J Ambient Intell Human Comput (2017) 8:205–212
DOI 10.1007/s12652-016-0420-1

ORIGINAL RESEARCH

Agent architecture for crowd simulation in indoor environments

Rafael Pax[1] · **Juan Pavón**[1]

Received: 28 January 2016 / Accepted: 27 September 2016 / Published online: 8 October 2016

Abstract Simulation of crowds demands coping with scalability and performance issues that are not usually well supported by general purpose agent based simulation toolkits. On the other side, the use of agent models provides a great degree of flexibility in the specification of the behaviour of the entities and their interactions. The agent architecture that is presented in this work addresses both types of requirements, by taking advantage of the characteristics of its specific problem domain: the simulation of crowds in indoor environments. Several algorithms are implemented to improve the efficiency of the management of a high number of agents in order to cope with the performance in the processing of their movements and their representation. At the same time, different models are supported to specify decision making of the agents in order to allow rich behaviours. Agents can represent different types of entities such as people, sensors and actuators. This is illustrated with a realistic case study of the evacuation of the building of our Faculty of Computer Science, where different types of human behaviours are modelled in this kind of situations.

Keywords Agent based modelling · Crowd simulation · Ambient Intelligence · Indoor scenarios · MASSIS

✉ Juan Pavón
jpavon@fdi.ucm.es

Rafael Pax
rpax@fdi.ucm.es

[1] Facultad de Informática, Universidad Complutense Madrid, 28040 Madrid, Spain

1 Introduction

Agent-based models for crowd simulation usually focus on scalability issues derived from the management of a large number of agents, specially when considering their visualization or the way the agents find their way while avoiding obstacles and other agents (Schuerman et al. 2010). Different techniques have been proposed to cope with these issues, by relying on specific simplifications of the model, such as considering homogeneous agent behaviours, characterized with a fixed number of parameters (for instance, Legion 2016; Owen et al. 1996; Thunderhead Engineering 2016; TraffGo Ht 2013; Serrano and Botía 2013). Other works, like Bicharra et al. (2013), Bosse et al. (2013), Massive Software (2016), Saifi et al. (2013) and Wu and Sun (2014), have addressed the specification of richer agent behaviours, but the methodological aspects for a design process when developing them are not sufficiently exposed.

In reality, although most of the people react in a similar way under certain situations, there are different types of behaviours. Also, some behaviours can be adapted by learning (e.g., with fire drills) or during the action by some driving actors (e.g., security officers). Modelling these scenarios can be done with agents, but this usually implies a loss in performance because of the computation that such complex behaviours demand. In the case of indoor scenarios the number of agents may be large (thousands) but not very large as in a city (millions). The scale and structure of the indoor environments open the possibility to consider some algorithmic solutions that can cope with performance constraints, while keeping the ability to modelling of individual agents with heterogeneous behaviours.

Taking these assumptions into account, this work proposes an agent architecture for indoor scenarios where both

performance and flexibility in the behaviour of the entities are sought. Agents are specified and managed individually, but the effects of the crowd are taken into account by several methods that take advantage of characteristics of the indoor domain in order to cope with the efficiency and scalability issues in the processing of their movements and their visualization. At the same time, some alternative reasoning mechanisms are provided for each agent in order to allow modelling of rich and heterogeneous behaviours.

This agent architecture for simulation of crowds in indoor scenarios has been implemented as the MASSIS (Multi-agent System Simulation of Indoor Scenarios) framework (Pax and Pavón 2015b), whose general structure and components are presented in Sect. 2. The agent architecture is described in Sect. 3. Special attention is given to the definition of the behaviour of humans under different situations, which includes the process for decision making of the agents. Other relevant aspects to model are interactions among agents and with their environment, the events on the environment, and the precise representation of the environment (i.e., the building and its elements).

The algorithms that support an efficient management of basic elements that are needed for the simulation of a large number of agents in a building are presented in Sect. 4, which extends the description in Pax and Pavón (2015a).

There is a tutorial for the use of this framework, which is available at www.massisframework.com. In this paper its application is illustrated with the simulation of the evacuation of a building, which is a typical case for crowd modelling. A real building has been modelled, in concrete, our Faculty of Computer Science. This is presented in Sect. 5, before the conclusions in Sect. 6.

2 The MASSIS framework

MASSIS is a Java based framework, made of several components, many of them well-proven open source software. These are depicted in Fig. 1.

The main component, the simulation engine, relies on the MASON library (Luke et al. 2005), which has been extended with a set of packages that facilitate the specification of agents to simulate scenarios of smart indoor environments. Some of these are described in Sect. 4. MASON has been chosen because it provides a good support for agent-based simulation, with well proven efficiency. Also, its clear separation of the simulation core and the GUI, allows MASSIS to use the MASON simulation engine, while implementing its own display system.

The environment is defined using Sweet Home 3D (http://www.sweethome3d.com), a well known tool that is used to model all components involved in a building, such as walls, doors, stairs, people, etc. This software has been extended with plugins that support more flexibility in the characterization of the physical elements, which in some cases, when they may show some dynamic behaviour, will be represented by agents. For instance, in the case of sensors and actuators, they will be reactive agents, with simple behaviour and attributes. In the case of people, some physical characteristics can be defined such as

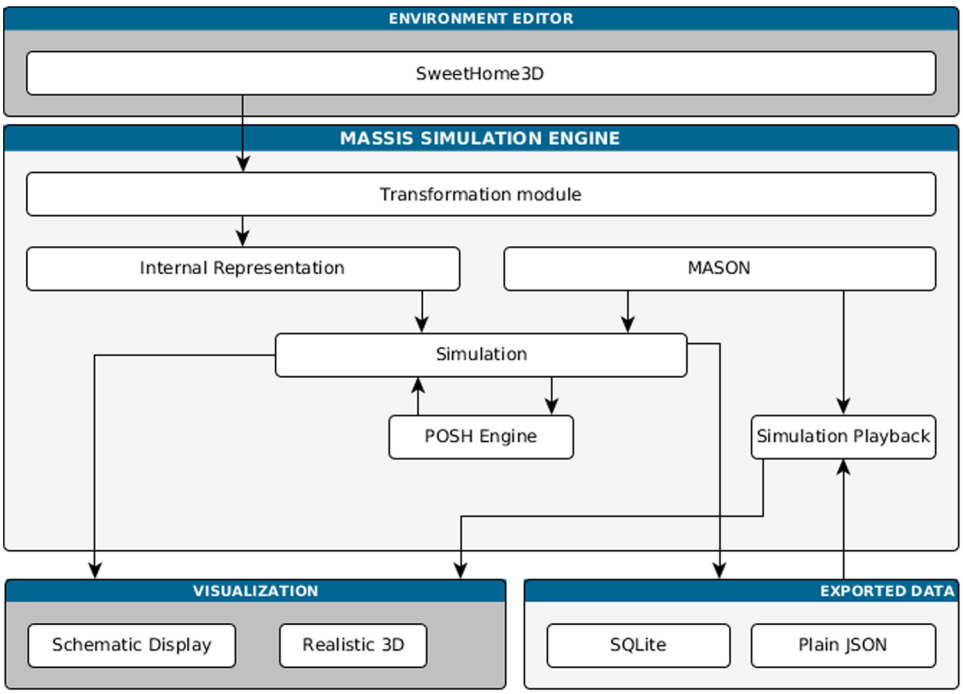

Fig. 1 Massis framework overview

weight, speed, but also some inherent attributes of the person (fear, courage, etc.) and a link to their behaviour. There are different ways to specify agent's behaviour, as it is explained in the next section.

Some tests have been performed in order to check the performance of their integration in MASSIS. Up to the order of ten thousand agents the experimentation has shown that execution times grow linearly with respect to the number of agents, so the results are satisfactory.

The evolution of the simulation can be visualized in real time by 2D (Figs. 5, 6, 2) and 3D (Fig. 8) displays. Although a 3D display is more realistic and nicer for demonstration purposes, the 2D view is useful for analysis and debugging. Also, the 2D visualization API allows the creation of user-defined layers in order to filter the different elements involved in the simulation. For instance, Fig. 2 represents the state of the agents as colors. Other possibilities are 2D representations of crowd density, perception area, agent's IDs, paths, states, steering forces, etc. Other customized views can be easily integrated as well.

The simulation can be logged in JSON format, as a single zipped file or in a SQLite database for further analysis or to playback again. This is interesting to allow the users to review the simulation when analysing what has happened.

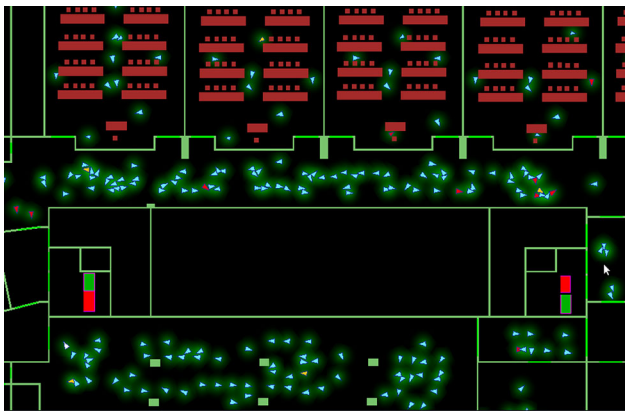

Fig. 2 Simulation 2D view, representing the different state of the agents by *color* (color figure online)

3 Agents in MASSIS

Agents support modelling of humans in the indoor environment. In this context, their behaviour is structured in two parts: those related with their perception and the interaction with the environment, and those dealing with the reasoning on the context and the decision making. They are implemented as low-level and high-level agent components, respectively. When the high level component decides *what to do next*, the action is executed by the low-level component, which performs all the necessary operations. The relationship between these components is illustrated in Fig. 3.

In order to create a new simulation model, the developer has to focus on the high level part of the agents, i.e. how agents take decisions based on their perception and past experience. In principle there is no need to deal with the low level part. This is provided by MASSIS packages, and it is there where MASSIS tries to get efficiency in the performance of the simulation. The developer has to concentrate on the definition of the high level part of the agents, and for that it is possible to use different techniques. In the first version of MASSIS the high level behaviour was controlled with the Pogamut's POSH engine (Gemrot et al. 2009). But others have been adopted recently, such as the cognitive agent model of ICARO (Gascueña et al. 2015).

3.1 High level behaviour

The high-level behaviour components deal with decision making, learning and communications with other agents. Decisions are taken on the knowledge of the environment, which is provided by the low-level components. As it has been said before, in the first implementation of MASSIS high level behaviour was controlled using a POSH engine and specified following the Behaviour Oriented Design (BOD) approach by Bryson (2001).

Developing MASSIS agents with the BOD approach requires the implementation of two parts:

1. A library of Behavior modules. They consist of a set of classes representing a set of modules for perception, action and learning. These are *primitives*, actions and

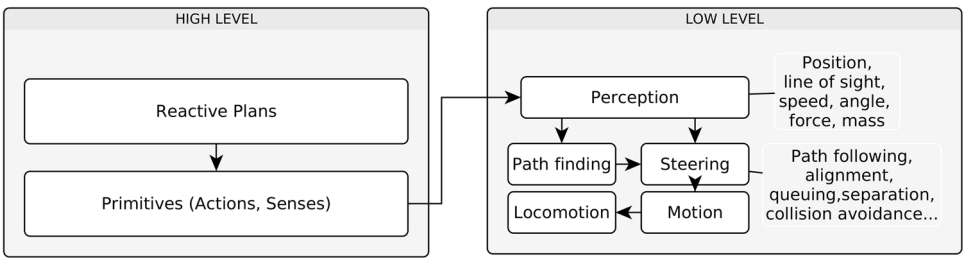

Fig. 3 MASSIS's agent model considers two main components: high level control and low level functions

senses, that can be called from the mechanism of action selection (see below). They also provide a place where certain states and knowledge can be stored in order to perform those actions, and they contain code that describes any sense that needs to be carried out to acquire that state and knowledge. In brief, they determine *how* to do something. These senses and actions are created in the native language for the problem space (in the case of MASSIS, Java).

2. POSH action selection scripts. A POSH (Parallel-rooted, Ordered Slip-stack Hierarchical) plan is a prioritized set of conditions and the related actions to be performed when the conditions have been met. It consists of *drive collections*, *competences*, and *action patterns*, as shown in the examples of Figs. 4 and 9.
   - Drive collections are the root of every POSH plan. On the action selection step, the drive collections select which goal the agent must try to accomplish. They can be seen as a set of conditional rules, that are evaluated from highest to lowest priority. Every time the condition of the drive collection with highest priority is satisfied (a higher rule interrupts a lower one), the POSH engine executes the corresponding action pattern or competence.
   - Competences are a set of nested if-then conditional trees, which can be reused several times inside the reactive plan. Competences are similar to drive collections, but rules they do not interrupt each other.
   - Action patterns are simple sequences of actions. Although they are not very flexible, they provide a layer of abstraction very useful when grouping actions.

   On the action selection step, the POSH engine executes the corresponding action pattern or competence from the drive collection with highest priority.

MASSIS encourages the use of variables and the agent's *Mental State* in POSH plans. *Mental States* are intended for representing the knowledge of the agent about its environment as a set of key-value pairs, but can be used for any other purpose, such as storing control variables in order manage the plan execution (see Fig. 4).

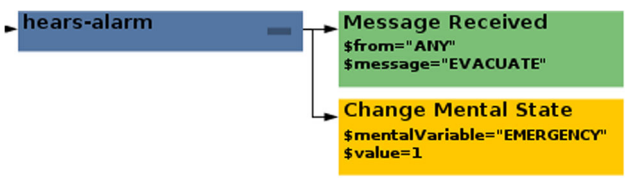

Fig. 4 *Mental state* modification in a POSH plan (*yellow*), triggered by the sense "Message Received" (*green*) on the Drive Element "hears-alarm" (*blue*) (color figure online)

3.2 Low level behaviour

The low-level components deal with the perception of the environment and a set of basic behaviours for interacting with it. These behaviours are mostly a combination of *steering behaviours* (Reynolds 1999), which are explained in the following section.

4 Crowd modelling and simulation issues

This section explains those aspects of the low level agents that are more relevant for the efficiency of the simulation. It is shown how some characteristics of the building model are used to improve the performance of the underlying algorithms.

The building model is designed with SweetHome3D and transformed into MASSIS internal representation to support the efficiency of algorithms implementation. This has an impact in the way agents interact with the environment, which is described below for different aspects: path finding, localization of elements and steering behaviours.

4.1 Path finding

Pathfinding is one of the issues that has more impact when simulating crowd behaviours. Some models treat the crowd as a single entity or a group in order to simplify the number of calculations, such as in Treuille et al. (2006). However, MASSIS, as it has been stated in the introduction, has as objective to support flexibility in agent behaviour, therefore the path finding model is executed individually by each agent, but taking advantages of some assumptions from the problem domain in order to gain in efficiency.

When the action selection component (see Fig. 3) decides that the agent must go to a particular location, a path must be computed from the agent's location to the target. Its computation is done in MASSIS by an A* search over the polygons' visibility graph. The computational cost has been considerably improved by taking advantage of several characteristics of the environment:

1. The perimeter of rooms consists of walls or doors.
2. The path from a point A to a point B, being B in a different room than A, must pass necessarily through a door.
3. In an indoor scenario walls intersect each other *very often*.

With these assumptions, several aspects have been improved to gain in computational efficiency:

- **Faster visible edges computation** In order to generate more realistic paths, the obstacle polygons are *inflated*, so the points of the path are slightly detached from the

polygon edges. The advantage of this separation in the case of pathfinding, is that the edges of the obstacle polygons are now inside the rooms boundaries, and the number of possible reachable nodes from the agent's location decreases (only the nodes inside the room's boundaries are considered).

- **Obstacle polygons reduction** The obstacles polygons that intersect can be merged, forming a bigger polygon. If most of the walls can be merged (as stated in Assumption 4.1), the number of obstacles is drastically reduced (for instance, in the case of the Faculty of Computer Science, the number of wall obstacles are reduced from 2351 to 171, less than 8 % of the original) and consequently also the number of intersection tests.
- **The complete path is not needed at once** The Assumption 4.1 implies that it is not necessary to compute always the whole path between two points A and B. It is frequent, due to events in the environment, that the agent decides to change its current targets before it has reached the end of the path. To avoid unnecessary calculations, at the beginning of the simulation, a navigation graph based on door-room connections is created, and the doors are converted into waypoints. When the agent requests a path, the A* algorithm runs only in the current room.

4.2 Elements localization

An intuitive way for storing the locations of the elements present during the simulation is using uniform grids. However, uniform grids are useful when the spatial data is distributed in an homogeneous way, which is rarely seen during simulation. The use of such grids can easily degenerate into situations where there are many redundant sparse cells. Furthermore, depending on the required accuracy, they can consume too many resources (see Fig. 5).

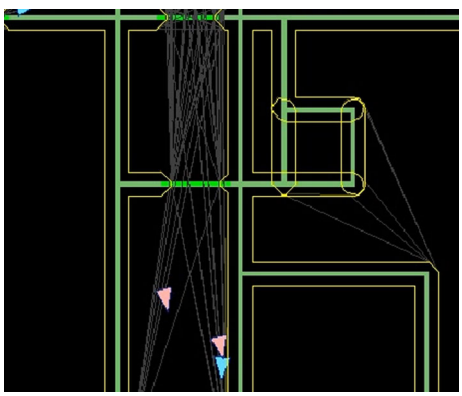

Fig. 5 Visibility graph and merged walls layer

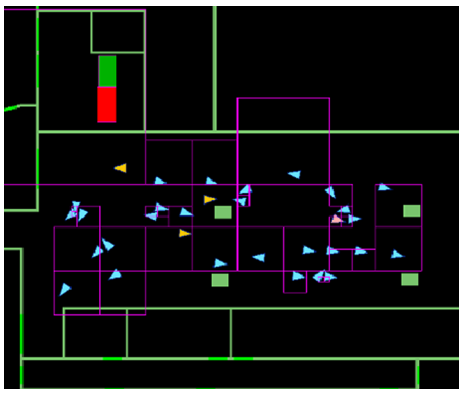

Fig. 6 Agents' Quad Tree layer, showing the space partitioning

People, sensors and actuators need real time information about the elements that surround them. This implies that, during simulation, lots of query ranges must be performed. In order to minimize the impact of this calculation, MASSIS uses a QuadTree (Finkel and Bentley 1974), a variable resolution data structure for retrieving agents' neighbours within a radius in an efficient manner (see Fig. 6). Although some CPU time is required in order to update the structure with the change of each agent, the gain obtained is worth it.

4.3 Steering behaviours

When the goals of the agent have been determined and the path to the target is known, the agent can start moving in the environment, avoiding collisions with obstacles (e.g., other agents, walls, etc.). These basic skills of the agent can be described with *steering behaviours* (Reynolds 1999), which need as parameters the agent's mass m, the agent's location $\mathbf{L}$, a maximum force f_{max} and a maximum speed s_{max}.

Every step n, the computed steering forces are applied to the agent's location (limited by f_{max}), producing an acceleration whose magnitude is inversely proportional to the vehicle's mass.

$$\mathbf{A}_n = \left(\frac{trunc(\mathbf{F}_n, f_{max})}{m}\right) \tag{1}$$

The velocity of the agent in every step n is approximated by the Euler integration. Adding the velocity at the previous step ($\mathbf{V}_{n-1}$) to the current acceleration ($\mathbf{A}_n$), produces a new velocity:

$$\mathbf{V}_n = trunc(\mathbf{V}_{n-1} + \mathbf{A}_n, s_{max}) \tag{2}$$

Finally, the velocity is added to the agent's location.

$$\mathbf{L}_n = (\mathbf{L}_{n-1} + \mathbf{V}_n) \tag{3}$$

MASSIS provides a flexible implementation of several behaviours of this type (e.g., seek, arrival, separation,

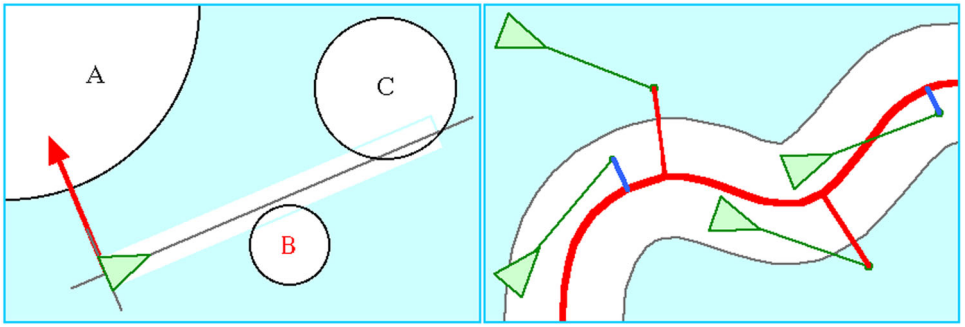

Fig. 7 Some of the steering behaviours implemented in MASSIS framework: seek and flee, obstacle avoidance and path following

collision avoidance, wall containment, and path following, see Fig. 7), that can be grouped into more complex behaviours, like flocking or queuing.

5 Simulation of the evacuation of a building

It is common practice in public buildings to define some emergency protocols, which may involve, for instance, evacuation of the building. Planning and testing these protocols might be costly, but making simulations about these situations can help to this task (at least, as a first approach). This case study addresses this kind of situation for the building of the Faculty of Computer Science at UCM, which is represented in Fig. 8.

Fig. 8 MASSIS 3D representation vs. a real photo

Three kinds of roles have been modelled, which are based on the behaviours described by Proulx (2001):

- **Students** have some knowledge about the building. Their priority is the evacuation, but if they see someone needing help, they will try to assist. They also pay attention to the evacuation signals and indications displayed in the Faculty's CCTV. Their behaviour can be modelled as a POSH plan (see Fig. 9).
- **Well-trained staff members**, are persons who work in the faculty (like a professor or administrative staff). They give instructions to the non-trained people, and try to assure that the evacuation is being done properly, following the established protocol.
- **Visitors** represent persons who have never been in the faculty, so the building is unknown to them. They interpret the fire alarm as something that is happening, so they will start searching for any person, expecting someone to tell them what to do, if something serious is really happening. TV messages can help them to understand that they must evacuate the building, and also other people (an *Student* or a *Well-trained staff member*).

Elements of the environment (sensors and actuators) can be also modelled as agents, with their respective plans, which are usually simpler than those of agents representing people. For instance, Fig. 10 shows a fire detector's plan: the

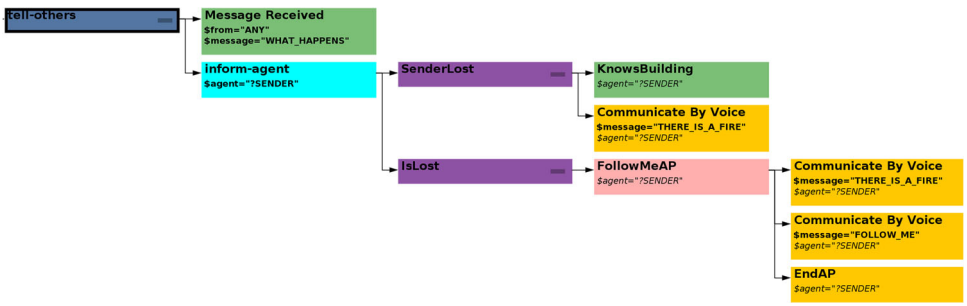

Fig. 9 Partial view of a student's POSH plan, having a drive element (*blue*),a competence (*cyan*), competence elements (*purple*), actions (*yellow*) and an action pattern (*pink*), which models the way in which the agent communicates with other people during the evacuation, reacting differently depending on the characteristics of the other agent. Note: some elements were omitted for clarity (color figure online)

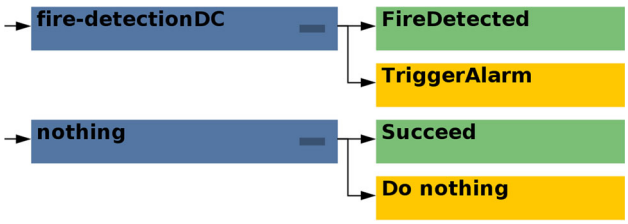

Fig. 10 Simplest reactive plan: a fire detector

existence of a fire triggers its only action, which is the activation of the fire alarm.

These behaviour definitions suggest that the agents in this context must be capable of using their visual perception of the environment, what they hear and the ability of interacting with other agents, in order to accomplish higher-level goals. These abilities are modeled using low-level behaviours, managed by POSH primitives, which are the leaves of any reactive plan tree.

For illustrating purposes, this is the protocol for a teacher (a well trained staff member) in an emergency situation:

> *When the alarm sounds the teacher of the group should go to the classroom door and order the students to close the windows if there is a fire. If instead of a fire there is a bomb threat, windows and doors should be left open. Students will leave the classroom through the door and they will be waiting for the teacher outside. The teacher will be the last person to leave the classroom. Once there is nobody in the classroom, the teacher will place a chair at the entrance of it,as an indication that the room has been evacuated entirely. Then the teacher will guide students toward the nearest exit.*

An agent that models the teacher has to consider the following basic skills:

- Hearing and vision capabilities.
- Ability to communicate with other agents by voice.
- Movement.
- Interaction with objects in the environment: Taking an object, carrying it, dropping it.

These skills are candidates to be *primitive* actions and senses. These primitives (which are implemented in the library of behaviour modules, see Sect. 3) are used by the reactive plan as *Triggers* of *Drive Elements*, components of *Action Patterns*, or they form part of one or more *Competences*.

Figure 11 shows the teacher's plan, which implements the protocol defined before.

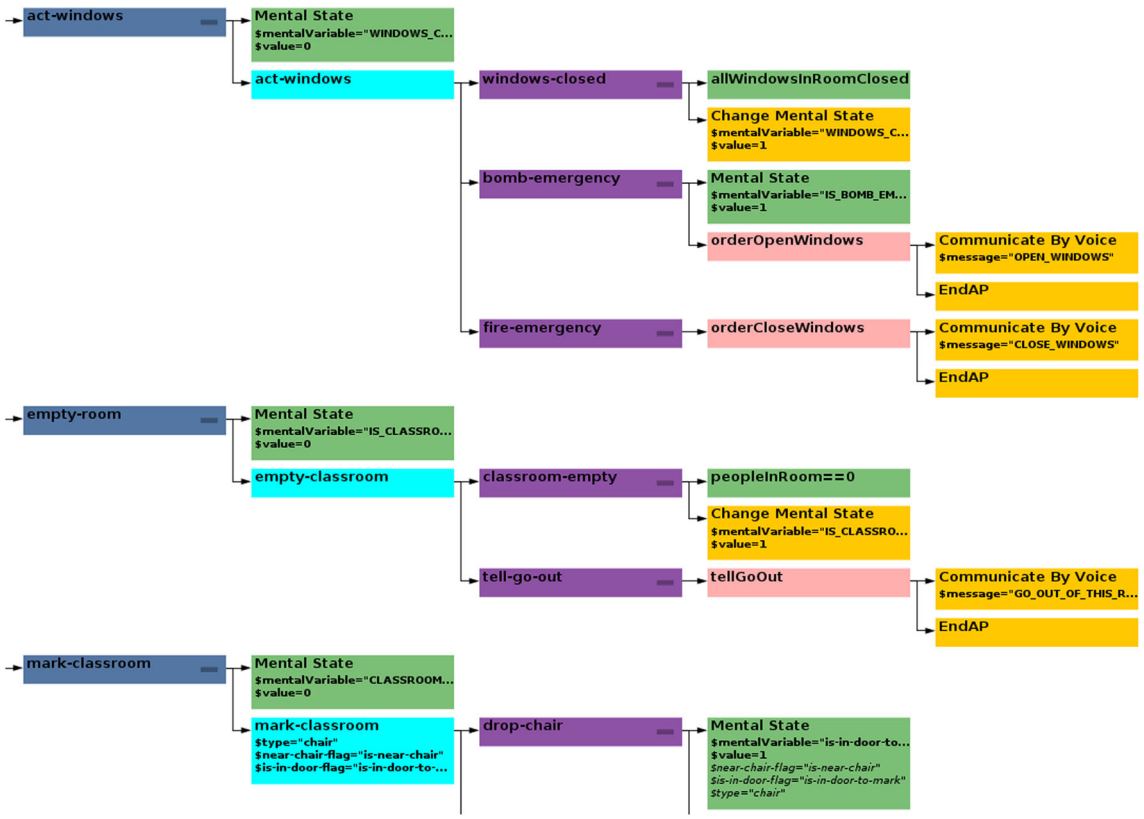

Fig. 11 Partial overview of the POSH plan for a Teacher agent

6 Conclusions

General purpose agent based simulation tools are well suited for quick prototyping in order to validate some social models with simple assumptions. However, dealing with a realistic environment model (e.g., a smart building) and rich agent behaviours, with different types of interactions, involves a deeper understanding of low level programming issues. This is even more relevant when dealing with large number of agents and interactions.

In our case we are concerned with crowd modelling in buildings. There are many applications in this domain such as the analysis of emergency protocols, recommendations for distribution of visitors in an exhibition or in a mall to avoid mass concentrations, testing infrastructure, etc. (see for instance Aversa et al. 2011; De la Prieta et al. 2015; Gómez-Romero et al. 2012). MASSIS builds on one of these agent based simulation tools, MASON, in order to solve performance issues, and at the same time allowing the specification of heterogeneous agent behaviours. The balance between performance and flexibility is achieved by taking advantage of the characteristics of the problem domain and by defining an agent architecture that hides the low level details. Section 4 shows several algorithms that implement basic agent functions with their environment (i.e., perception, movement) in a very efficient way. These are encapsulated in the low level behaviour of the agent. The developer does not need to deal with these and only has to model the high level behaviour, i.e., the decision making.

Decoupling low level from high level behaviour has also the advantage of allowing different agent models. This paper illustrates the use of the behaviour oriented design (BOD) approach, but others have been implemented, such as the goal driven agent model of ICARO and reactive agents (which can be seen in the tutorial that is available at http://www.massisframework.com).

Furthermore, the extensibility of the MASSIS platform is well supported through its component-based architecture. For instance, different visualizations can be managed during the simulation, new algorithms and agent attributes can be supported and monitored.

Acknowledgments This work has been supported by the Government of the Region of Madrid through the research programme MOSI-AGIL-CM (Grant P2013/ICE-3019, co-funded by EU Structural Funds FSE and FEDER), and by the Spanish Ministry for Economy and Competitiveness, with the project ColoSAAL (Grant TIN2014-57028-R).

References

Aversa R, Di Martino B, Ficco M, Venticinque S (2011) A simulation model for localization of pervasive objects using heterogeneous wireless networks. Simul Model Pract Theory 19(8):1758–1772

Bicharra AC, Sánchez-Pi N, Correia L, Molina JM (2013) Multi-agent simulations for emergency situations in an airport scenario. Adv Distrib Comput Artif Intell J 1(3):69–73

Bosse T, Hoogendoorn M, Klein M, Treur J, van der Wal N, van Wissen A (2013) Modelling collective decision making in groups and crowds: integrating social contagion and interacting emotions, beliefs and intentions. Auton Agents Multi-Agent Syst 27(1):52–84

Bryson J (2001) Intelligence by design. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology

De la Prieta F, Rodríguez S, Corchado JM, Bajo J (2015) Infrastructure to simulate intelligent agents in cloud environments. J Intel Fuzzy Syst 28(1):29–41

Finkel RA, Bentley JL (1974) Quad trees a data structure for retrieval on composite keys. Acta Inf 4(1):1–9

Gascueña JM, Navarro E, Fernndez-Sotos P, Fernndez-Caballero A, Pavn J (2015) IDK and ICARO to develop multi-agent systems in support of ambient intelligence. J Intell Fuzzy Syst 28(1):3–15

Gemrot J, Kadlec R, Bida M et al (2009) Pogamut 3 can assist developers in building AI (not only) for their videogame agents. Agents Games Simul LNCS 5920:1–15

Gómez-Romero J, Serrano MA, Patricio MA, Garca J, Molina JM (2012) Context-based scene recognition from visual data in smart homes: an information fusion approach. Person Ubiquitous Comput 16(7):835–857

Legion (2016) Legion pedestrian simulations. http://www.legion.com (Accessed 30 June 2016)

Luke S, Cioffi-Revilla C, Panait L, Sullivan K, Balan G (2005) Mason: a multiagent simulation environment. Simulation 81(7):517–527

Massive Software (2016) Massive: simulating life. http://www.massivesoftware.com/ (Accessed 30 June 2016)

Owen M, Galea ER, Lawrence PJ (1996) The exodus evacuation model applied to building evacuation scenarios. J Fire Prot Eng 8(2):65–84

Pax R, Pavón J (2015a) Agent-based simulation of crowds in indoor scenarios. In: Intelligent distributed computing IX, proceedings of the 9th international symposium on intelligent distributed computing—IDC'2015., Studies in Computational Intelligence, vol 616, pp 121–130

Pax R, Pavón J (2015b) Multi-agent system simulation of indoor scenarios. In: 2015 federated conference on computer science and information systems, FedCSIS 2015, pp 1757–1763

Proulx G (2001) Occupant behaviour and evacuation. In: Proceedings of ninth international fire protection symposium, pp 219–232

Reynolds CW (1999) Steering behaviours for autonomous characters. In: Proceedings of game developers conference 1999, San Jose, pp 763–782

Saifi L, Boubetra A, Nouioua F (2013) Approaches to modeling the emotional aspects of a crowd. In: 8th EUROSIM congress on modelling and simulation, pp 151–154

Schuerman M, Singh S, Kapadia M, Faloutsos P (2010) Situation agents: agent-based externalized steering logic. J Vis Comput Anim 21(3–4):267–276

Serrano E, Botía J (2013) Validating ambient intelligence based ubiquitous computing systems by means of artificial societies. Inf Sci 222:3–24

Thunderhead Engineering (2016) Agent based evacuation simulation. http://www.thunderheadeng.com/pathfinder/ (Accessed 30 June 2016)

TraffGo Ht (2013) Pedgo. http://www.traffgo-ht.com/ (Accessed 30 June 2016)

Treuille A, Cooper S, Popović Z (2006) Continuum crowds. In: ACM SIGGRAPH 2006 papers, pp 1160–1168

Wu S, Sun Q (2014) Computer simulation of leadership, consensus decision making and collective behaviour in humans. PLoS One 9:1