing standards for distributed simulation. HLA [13] provides a general framework within

which simulation developers can structure and describe their simulation applications. In particular, the HLA addresses two key issues: (i) promoting interoperability between simulations and (ii) aiding the reuse of models in different contexts. Although the HLA is an architecture, not a software platform, the use of runtime infrastructure (RTI) software is required to support operations of a federation execution. The RTI software provides a set of services, as defined by the Federate Interface Specification, used by federates to coordinate operations and data exchange during a runtime execution. While HLA is one of the leading standards for distributed simulations, it is coupled with issues of poor scalability, without addressing semantic interoperability. Furthermore, it is a very complex standard and time-consuming to adopt and use.

Distributed interactive simulation (DIS) [45] is a standard used mostly in military simulations. It is used to integrate heterogeneous simulators with different fidelity levels that are geographically dispersed. An application is then used to receive, process and forward the DIS data across the WAN. While DIS is a much lighter and simpler protocol than HLA, its inherent limitations require proprietary customizations and modifications, limiting its reusability. Finally, DIS is highly focused on military training simulations.

2.3. Movement decision support

The problem of finding the best path to a specified location in order to guide humans towards an exit or an area of interest has been approached in various ways.

An algorithm that aims at navigating a robot using a predeployed sensor network is proposed in [46]. The authors use the value iteration algorithm for determining the direction towards which the robot should move, while the relevant calculations are performed by the sensor network in a distributed manner. They evaluate their approach using a wireless sensor network of nine nodes and a robot that has to navigate from a 'home' node to a destination. The robot is able to navigate inside the network, but the approach is not tested under dynamic conditions since the cost of the links remains static. Moreover, parameters inherent to human navigation during a disaster, such as potential congestion or hazard along paths, are not taken into account.

In [47] the authors use a sensor network to navigate a flying robot. The path calculation algorithm is based on a routing protocol for sensor networks. The robot gathers lists of path segments from multiple sensors as it moves and is able to assemble the entire path. A network of 54 wireless sensor nodes was used for the experimental evaluation of the approach. The nodes were positioned in a grid topology while a robot helicopter, equipped with a wireless sensor, hovered over them. The helicopter was able to successfully follow the paths suggested to it by the sensor network. The proposed system was extended for guiding humans, but the approach consisted of only one human and twelve sensors positioned inside a building. The approach was not evaluated for larger building occupancies and dynamic conditions such as the spreading of a hazard.

A system composed of a sensor network that navigates a user to a goal region, by avoiding hazardous areas, is presented in [48]. The approach relies on artificial potential fields. The dangerous areas are represented as obstacles, which generate a repulsive potential. The goal area generates an attractive potential that pulls objects towards it. The authors propose three algorithms for calculating the potential field, computing the safest path to the goal and navigating the user along it. They evaluated the system using a testbed of 50 wireless nodes. Various network topologies were used, with different positions of goal nodes and obstacles. The focus was on the network adaptability to environmental changes, such as the time needed for the network nodes to obtain the shortest path to the goal. However, during a disaster it is very probable that the size of the dangerous areas will not remain unchanged. The authors have not taken into account the case of a dynamic hazard that spreads in different locations. Moreover, the system is not tested in a scenario that incorporates a large number of evacuees.

Finally, [49] presents a distributed navigation algorithm for emergency situations. The approach is based on a routing protocol for mobile *ad hoc* networks. The algorithm sets a low altitude value to sensors located near exits. Sensors that are away from an exit have higher altitude value. The distance metric used is hop count. When an emergency is detected, the respective sensor sets its altitude value to a large constant and broadcasts a message to the network. The authors tested their proposed method using simulations that included two sizes of sensor networks (100 and 2500 nodes), various emergency locations and different network topologies. They used a 20 mote testbed, where emergency events were simulated by light readings, to evaluate their approach. In all cases, the proposed algorithm was able to find paths towards the exit that avoided hazardous areas. Nevertheless, the simulations do not take into account the spreading of a hazard as they assume that the size of the emergency area remains the same. The system's performance in scenarios that include evacuees' movement inside the area is also not evaluated.

3. DBES

The DBES presented in this paper is a tool that simulates emergency response scenarios taking place inside buildings. It is built on top of the JADE platform [50], following the multi-agent paradigm, modelling all simulated entities as dedicated agents. These entities are able to interact with, search for and subscribe to one another. The communication between them is done using a well-defined communication ontology and all interactions follow the FIPA standards [51]. DBES is able to conduct sequential simulations and provide a summary of the results, or can act as a demonstrator for each individual simulation, as seen in Fig. 1. The latter approach allows the user

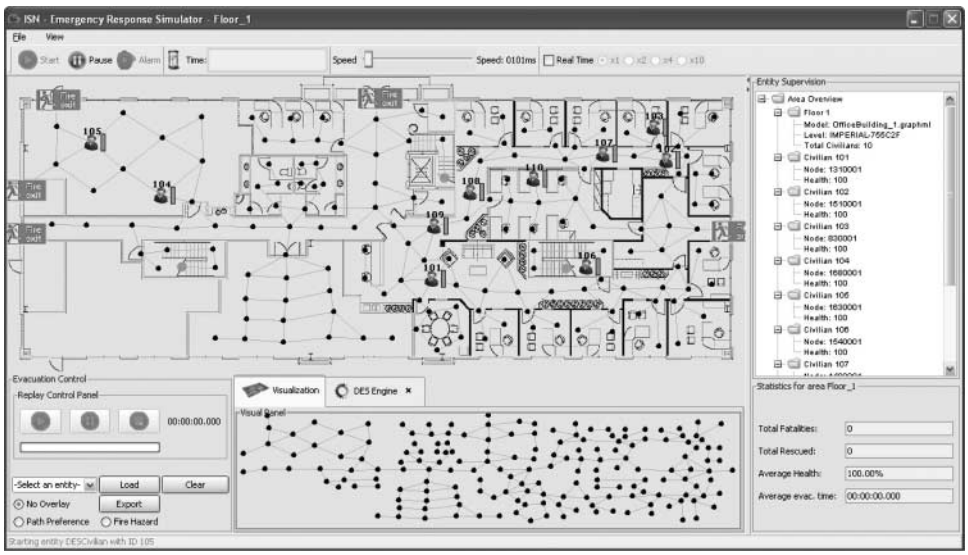

FIGURE 1. The graphical user interface of the DBES simulating the evacuation of 10 entities from the ground floor of an office building.

to interact with the simulation scenario and alter the simulation parameters (e.g. to initiate a fire in a specific location). DBES is also used as a tool to evaluate algorithms for evacuation and resource allocation in emergency response scenarios. In the following paragraphs we elaborate on the building blocks of the DBES and highlight its most important features.

3.1. Architecture

DBES considers only two types of agents: *simulators* and *simulated agents*. The simulated agents are active agents that implement a specific strategy, such as evacuating a building by going to the nearest exit, searching for other agents in a building or locating and transporting agents to the exit. Each strategy is private to the agent itself and at every point in time it determines the agent's next action. Each action is modelled by an event, which is described by a time-stamp and the event description such as $event := \langle t, type \rangle$. Simulators on the other hand are a different type of agents. They are passive in terms of how they affect the simulation evolution. Their main task is to receive a list of events that each agent is committed to simulate in the upcoming future, sort them in an ascending order of time-stamp and trigger the appropriate agent when it is its time to make the next action. Prior to be notified, each agent receives information about its line of sight, so as to be able to determine if its action is feasible or not (to avoid, for example, a possible collision). After determining the output of its strategy, the agent notifies the simulator which in turn makes a final validity check about the agent's new state. The component that is responsible for the ordering of the events and determining the next agent to be triggered is the simulation engine.

Using DBES, a simulation scenario can be broken down in smaller ones and can be assigned an individual simulator. The reason for this capability is that in the case of building evacuation, we can take advantage of the temporal and spatial independence of simulated events. For example, if we consider a scenario of a fire in a three-storey building with any number of people in each floor, we can see that the events that take place on the third floor are independent of the events that take place on the ground floor. Similarly, the events taking place in certain rooms are independent of the events taking place along stairs, etc. We could thus create a network of simulators, each of which would simulate the events in a small part of the overall area instead of having a single simulator controlling the whole area. To achieve this, we need a method that is able to model the simulation area in such a way that allows the separating of sub-parts.

DBES uses graphs to model the different building areas. The nodes of a graph reflect specific locations in the simulated area and the edges model the distance between these locations. We consider that this distance is the point-to-point physical distance between each set of nodes. The graphs may contain special nodes, such as *collection points* (e.g. building exits).

Using this graph-based approach, we are able to put more focus in special areas of the simulation and dedicate an individual simulation network for each area separately, instead of simulating the whole area. This allows us to maintain a smaller graph, which improves the performance of the simulation, especially in the case of operations that take place on the graph such as finding the shortest path or solving connectivity problems.

3.1.1. The simulated agents

The simulated agents are members of our platform and are given individual characteristics. We consider that all our agents are

honest and have no incentive of providing wrong information intentionally. However, a certain level of control is in the hands of the simulators: after receiving an agent's decision, a simulator is able to determine if the action is valid (for instance, an agent can only move to adjacent nodes).

The agents' skill-set is differentiated into various levels which forms the agent hierarchy. At the lowest level, the agents are given functionalities to interact with the underlying agent framework compile messages given a certain ontology (described in Section 3.1.3), etc. Higher levels comprise more sophisticated functionalities such as determining the next action given a current state. Apart from their agent front-end, which they use to interact with each other and with the rest of the simulation platform, they comprise key elements that refer to their nature and to their operational level:

State: The state of each entity is reflected by its location, health level and goal, which is considered to be a specific node of the graph. As the simulation evolves, these parameters are affected by the environmental conditions, for example the hazard of a room in the case of a fire. This information is private to the agent itself, but can be forwarded to the simulator as well, for visualization reasons, or to extract statistical information.

World perspective: Each entity has its own world perspective, which initially is an estimate of the overall graph model for the whole simulated area. As the simulation evolves and the entities traverse the graph, they update their internal perspective with more up-to-date information about the area in the agent's proximity. This updated view is the basis for their next decisions.

Functionality/purpose: The entities participating in the simulation have different functionalities. For instance, a possible evacuation scenario involves civilians who wish to evacuate the building and reach predefined collection points. At any time, they select their next actions so that they move towards the closest exit. In addition, robot agents who search for civilians inside the building could also be present.

Behavioural model: Our entities maintain a behavioural model which comprises mobility and health behaviours. In our simulations we consider that these models pertain to moving speeds and how any form of danger affects the health of the agent.

After each action is completed, the simulated agents update their world perspective and, by consulting their behavioural model, they recalibrate their internal state. The decisions about their next action are based on their functionality, which we assume does not change during the simulation. At the final step, the updated information is forwarded to the simulator which in turn notifies the next entity. A snapshot of an agent's sequence diagram is shown in Fig. 2.

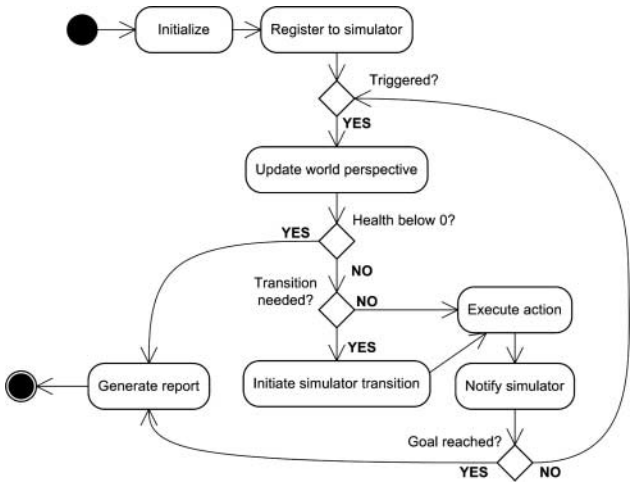

FIGURE 2. The activity diagram of a simulated agent in DBES.

3.1.2. *Standard communication protocols*

As proposed in the FIPA Abstract Architecture Specification [51], the information exchange between agents is completely based upon a specific communication ontology in order to ensure that the semantic content of tokens is preserved across agents. Distributed development of multiple agents and the necessity to understand each other increase the importance of a common semantic concept. Together with a FIPA-compliant implementation of the agent communication and an agent coordination mechanism, the communication ontology forms a sophisticated technique for the collaboration among agents [52].

Our interaction models follow in their majority the FIPA-Request protocol, a standard that has been proposed by FIPA. Initially the sender makes a request, which the recipient acknowledges with a refuse message if the message was not valid, or with an inform message if the message has been accepted. Both messages act as an acknowledgement and hence the initiating agent knows whether its request has been received and what its outcome was.

The reason for adopting this protocol, instead of simply sending a message without an acknowledgement, is the fact that all our simulation entities, including the simulators themselves, are located in different computers. Notifying a simulated agent that it is its time to act, without pending on its acknowledgement, would deteriorate the consistency of the simulation results in the case of a failure in the computer in which the agent resided. Introducing this additional handshake step prohibits unexpected behaviours during the simulation, with a very small communication overhead.

3.1.3. *Ontological messaging*

Communication in DBES is achieved by the use of ontologies. The platform distinguishes between two communication channels: (i) interfacing with the simulation engine and (ii) interacting with other agent-members of the simulation

network. The formulation and exchange of messages between a simulated agent and the simulation engine is done at the lower level of the agent hierarchy. In this manner, the simulated agents and their interaction with the simulator is transparent to the user of the simulation platform. Each ontological layer comprises concepts and actions for a certain type of events. For instance, the ontology which refers to the interfacing with the simulation engine contains events such as Movement, Wait, Exit, Die etc., whereas the ontology for interfacing with other agents contains events such as MakeEvent, RegisterToSimulator, UnregisterFromSimulator, SubscribeTo etc. The overall communication ontology encapsulates in a hierarchical fashion all ontological layers and allows for complex communication exchanges: A MakeEvent which contains a Wait event.

Appropriate mechanisms have been integrated so that agents only receive messages that are of interest to them, in order to reduce the unnecessary overhead. Moreover, the ontologies have been designed in such a way so as to facilitate the reusability of messages. This feature has been used in a plethora of occasions in our platform, especially when integrating additional types of simulated agents.

A fragment of the communication ontology used in DBES is depicted in Fig. 3. The communication ontology uses concepts of the core ontology as message content in the answers to of requests, whenever artefacts are required, which are part of the common domain. Moreover, it extends the core ontology by tokens, which are specific to agent communication and not defined in the core ontology, particularly agent actions for requesting services and specialized result classes. Each simulated agent class can define its own communication ontology and use it appropriately. DBES provides an initial set of well-defined concepts and actions. Moreover, an additional ontology has been designed, which encapsulates the actions and concepts for supporting the simulation, such as the notification of the next entity or the type of action that is to be executed.

The message handling of the agent framework consists of several parts and is based on the DBES communication ontology. A basic abstract class for all agents provides methods for creating, sending, receiving and decoding messages, which are strictly based on the semantic entries of the communication ontology. Furthermore, these methods, together with additional initiator and responder classes for submitting and receiving

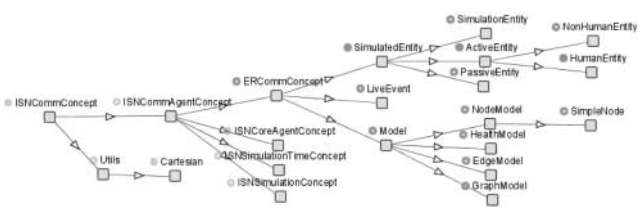

FIGURE 3. A fraction of the communication ontology structure of DBES.

messages, ensure that the agent communication is strictly compliant to the FIPA interaction protocols and communicative acts. The application of FIPA protocols has proved to be a very significant factor in detecting a large number of errors which are expected to arise in such systems, despite the fact that it introduces an additional overhead due to their handshake nature. For example, assume that a simulator agent grants access to an agent to execute its next event. If this agent dies in the process, the simulator will not be in a position to realize that, and might continue the simulation process, inevitably causing causality errors.

The benefit of using this approach is that we are able to simulate a number of different scenarios using the DBES. For instance, we have investigated scenarios in which a group of robots moves inside a site looking for trapped civilians and forming a wireless network [23] or a group of rescuers move inside a multi-storey building in an effort to transport injured civilians. The DBES also allows for the parallel simulation of more complex scenarios in which all these entities co-exist, as each strategy and each set of behaviours are independent among different entities and are not required by the simulation network.

3.1.4. Integration with a sensor network

An important part of an intelligent building is its ability to monitor the environment and detect the presence of hazards. Rudimentary versions of these abilities are the familiar domestic smoke detectors, or the more advanced devices present in office buildings. Future buildings, however, will be monitored by distributed wireless sensor networks (WSNs) capable of processing the data obtained by their sensors, communicating independently of the building infrastructure, and playing an integral role in evacuation procedures in the presence of hazards. In order to aid the development of such systems, we have integrated a real WSN, which consists of 34 telosb motes, into our distributed evacuation simulator. In addition to their wireless capabilities, the motes are connected to a wired network, which allows them to be programmed and also facilitates two-way communication between the motes and the simulator.

Each mote has a corresponding agent present within the simulator. These agents relay the intensity of the hazard in the simulated location of a sensor to a real sensor, which then samples these values and communicates them to a common data-sink. The data-sink also has a simulator presence, in that each floor of the building receives updates on the state of the hazard via the network. The agents also enable other forms of integration between the real sensors and the DBES, including mobility, and allow us to study the impact of network properties (such as delay and packet loss) on the outcome of the evacuation.

The distributed nature of the DBES enables the augmentation of the simulator with a real sensor network. In order for this interaction to be possible, the simulator must run in real-time. It is possible to slow down the advancement of the event-driven simulation to approximate real-time operation, by waiting for an appropriate amount of time between events. There must,

however, be guarantees that the simulator is capable of operating faster than real-time at all times. This is not possible when using a single desktop PC simulating a large building, but can be ensured by distributing the simulation over a sufficient number of machines.

3.2. Simulation approach

Our simulation approach is based on discrete event simulation techniques, which we have enriched with additional features to match the complexity of our simulation scenarios. As soon as an agent wishes to move to a location which is not modelled by its current simulator, then it moves to the next one and continues it progress as before. During this movement, the agent maintains its state completely and is registered to the new simulator as a new agent. To further clarify this, consider Fig. 6 which shows a three-storey building. If each floor is controlled by a different simulator and an agent moves from the second floor to the ground floor, then it has to inform the ground floor simulator of its arrival. At the same time, the second floor simulator should remove this agent from its agent list and manage the remaining agents. Both steps are done when an agent changes simulators. In our approach, simulators interact with each other only when an agent is to be moved from one simulator to another, and not on a regular basis. Agents change simulators when their location is not controlled by their current simulator. This benefits the overall simulation performance as less communication is required, but it also affects to some extent the simulation results. The reason for this is that as soon as an entity wishes to move from one simulator to another, the time when the registration request arrives in the new simulator plays a key role in the evolution of the simulation. Since the simulators have no way of knowing when such an event is imminent, they continue their operations as normal. If the registration that arrives puts the simulated entity in the far future, then the effect on the simulator is minimal and the simulator places the new entity in its queue. However, if the entity arrives from the 'past', then the moment that this request arrives forces the simulator to re-structure its event queue.

To provide an error control mechanism, we use a partial synchronization scheme among the network of simulators, so that at regular intervals their local clocks are synchronized. This approach enables the consistency of the simulation results to be bounded by the synchronization interval, while introducing a very low overhead in the simulation. In practice, a synchronization agent is responsible for setting 'rendez-vous' points in all simulators that participate in the simulation network. These 'rendez-vous' points are references in virtual time. As soon as this point in virtual time has been reached, the simulators assign the next action to the synchronization agent, which reschedules another 'rendez-vous' time in the future. In Section 5 we demonstrate how this scheme maintains a very high confidence level for the simulation results, without introducing a significant communication overhead.

In emergency response scenarios, especially in building evacuation, we expect that as the simulation evolves the agent transitions between simulators become increasingly frequent, since the simulated entities will need to get to the collection points (e.g. building exits). The variation from the expected results is expected to be seen in the later parts of the simulation.

3.3. Unique features

The DBES has been designed to facilitate the integration of new strategies for evacuation and emergency personnel allocation, and to be able to simulate them in parallel without requiring tedious tasks such as manually configuring the simulation scenario.

The core functionalities of the DBES include a pluggable mechanism for loading individual agent behaviours on runtime, depending on each agent's purpose. This makes the DBES platform not only scenario-independent but also independent of the number of different agent classes that can participate in the simulation. Moreover, each agent's strategy is not required to be known in advance as each agent needs to respond to simulator requests following the ontological structure that was previously presented. Furthermore, functionalities for interacting with the underlying framework are also included such as capabilities for searching agents, compiling messages using appropriate FIPA protocols and template matching are also included at the lowest level of agent hierarchy. The simulated agents are given the capability to interface with the simulation engine, by compiling messages using the appropriate ontology, or to interact with other agents in forming and executing a group task. Moreover, the simulated agents may subscribe to be notified when an agent of a given type is instantiated, given some constraints.

To increase the flexibility of the platform, we have designed the simulator to require very little knowledge of the overall simulation environment. Each simulator is only given the graph model of the area that it will simulate and the simulation engine that it will use in order to orchestrate the simulation evolution. In addition to that, the simulator maintains a set of internal variables which reflect the current state of the simulation, and a number of passive communication links for incoming requests. This scheme allows for very high flexibility in the modelling of different areas, regardless of their complexity and size. Finally, despite the fact that each simulator is not aware of the presence of other simulators, it is able to determine its neighbours, that is the simulators that are responsible for adjacent areas. This is achieved by allowing each simulator agent to advertise its edge nodes and request to be notified when other simulator agents have an identical edge node. Events are processed upon arrival with no prior coordination or negotiation.

Our simulation platform is designed so as to perform batch simulations and conduct performance evaluation of emergency response systems and algorithms. However, it can also provide a graphical user interface that combines the majority of its features into a more pleasant visual form. This

interface makes the simulation environment highly interactive as it can dynamically alter the simulation scenario. Moreover, this interface can visualize statistical information, such as the average evacuation time, the path followed during the evacuation and the fire expansion rate, and can also maintain the history of the agents' actions so that they can be replayed later.

Our agents are mobile and move inside the computer network either in an automated fashion, or according to the user's request. In our simulation experiments shown in a subsequent section, we assume that all agents are instantiated on the same computer as the simulator that simulates their current location. As soon as they move to a different simulated area, they physically move to the new computer as well. In our platform, entities move transparently inside the network and maintain their state during this transfer. As soon as the agents reach their goal and no other action is required, they exit the platform.

4. A DECISION SUPPORT SYSTEM FOR BUILDING EVACUATION

We have used the DBES in order to evaluate a system geared towards decision support for emergency situations in buildings. In the next sections we give a description of the system, present the algorithm that is used to provide decision support and evaluate our approach using the DBES. Many of the ideas that we develop here find their origins on earlier work concerning the simulation of smart adaptive agents that take on-line decisions based on locally available information [53–56]. More sophisticated routing techniques are also discussed in [57]. One area that we have not pursued is the possibility of conducting simulations with a visually realistic representation as suggested in [58].

During a crisis, the occupants of a building will not be aware about the safest routes that lead to an exit, and even if they are familiar with the building's layout, a hazard (such as a fire) may be spreading in different locations inside the building [59]. This effectively means that the safest exit path changes as the evacuation procedure evolves. We propose a system that computes the best evacuation routes in real-time and informs the evacuees accordingly. The system consists of a number of decision nodes (DN) and of a network of sensors. The DNs are positioned in specific locations in the building. Their role is to provide directions to the evacuees regarding the best available exit, based on real-time information coming from the sensor network. An underlying communication network links the DNs and the sensors. The recommendations of the DNs are computed in a distributed manner, at each DN, and are then communicated to the evacuees or emergency personnel. This can be achieved in the form of 'smart panel' indicators installed on the DNs, or as information sent wirelessly from the DNs to handheld devices. Figure 4 depicts a simulation scenario where the decision support system is used inside the DBES.

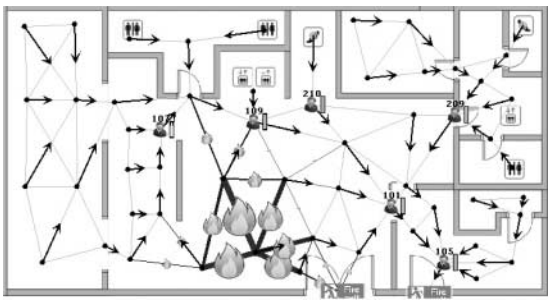

FIGURE 4. The decision support system implemented in the evacuation simulation platform. The arrows correspond to directions from the 'smart panel' indicators.

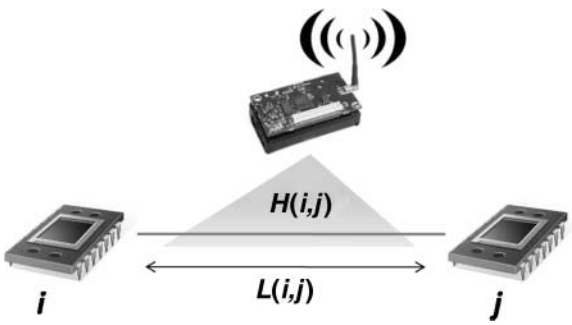

FIGURE 5. A sensor that monitors hazard intensity on a link between two DNs.

A fire is spreading inside while the occupants try to find the best available exit. The arrows correspond to directions from the smart panel indicators positioned inside the building. Each arrow points towards the direction a civilian should follow in order to reach the best available building exit.

When designing the system, we made the assumption of a known building layout, since such a system will be pre-deployed in the building. Based on this assumption, we construct a graph G. The vertices of the graph correspond to locations where people can congregate, such as rooms, corridors and doorways. The length $l(i, j)$ of a link between two vertices represents its physical distance. Each sensor is associated with each link (i, j) and monitors its hazard intensity $H(i, j)$. When there is no hazard present, $H(i, j)$ is equal to 1 and it will increase with the observed hazard. The 'effective length' $L(i, j)$ of a link is defined as $L(i, j) = l(i, j) \cdot H(i, j)$. Each DN is placed at each of the vertices of the graph G. In practice, however, there could be fewer DNs than nodes in G, with each DN being in charge of providing decisions for a set of contiguous locations of G. Figure 5 illustrates how a sensor node can be used in order to determine the value of the effective length between two DNs.

4.1. The algorithm at the core of the system

Instead of using a centralized system to compute the value of the effective length of the paths to an exit, we propose a

distributed decision system for the building evacuation itself which is similar to principles developed in [56, 60], and inspired by the distributed shortest path algorithm [61–63] and adaptive routing techniques such as the Cognitive Packet Network [64]. It is executed by each DN, in a distributed manner, and its output is the next DN that is on the best available path towards an exit.

A DN, at vertex u, stores the following information:

(i) the effective length L of all the links that are incident to u;
(ii) for every neighbour n of u, the effective length of the path y from n to an exit e : $L(n, e)$;
(iii) the effective length of the shortest path from u to an exit e: $L(u, e)$;
(iv) the next suggested DN.

The initial value for $L(u, e)$ is set to zero if node u is an exit, otherwise it is set to infinity. A DN does not keep information regarding the effective length L of the paths towards all the available exits. As the algorithm is executed, this information is propagated from the exits to the DNs. Each DN will eventually select the exit that minimizes the value of the effective length of the path from the node to the exit. The selection of an exit depends on the location of the DN, the locations of the exits and the spreading of the hazard.

When the decision support system is in operation, each DN at u periodically executes the decision support algorithm and provides a suggestion to the evacuees who are in its vicinity. The suggestion is of the form "*go to v*", where v is a neighbour of u.

More specifically, each DN periodically:

(i) *Sends* to every neighbour n of u, the effective length of the path from u to the exit e : $L(u, e)$;
(ii) *Requests* the hazard intensity H from each sensor node that monitors a link incident to u;
(iii) *Calculates* the effective lengths $L(u, n)$,where n is a neighbour of u;
(iv) *Updates* the effective length $L(u, e)$ of the shortest path to the exit as follows:

$$L(u, e) = \min\{L(u, n)+L(n, e) : \forall \text{neighbours } n \text{ of } u\};$$

(v) *Sets* the next suggested DN v as follows:

$$v = \text{argmin}\ \{L(u, n) + L(n, e): \forall \text{ neighbours } n \text{ of } u\}.$$

4.2. Implementation and evaluation of the system in the DBES

We have implemented the proposed decision support system using the DBES, in order to evaluate its performance. In the case where the system is used, each civilian decides its next destination based on the recommendation of the respective DN. When the decision support system is not used, we assume that each evacuee moves according to an initial knowledge of the building structure and becomes aware of the hazard when he approaches a location close to it. Figure 4 depicts the decision support system being used inside the DBES. Each arrow points towards the direction a civilian should follow in order to reach the best available building exit. In the evacuation scenarios we consider the three-storey building depicted in Fig. 6.

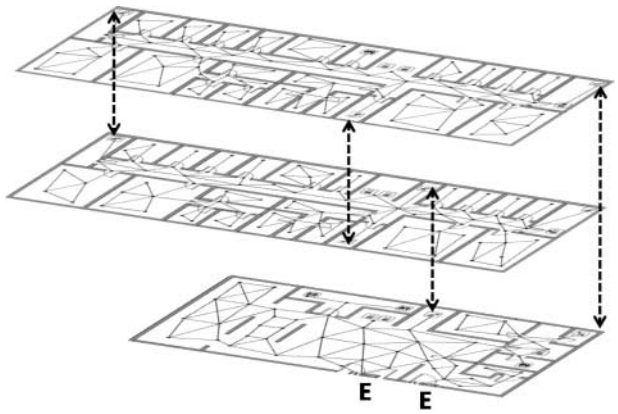

FIGURE 6. The building used in the evacuation scenarios.

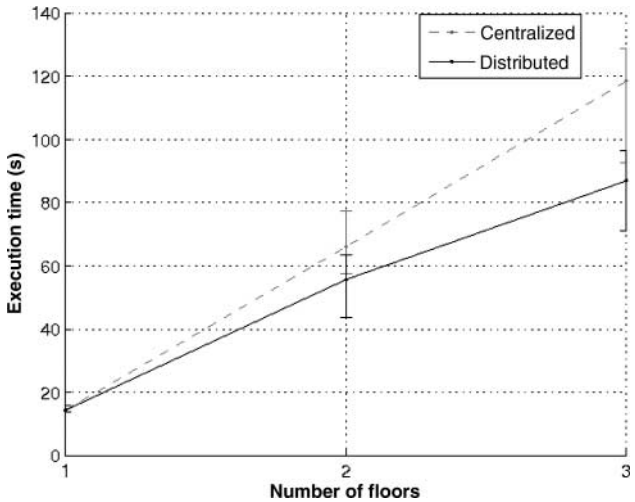

FIGURE 7. Simulation performance in centralized and distributed settings.

The first part of our evaluation investigates how the distribution of the simulation platform affects the execution time of an evacuation scenario. We have tested our system both in a centralized and in a distributed simulation configuration. In the distributed setting, the number of machines is each time equal to the number of floors. Figure 7 shows the simulation execution time for 200 simulation runs. We have run various emergency scenarios, with the number of floors varying from one to three. The number of civilians per floor is 20. As we can observe, by simulating each building floor on a different machine we obtain faster simulation execution times.

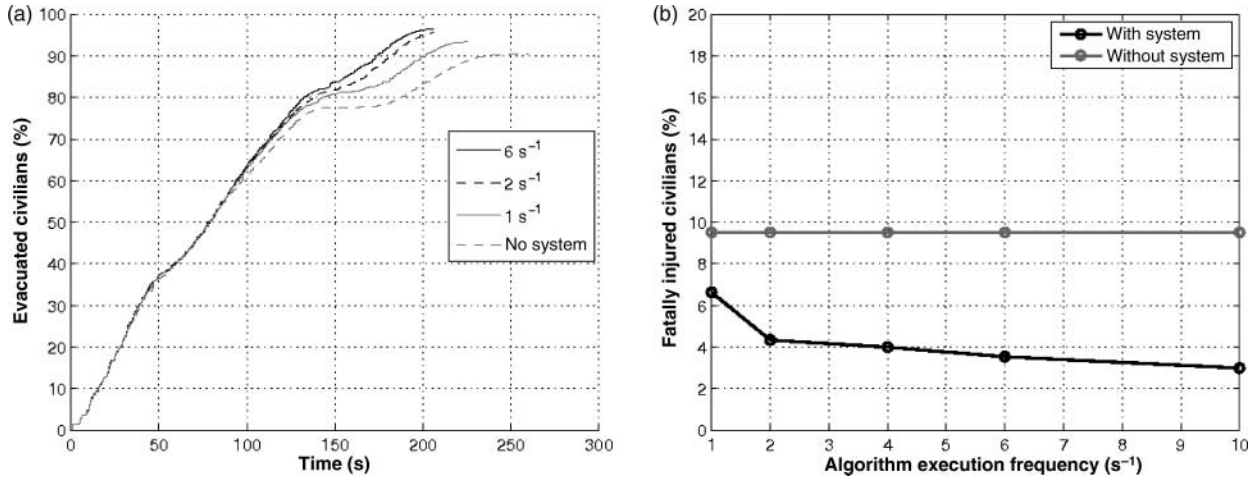

FIGURE 8. Performance evaluation of the decision support system for different algorithm execution frequencies. (a) Percentage of successfully evacuated civilians versus simulation time. (b) Percentage of fatally injured civilians.

The second part of the evaluation measures the efficiency of the system during various evacuation scenarios inside a three-storey building. The building occupancy is 10 civilians per floor and the exit is located on the ground floor, while a fire starts at the first floor of the building We tested our-system for different cases of the algorithm execution frequency. We conducted 200 simulation runs for each case, with different initial civilian locations and different fire-spreading rates each time.

Figure 8a illustrates the percentage of evacuees who have exited the building versus the evacuation time. We can note that when the system is in use, the civilians evacuate the building faster. This is verified by the slope of the respective curves. Furthermore, the use of the decision support system results in a higher percentage of safely evacuated civilians. The percentage of fatally injured evacuees is shown in Fig. 8b. We can again verify that the use of the decision support system minimizes the casualties during the emergency situation and provides better results compared with the case where the system is not present. We should also note that a high value of the execution frequency results in a higher percentage of safely evacuated civilians, as can be verified by Fig. 8a. This is due the fact that the propagation of the changes in the environment depends on the execution frequency. A high value for the algorithm execution frequency results in a more adaptive system which is able to give fast, correct suggestions to the evacuees. Lower execution frequencies, however, result in inferior performance. The changes in the environment are due to the fire that is spreading inside the building. The spreading mechanism is similar to the one used in our earlier work on emergency simulation [17, 18, 65, 66], and is based on a Bernoulli trial model that uses a graph to propagate fire through the building. The speed at which evacuation paths become unsafe depends on that hazard spreading rate and the initial location of the fire. In the case where a fire starts inside a room, the majority of evacuation paths will not be affected (except the ones corresponding to civilians located inside the room). If, however, a fire starts near one of the building exits or along a corridor that belongs to one or more evacuation paths, escape routes will be affected and the occupants should choose another evacuation route in the course of the evacuation procedure. Further work will be useful to evaluate how well such approaches compare with random or misinformed decisions [60], and also to develop appropriate mathematical models that help compute the actual time it takes to reach a safe evacuation point in a random and dangerous context [67, 68]. Some of the distributed computing and autonomic network infrastructure needs of such decision systems are discussed in [69–72]. The optimal allocation of resources, such as emergency equipment or personnel, has also been considered in [73, 74].

5. PERFORMANCE OF THE DISTRIBUTED SIMULATION PLATFORM

In this section we present the results obtained by a series of simulation runs in a network of five Linux 2.5 GHz computers with 4 GB RAM each, using a dedicated network. In these scenarios we simulated a three-storey building with two exits on the ground floor, where each floor was connected with the one above it with three different staircases, as depicted in Fig. 6. Each area operated on a different computer and had a number of agents to handle. The purpose of the individual agents is to evacuate the building by choosing the path towards their nearest exit. The results will demonstrate that the distributed simulation approach we adopted favours the simulation performance without significantly affecting the simulation results. We also compare our results with the case where a single simulator is responsible for the whole area.

Figure 9a and b show the effect of synchronization among the simulators and how it affects both the total execution time and

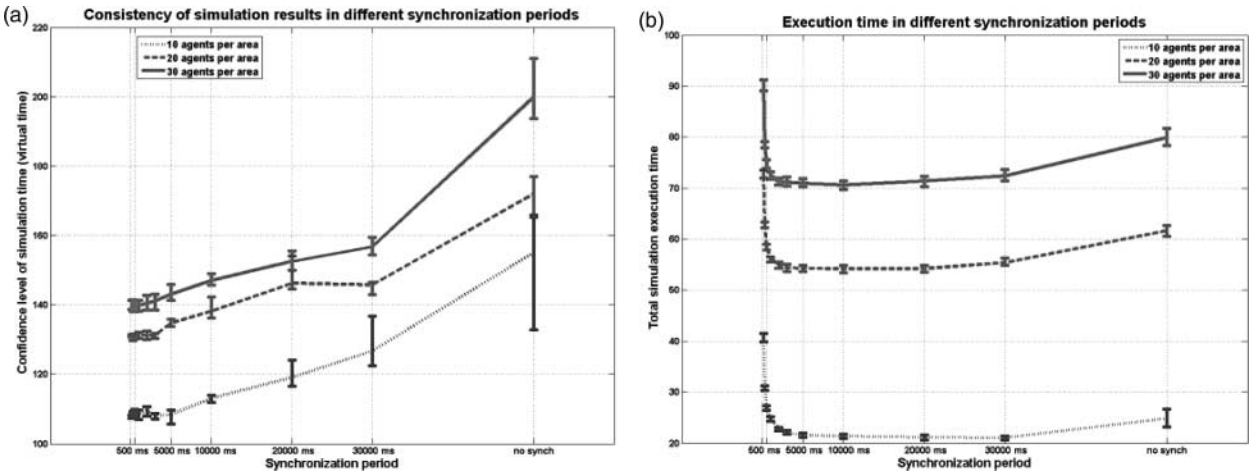

FIGURE 9. How different population settings affect the simulation performance and consistency of the simulation results. (a) Synchronization period versus simulation consistency in different population settings. (b) Synchronization period versus simulation performance in different population settings.

the consistency of the simulation results[1] in different building occupancy settings. These figures verify that scenarios that involve more agents take more time to complete. At the same time, the error that is introduced during the simulation increases as the synchronization period increases, a phenomenon that is clearer in lower-populated scenarios. The reason for this is that during higher-populated scenarios the simulation evolution is done in smaller time steps. External arrivals thus place the new agents in relatively the same time span at each execution. However, when the simulation involves fewer agents, as soon as each agent evacuates the simulation progresses even faster as less events are in the queue. This renders the placement of each new external arrival volatile.

To demonstrate the benefit of distributing the simulation in individual simulators, we present Fig. 10a and b, which depict a more complex scenario involving a five-storey building but having various simulator allocation settings: one simulator per computer (which is a fully distributed setting), three in the same computer and two on others etc. We can clearly see that there is a linear performance increase, something expected as we are now taking advantage of spatial and temporal independence of simulation events. At the same time, the error in the results is always lower than 3.5% as seen in Table 1, and was only experienced in one of the scenarios with 10 agents per area. As explained before, we expect that as the simulation gets more complicated, this error will drop significantly since scenarios involving more agents make the simulation progress in tighter time steps and the possible error caused by an external event is reduced. Fewer agents make the simulation progress faster, as the computational load is decreased and the potential error is increased.

We should also note that synchronization at regular intervals improves the simulation performance. This is explained by the fact that if all simulation agents and simulators are operating on a single computer, then this computer has to control the entire load that is generated during the simulation. However, since the simulators operate in parallel and not in a sequential fashion, the transmission and reception of a message from any simulated agent is affected by the overall load. Having no synchronization scheme would render the consistency of the simulation results highly variable, and each individual simulation execution unreliable.

However, in the case of a too frequent synchronization scheme, i.e. every 100 ms in virtual time, the simulation performance resembles the conservative approach as the simulators spend more time synchronizing than serving simulation-related tasks. This is clearly seen in Fig. 9b, where we simulate 30 agents per building area. Having a 500-ms synchronization scheme would require a 90-s simulation run, while a 20-s synchronization scheme requires 71 s, increasing the performance by 20%. On the other hand, further increasing the synchronization period has negative effects as it resembles the no-synchronization scheme.

The errors in the simulation are related to the fact that as soon as an agent wishes to move from one simulator to another, then for a period of time this agent does not belong to any of the simulators but is in the process of registering to the new one. The time of arrival of the registration message significantly affects the consistency of the result. As seen in our results, there is an optimal selection of a synchronization period, which in our case is every 5–10 s.

The simulation performance and the level of accuracy when having a distributed simulation rather than a centralized one

[1]Consistency of the simulation results refers to how consistent each simulation run is. The confidence level that is shown in the figure illustrates the range of the results.

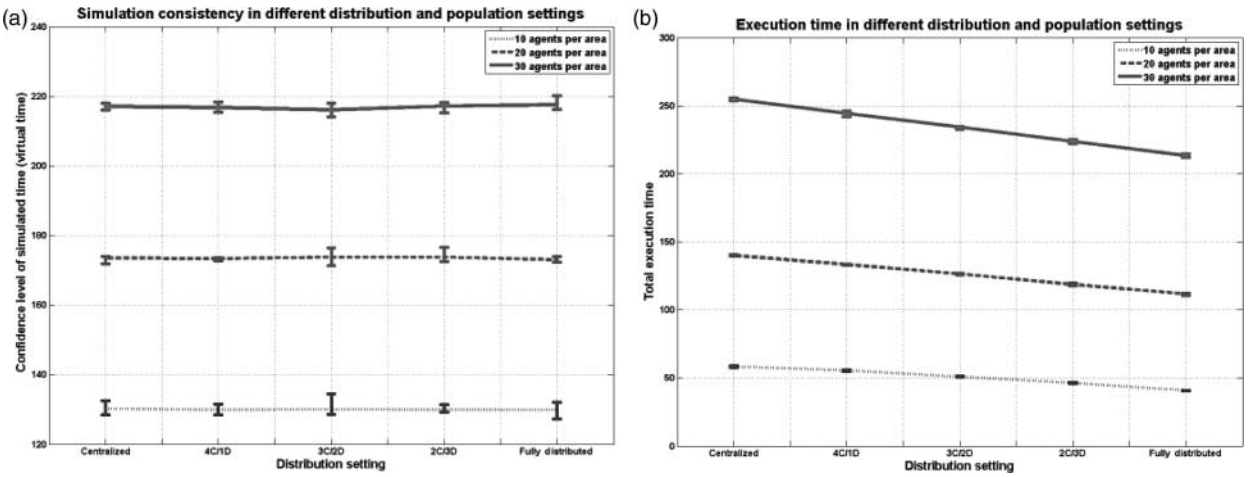

FIGURE 10. How the distribution setting affects the simulation performance and consistency of the simulation results, in different population scenarios. (a) How distributing the simulated entities improves the simulation consistency in different population scenarios. (b) How distributing the simulated entities improves the simulation performance in different population scenarios.

TABLE 1. Summarizing the simulation results in different population and distribution settings, compared to having a single simulator controlling the total area.

	10 agents per area (%)		20 agents per area (%)		30 agents per area (%)	
Distribution setting	Speed	Max Error	Speed	Max Error	Speed	Max Error
Centralized	–	1.8	–	0.98	–	0.46
Four centralized one distributed	5.15	1.25	4.84	0.38	4.15	0.77
Three centralized two distributed	12.95	3.41	9.76	1.5	8.08	0.93
Two centralized three distributed	20.78	1.15	15.2	1.58	12.15	0.88
Distributed	30.2	1.98	20.21	0.56	16.22	1.17

is shown in Fig. 10a and b. Our simulation scenarios involve a five-storey building and three different population settings: 10, 20 and 30 civilians per floor. In the first figure we show how the execution time is affected by distributing the simulated agents and simulators in more computers. We evaluate a range of cases, from having all five simulators and the agents in the same computer, to a fully distributed case where all simulators and their corresponding agents on a dedicated computer. Performance-wise, increasing the level of distribution improves the simulation performance as more independent events can be simulated at the same time, by independent simulators. As far as the consistency of our results is concerned, they remain bounded by the synchronization mechanism, which in this case is every five virtual seconds.

In Table 1 we present the results that were obtained after 1000 simulations of a five-storey building with different population settings in an evacuation scenario. In this table we show the performance increase in different distribution settings. Comparing the results to the centralized case, we observe an increase to the simulation performance. For example, having two simulators distributed and three on the same computer, when we simulate 20 agents per area, will result in a performance increase of 9.08% with an accuracy of 1.5%. The accuracy of the simulation results is bounded by the synchronization scheme, which in this case is five virtual seconds.

The errors in the consistency of the simulation results are inherent in the simulation approach that we have adopted. To clarify this, let us consider the following case. Our example consists of a network of simulators, each of which simulates the events of a part of a multi-storey building. Each floor is assigned to an individual simulator and each simulator simulates a number of simulated agents, all of which move towards the exit located on the ground floor. Let us assume that one of the simulators receives a notification from an agent that arrives from another simulator (e.g. an agent that arrives to the lower floor via the stairs). Assume also that this agent left the previous simulator at a given simulation time. The agents sends a message to the new simulator, and requests to be put in the simulation queue at a specific point in virtual time. The time of arrival of this

message is crucial to the evolution and thus to the accuracy of the simulation. Since the receiving simulator cannot determine if an incoming agent is about to arrive, it continues its operation. Let us now consider two cases: (i) if the network was not loaded and the processing load of the computer was not high, then the message is received in a timely manner and the scenario evolves in a certain way, and (ii) if the message is received with a delay, then the whole simulation sequence is totally different and the simulation time is significantly altered. If the point in time that the arriving agent is referring to is in the past, then the amount of already executed events are more depending on how late the agent's message arrives. However, even if the point in time that the arriving agent is referring to is in the far future, its presence in the evacuation scenario may alter the rest of the agents' decisions and again alter the simulation evolution.

Moreover, due to the nature of our simulation scenarios, we expect that such agent–simulator interaction is not a highly improbable event, as all entities are expected to reach the area that has the collection points. In the case of having distributed simulators, the only factor that seems to play a part in this accuracy issue is the network load. In our experiments, the simulators were operating on a dedicated network and thus we have achieved a more controlled environment. In such a manner, by simply distributing the simulators and the simulated agents, we are able to gain a 15% performance increase in comparison to a centralized unsynchronized setting, while at the same time we are able to gain much more consistent results.

As a final note we elaborate on the distributed nature of our simulation platform. Our scenarios include a large number of simulated agents that interact with the simulator they operate on and make their appropriate actions. In Fig. 10 we show how increasing the number of simulated entities and simulation areas degrades the simulation performance when run centrally, and how the performance is increased when the same entities are simulated in a distributed fashion. Due to the discrete event simulation approach a larger number of simulated entities slows down the simulation, as more entities are to be simulated. However, even if more simulators are being used, since they operate in parallel, they introduce much higher processing load which eventually degrades the simulation performance significantly. Scattering them in a computer network significantly increases the simulation performance, while keeping the consistency of the simulation results at identical levels, if a partial synchronization scheme is adopted. What is also worthy of mention is that distributed simulation is not a fully parallelized problem, and thus we do not expect a speed-up that is analogous to the number of processors available. Furthermore, it is expected that the computation load is not at all balanced not only in the beginning but also throughout the evolution of the simulation. As all entities need to evacuate from a single position of the whole simulated area, all of the agents will have to move from their position (and thus from their initial simulator) to the exit and the simulator that contains this node. Eventually, this becomes the bottleneck of the simulator. However, we claim that better allocations of parts of the simulated area will generate better results. Finally, in the ideal scenario when all individual areas contain a collection point, then we could expect an even higher speed up.

6. CONCLUSIONS

We have presented a distributed simulation system that has been designed and implemented for building evacuation simulations and emergency response scenarios in confined environments. The underlying principles and the building blocks of the simulation system have been discussed, in particular the system's flexibility to incorporate different types of entities that need to be simulated, their interactions and the strategies that each entity may be using. The possibility of distributing the simulation system on multiple servers has also been detailed. We have illustrated the use of the system via simulation examples and results that show that the proposed approach provides a very good level of scalability that can support a large number of individual simulated agents with simulator subsystems operating in a distributed mode. The simulation of smart entities that use adaptive algorithms has also been illustrated.

In future work we will investigate the incorporation of richer user interfaces, and we will also investigate techniques to improve the performance of the simulation scenario by identifying 'short-cuts' in the simulated events. For the latter case, subsets of simulated events that are independent of each other could be automatically identified during a simulation run, and their execution could proceed in parallel so as to provide high throughput for large numbers of events and simulated entities.

FUNDING

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

REFERENCES

[1] Galea, E. (1998) A general approach to validating evacuation models with an application to EXODUS. *J. Fire Sci.*, **16**, 414–436.

[2] Gwynne, S., Galea, E., Lawrence, P. and Filippidis, L. (2001) Modelling occupant interaction with fire conditions using the building EXODUS evacuation model. *Fire Safety J.*, **36**, 327–357.

[3] Thompson, P.A. and Marchant, E.W. (1995) A computer model for the evacuation of large building populations. *Fire Safety J.*, **24**, 131–148.

[4] Fahy, R. (1991) EXIT89: an evacuation model For high-rise buildings. *Fire Safety Sci.*, **3**, 815–823.

[5] Kisko, T.M. and Francis, R.L. (1985) EVACNET+: a computer program to determine optimal building evacuation plans. *Fire Safety J.*, **9**, 211–220.

[6] Drager, K., Lovas, G., Wiklund, J., Soma, H., Duong, D., Violas, A., and Laneres, V. (1992) EVACSIM — A Comprehensive Evacuation Simulation Tool. *1992 Emergency Management and Engineering Conf.*, Orlando, FL, pp. 101–108. SCS, San Diego, CA.

[7] Kuligowski, E.D. and Peacock, R.D. (2005) *A Review of Building Evacuation Models*. US Deptartment of Commerce, National Institute of Standards and Technology.

[8] Loa, S., Fangb, Z., Lina, P., and Zhic, G. (2004) An evacuation model: the SGEM package. *Fire Safety J.*, **39**, 169–190.

[9] Yuana, J., Fanga, Z., Wangb, Y., Loc, S. and Wang, P. (2009) Integrated network approach of evacuation simulation for large complex buildings. *Fire Safety J.*, **44**, 266–275.

[10] Bo, Y., Cheng, W., Hua, H. and Lijun, L. (2007) A Multi-agent and PSO Based Simulation for Human Behavior in Emergency Evacuation. *Int. Conf. Computational Intelligence and Security*, Harbin, Heilongjiang, China, December 15–19, pp. 296–300. IEEE Computer Society, Los Alamitos, CA, USA.

[11] Pan, X., Han, C.S., Dauber, K. and Law, K.H. (2007) A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations. *AI Soc.*, **22**, 113–132.

[12] Liu, K., Shen, X., Georganas, N.D., Saddik, A.E. and Boukerche, A. (2007) SimSITE: The HLA/RTI Based Emergency Preparedness and Response Training Simulation. *DS-RT'07: Proc. 11th IEEE Int. Symp. on Distributed Simulation and Real-Time Applications*, Chania, Crete Island, Greece, October 22–24, pp. 59–63. IEEE Computer Society, Washington, DC, USA.

[13] Dahmann, J. and Morse, K. (1998) High Level Architecture for Simulation: An Update. *2nd Int. Workshop on Distributed Interactive Simulation and Real-Time Applications*, Motreal, Canada, July 19–20, pp. 32–40. IEEE Computer Society, Los Alamitos, CA, USA.

[14] Livesey, G.E. and Donegan, H.A. (2008) Using classical graph theory to generate non-isomorphic floorplan distributions in the measurement of egress complexity. *Math. Comput. Model.*, **48**, 1–10.

[15] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I. and Osawa, E. (1997) RoboCup: The Robot World Cup Initiative. *AGENTS'97: Proc. First Int. Conf. Autonomous Agents*, Marina del Rey, CA, USA February 5–8, pp. 340–347. ACM, New York, NY, USA.

[16] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E. and Matsubara, H. (1997) RoboCup: a challenge problem for AI. *AI Mag.*, **18**, 73–85.

[17] Filippoupolitis, A., Hey, L., Loukas, G., Gelenbe, E. and Timotheou, S. (2008) Emergency Response Simulation using Wireless Sensor Networks. *Proc. 1st Int. Conf. Ambient Media and Systems*, Quebec City, Canada, February 11–14, pp. 1–8. ICST, Brussels, Belgium.

[18] Filippoupolitis, A., Gelenbe, E., Gianni, D., Hey, L., Loukas, G. and Timotheou, S. (2008) Distributed Agent-Based Building Evacuation Simulator. *Proc. Summer Computer Simulation Conf.* (*SCSC'08*), Edinburgh, UK, June 16–19, pp. 1–8. SCS, San Diego, CA.

[19] Gianni, D., Loukas, G. and Gelenbe, E. (2008) A Simulation Framework for the Investigation of Adaptive Behaviours in Largely Populated Building Evacuation Scenarios. *AAMAS Workshop on Organised Adaptation in Multi-Agent Systems*, Estoril, Portugal, May 12, pp. 1–15. Springer, Berlin/Heidelberg.

[20] Filippoupolitis, A., Loukas, G., Timotheou, S., Dimakis, N. and Gelenbe, E. (2009) Emergency Response Systems for Disaster Management in Buildings. *Proc. NATO Symp. C3I for Crisis, Emergency and Consequence Management*, Bucharest, Romania, May 11–12, pp. 1–14. NATO Research and Technology Organisation.

[21] Loukas, G., Timotheou, S. and Gelenbe, E. (2008) Robotic Wireless Network Connection of Civilians for Emergency Response Operations. *Proc. 23rd Int. Symp. Computer and Information Sciences* (*ISCIS 2008*), Istanbul, Turkey, October 27–29, pp. 1 – 6. IEEE, New York, NY, USA.

[22] Loukas, G. and Timotheou, S. (2008) Connecting Trapped Civilians to an Ad Hoc Network of Emergency Response Robots. *Proc. 11th IEEE Int. Conf. Communications Systems* (*ICCS 2008*), Guangzhou, China, November 19-21, pp. 599–603. IEEE, New York, NY, USA.

[23] Timotheou, S. and Loukas, G. (2009) Autonomous Networked Robots for the Establishment of Wireless Communication in uncertain emergency response scenarios. *Proc. 24th ACM Symp. Applied Computing, Track on Intelligent Robotic Systems*, Honolulu, Hawaii, USA, March 8–12, pp. 1171–1175. ACM, New York, NY, USA.

[24] Filippoupolitis, A. and Gelenbe, E. (2009) A Distributed Decision Support System for Building Evacuation. *Proc. 2nd IEEE Int. Conf. Human System Interaction*, Catania, Italy, May 21–23, pp. 323–330. IEEE, New York, NY, USA.

[25] Chandy, K.M. and Misra, J. (1979) Distributed simulation: a case study in design and verification of distributed programs. *IEEE Trans. Softw. Eng.*, **5**, 440–452.

[26] Bryant, R.E. (1984) A switch-level model and simulator for MOS digital systems. *IEEE Trans. Comput.*, **33**, 160–177.

[27] Reynolds, J. and Paul, F. (1988) A Spectrum of Options for Parallel Simulation. *WSC'88: Proceedings of the 20th Conf. Winter Simulation*, San Diego, CA, USA, December 12–14, pp. 325–332. ACM, New York, NY, USA.

[28] Peacock, J.K., Manning, E.G. and Wong, J.W. (1980) Synchronization of distributed simulation using broadcast algorithms. *Comput. Netw.*, **4**, 3–10.

[29] Paul, F. and Reynolds, J. (1982) A shared resource algorithm for distributed simulation. *SIGARCH Computer Archit. News*, **10**, 259–266.

[30] Nicol, D. and Reynolds, J. (1985) Problem oriented protocol design. *SIGSIM Simul. Dig.*, **16**, 27–30.

[31] Kumar, D. (1986) Simulating Feedforward Systems Using a Network of Processors. *ANSS'86: Proc. 19th Annual Symp. Simulation*, Tampa, FL, USA, March 12–14, pp. 127–144. IEEE Computer Society Press, Los Alamitos, CA, USA.

[32] Chandy, K.M. and Misra, J. (1987) Conditional Knowledge as a Basis for Distributed Simulation. Technical Report 00000029, California Institute of Technology, Pasadena, CA, USA.

[33] Lubachevsky, B.D. (1988) Bounded Lag Distributed Discrete Event Simulation. *Multi-Conf. Distributed Simulation*, January, pp. 183–191. SCS, San Diego, CA.

[34] Jefferson, D.R. (1985) Virtual time. *ACM Trans. Program. Lang. Syst.*, **7**, 404–425.

[35] Jefferson, D.D. and Sowizral, H. (1985) Fast Concurrent Simulation Using the Time Warp Mechanism. *Distributed Simulation*, San Diego, CA, USA, January 24–26, pp. 63–69. SCS, San Diego, CA.

[36] Gafni, A. (1988) Rollback Mechanisms for Optimistic Distributed Simulation Systems. *SCS Multiconf. Distributed Simulation*, San Diego, CA, USA, February 3–5, pp. 61–67. SCS, San Diego, CA.

[37] Fujimoto, R.M. (1990) Parallel discrete event simulation. *Commun. ACM*, **33**, 30–53.

[38] Damani, O.P., Wang, Y., and Garg, V.K. (1997) Optimistic Distributed Simulation Based on Transitive Dependency Tracking. *PADS'97: Proc. Eleventh Workshop on Parallel and Distributed Simulation*, Lockenhaus, Austria, June 10–13, pp. 90–97. IEEE Computer Society, Washington, DC, USA.

[39] Wang, J. and Tropper, C. (2007) Optimizing Time Warp Simulation with Reinforcement Learning Techniques. *WSC'07: Proc. 39th Conf. Winter Simulation*, Washington, DC, December 9–12, pp. 577–584. IEEE Press, Piscataway, NJ, USA.

[40] Rönngren, R. and Ayani, R. (1994) Adaptive Checkpointing in Time Warp. *PADS'94: Proc. Eighth Workshop on Parallel and Distributed Simulation*, Edinburgh, Scotland, UK, July 6–8, pp. 110–117. ACM, New York, NY, USA.

[41] Lin, Y., Preiss, B.R., Loucks, W.M., and Lazowska, E.D. (1993) Selecting the checkpoint interval in time warp simulation. *SIGSIM Simul. Dig.*, **23**, 3–10.

[42] Chetlur, M., Abu-Gazaleh, N., Radhakrishnan, R. and Wilsey, P.A. (1998) Optimizing Communication in Time-Warp Simulators. *PADS'98: Twelfth Workshop on Parallel and Distributed Simulation*, Banff, Alberta, Canada, May 26–29, pp. 64–71. IEEE Computer Society, Washington, DC, USA.

[43] Moreira, E.M., Helena, R., Santana, C. and Santana, M.J. (2005) Using Consistent Global Checkpoints to Synchronize Processes in Distributed Simulation. *IEEE International Symposium on Distributed Simulation and Real-Time Applications*, Montreal, Canada, October 10–12, pp. 43–50. IEEE Computer Society, Los Alamitos, CA, USA.

[44] Akyildiz, I.F., Chen, L., Das, S.R., Fujimoto, R.M. and Serfozo, R.F. (1992) Performance analysis of "Time Warp" with limited memory. *SIGMETRICS Perform. Evaluation Rev.*, **20**, 213–224.

[45] IEEE 1278 (1993) *Protocols for distributed interactive simulation applications*. International Telecommunications Union. Piscataway, NJ.

[46] Batalin, M., Sukhatme, G.S. and Hattig, M. (2004) Mobile Robot Navigation Using a Sensor Network. *IEEE Int. Conf. Robotics and Automation*, New Orleans, LA, April 26 – May 1, pp. 636–642. IEEE Press, Piscataway, NJ, USA.

[47] Corke, P., Peterson, R. and Rus, D. (2003) Networked Robots: Flying Robot Navigation Using a Sensor Net. *11th Int. Symp. Robotics Research* (*ISRR 2003*), Siena, Italy, October 19–22, pp. 234–243. Springer, Berlin/Heidelberg.

[48] Li, Q., Rosa, M.D. and Rus, D. (2003) Distributed Algorithms for Guiding Navigation Across a Sensor Network. *MobiCom'03: Proc. 9th Annual International Conf. Mobile Computing and Networking*, San Diego, CA, USA, September 14–19, pp. 313–325. ACM, New York, NY, USA.

[49] Tseng, Y., Pan, M. and Tsai, Y. (2006) Wireless sensor networks for emergency navigation. *Computer*, **39**, 55–62.

[50] JADE. *Java agent development environment*. http://jade.tilab.com (accessed January 17, 2010).

[51] FIPA. *The foundation for intelligent physical agents*. http://www.fipa.org (accessed January 17, 2010).

[52] Soldatos, J., Dimakis, N., Stamatis, K. and Polymenakos, L. (2007) A breadboard architecture for pervasive context-aware services in smart spaces: middleware components and prototype applications. *Pers. Ubiquitous Comput. J.*, **11**, 193–212.

[53] Badel, M., Gelenbe, E., Leroudier, J. and Potier, D. (1975) Adaptive optimization of a time-sharing system's performance. *Proc. IEEE*, **63**, 958–965.

[54] Gelenbe, E., Schmajuk, N.A., Staddon, J. and Reif, J.H. (1997) Autonomous search by robots and animals: a survey. *Robot. Auton. Syst.*, **22**, 23–34.

[55] Gelenbe, E. and Cao, Y. (1998) Autonomous search for mines. *Eur. J. Oper. Res.*, **108**, 319–333.

[56] Gelenbe, E., Seref, E. and Xu, Z. (2001) Simulation with Learning Agents. *Proc. IEEE*, **89**, 148–157.

[57] Gelenbe, E., Liu, P. and Lainé, J. (2006) Genetic Algorithms for Route Discovery. *IEEE Trans. Syst. Man Cybern. B*, **36**, 1247–1254.

[58] Gelenbe, E., Hussain, K. and Kaptan, V. (2005) Simulating autonomous agents in augmented reality. *J. Syst. Softw.*, **74**, 255–268.

[59] Gershon, R.R.M. (2006) The World Trade Center Evacuation Study: Lessons for Other High Rise Office Buildings. *NFPA World Safety Conf. and Exposition*, Orlando, FL, June 4–8.

[60] Gelenbe, E. (2003) Sensible decisions based on QoS. *Comput. Manage. Sci.*, **1**, 1–14.

[61] Bertsekas, D. and Gallager, R. (1987) *Data Networks*. Prentice-Hall, Upper Saddle River, NJ, USA.

[62] Humblet, P.A. (1991) Another adaptive distributed shortest path algorithm. *IEEE Trans. Commun.*, **39**, 995–1003.

[63] Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. (2001) *Introduction to Algorithms, (2nd edb)*. MIT Press, Cambridge, MA, USA.

[64] Gelenbe, E., Lent, R. and Xu, Z. (2001) Measurement and performance of a cognitive packet network. *J. Comput. Netw.*, **37**, 691–701.

[65] Elms, D.G., Buchanan, A.H. and Dusing, J.W. (1984) Modeling fire spread in buildings. *Fire Technol.*, **20**, 11–19.

[66] Hasofer, A.M. and Odigie, D.O. (2001) Stochastic modelling for occupant safety in a building fire. *Fire Safety J.*, **36**, 269–289.

[67] Gelenbe, E. (1979) Diffusion approximations: waiting times and batch arrivals. *Acta Inform.*, **12**, 285–303.

[68] Gelenbe, E. (2007) A diffusion model for packet travel time in a random multi-hop medium. *ACM Trans. Sensor Netw.*, **3**, 1–19.

[69] Gelenbe, E., Lent, R. and Nunez, A. (2004) Self-aware networks and QoS. *Proc. IEEE*, **92**, 1478–1489.

[70] Dobson, S., Denazis, S., Fernández, A., Gati, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N. and Zambonelli, F.

(2006) A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, **1**, 223–259.

[71] Ghanea-Hercock, R., Gelenbe, E., Jennings, N.R., Smith, O., Allsopp, D.N., Healing, A., Duman, H., Sparks, S., Karunatillake, N.C. and Vytelingum, P. (2007) Hyperion — next-generation battlespace information services. *Comput. J.*, **50**, 632–645.

[72] Gelenbe, E. (2009) Steps toward self-aware networks. *Commun. ACM*, **52**, 66–75.

[73] Gelenbe, E. and Timotheou, S. (2008) Random neural networks with synchronised interactions. *Neural Comput.*, **20**, 2308–2324.

[74] Gelenbe, E. and Timotheou, S. (2008) Synchronized interactions in spiked neuronal networks. *Comput. J.*, **51**, 723–730.