AgentSimJS: A Web-based Multi-agent simulator with 3D capabilities

Tobia Calenda, Massimiliano De Benedetti, Fabrizio Messina, Giuseppe Pappalardo, Corrado Santoro
University of Catania – Dept. of Mathematics and Computer Science
Viale Andrea Doria, 6 — 95125 - Catania, ITALY
{m.debenedetti,messina,pappalardo,santoro}@dmi.unict.it

***Abstract*—This paper describes the architecture of AgentSimJs, a web-based multi-agent simulator written in Javascript. The simulator is capable to render a 3D scene with objects and agents and allows the programmer to arrange simulations by specifying the behavior of agents. The architecture of AgentSimJs is highly modular, as several different components have been designed to enrich the simulator with the desired functionalities. AgentSimJs also has the capability of distributing the simulation among different machines and/or different thread, forming group of agents for broadcast communications.**

I. Introduction

Agent-based modeling and simulation (ABMS) is a wide used technique to model systems of autonomous, interacting agents [18], [13], [15], [7], [9], [8]. There are various applications which range from stock market simulations [10], [29] supply chains, consumer markets [27], [19], [22], as well as problems related to complex networks [4], [3], [21], [20], [21], as predicting the spread of epidemics, the threat of biowarfare and multi-robot simulations [25], [26], [6], [12], [24] The theoretical and practical foundations of ABMS have been widely studied [18], and a wide variety of toolkits and methods for developing agent models can be found in the literature [17].

In this work we present AgentSimJs, a JavaScript-based 3D simulation framework developed to build web based, 3D multi-agent simulations. Running this kind of simulation in a Web Browser has several advantages. The first benefit is that the simulator is multi-platform, as browsers natively support the JavaScript language, which is used nowadays to develop web application [11]. Moreover, AgentSimJs allows an easy collaboration between researchers, since running a simulation simply implies to start a web browser and connect to a specific link. Such a collaboration aspect is also stressed by AgentSimJs since, thanks to its modular architecture, algorithms and code developed for agents behaviors can be easily rescued.

The architecture of AgentSimJs is highly modular. Two different components handle agent behavior and agent interaction. The simulator also includes a component designed to manage group of agents, while the simulated physical environment is handled by a further dedicated component, which deals with the computation of the interactions among agents and between agents and scene objects. Code reuse and information sharing is properly managed in order to share all the algorithms and behaviors developed for the agents. Last but not least, the simulator offers the opportunity to distribute the simulation among several machines/thread through the component named Web-API integrator.

The paper is structured as follows. Section II discusses related work. Section III contains a detailed description of the architecture of the simulator. Section IV briefly describes a case study in order to highlight the capabilities of the simulator. Finally, Section V reports the conclusions.

II. Related work

One of the most used graphical environments for agent simulation is NetLogo [32], Java-based tool started in 1999 and still active. The aim of NetLogo is mainly modelling the evolution of complex systems where hundreds or thousands of agents can operate independently. It include also an authoring environment that allows researchers to publish their own models. In fact, it comes with the Models Library, which is a collection of pre-built simulation models to use and modify for biology, physics, computer science and so on. NetLogo gives also the opportunity to the user to run into a "classroom participatory-simulation" tool called HubNet, which allows students to control an agent in a simulation through the network.

Another tool, intended for SMP-aware, high performance, complex network simulations is ComplexSim [23], a C-based simulation platform developed to support the study of complex networks. Its architecture is based on two layers, the parallel simulation kernel and the complex network data and runtime. The simulator does not provide a graphical environment, and offers a simple a C-based programming APIs which, although simple, leads the programmer to write C code and implement his own data structures to program the behaviors entities.

Breve is a 3D simulation environment designed for simulation of decentralized systems and artificial life [16]. With respect to NetLogo, Breve is able to simulate continuous time and continuous 3D space, and, therefore,is suited to a different class of simulations. Breve includes an interpreted object-oriented language, an OpenGL display engine, collision detection, and the support for articulated body physical simulation and collision resolution with static and dynamic friction. Agent behavior can be written in Python [1] or by through a language named "steve", which is easy-to-use. Nevertheless, Breve is no longer maintained from 2009, but the simulation environment is still used, indeed Jon klein, brave author, has partially restored the website[1]. Furthermore, a JavaScript version of brave is available online [2] and it allows the researcher to build simple simulations.

[1] www.spiderland.org
[2] http://artificial.com/breve.js/

PALAIS [31] is a 3D virtual simulation environment for artificial intelligence with a special focus on games. It provides functionality for prototyping, testing, visualization and evaluation of game AI. It allows the definition and execution of arbitrary, three-dimensional game scenes and behavior. It provides also a scripting environment along with a programming interface, simulation control and data visualization tools. As stated by the authors, the scripting interface is minimal and can be accessed via simple JavaScript. PALAIS projects can be easily shared, in order to, e.g., collaborate with peers and build up showcases for algorithms and behaviors.

ARGoS [30] is an open source multirobot simulator capable of simulating at real-time large heterogeneous swarms of robots. The authors state that ARGoS is highly modular, therefore users can easily add custom features and allocate further computational resources where needed. Furthermore, multiple physics engines can be used and assigned to different parts of the environment. Results show that ARGoS is able to allocate and simulate about 10,000 simple wheeled robots 40% faster than real-time.

FLAME [2] is another agent-based modelling framework for high performance computing. The strength of the framework is that parallel programming expertise are not required to modelers, that can concentrate only to design the model. The authors used half a million of agents and 432 processors to show that a parallel efficiency of above 80% can be reached.

III. Architecture

AgentSimJs is a tool exploiting the library **threejs** [3] for the 3D engine. Furthermore, JavaScript Web Workers are used to implement a sort of parallelism in executing agent behaviors and the supporting message communication.

The architecture of AgentSimJs, which is depicted in Figure 1, is composed by the several components:

- **Agent**. It is core component of the framework, as it contains the primitives needed for the simulation of the behavior.

[3]http://threejs.org/

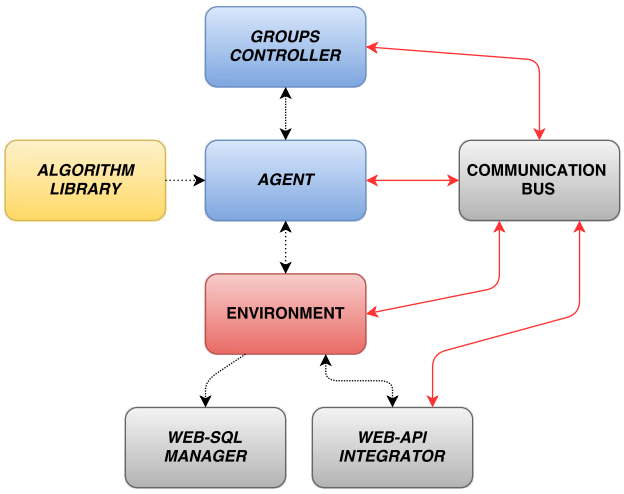

Fig. 1. Architecture of AgentSimJs

- **Communication Bus**. It allows the simulation of a communication system during the multi-agent simulation and handles agent interaction.
- **Group controller**. It enables the formation of groups of agents that will perform different tasks.
- **Environment**. It is used to simulate the graphical scenario and to handle physical interactions and (possible) collision among agents and the other objects present on the scenario.
- **Algorithms Library**. It is used by the programmer to design the algorithms that the agents will use to perform their task, as it provides an interface and some basic algorithms to be used by the simulation designer.
- **IndexedDB Manager**. This component is responsible to store the simulation data in a temporary cache in order to be export after the simulation.
- **Web-API integrator**. It enables the execution of several simulation on different machines by means of a shared environment (through a web-api or a socket interface).

A. Agent

JavaScript class `Agent` is the main component of the framework, as it represents the agent abstraction at framework level and it contains the implementation of some methods which implement several functionality, listed below:

- Movement: all the methods that can be used to set the position of the Agent, type of motion and path to follow. Indeed, each agent is able to perform a certain number of predefined motions (e.g. parabolic, linear, circular).
- Status: the methods used to get the agents status (e.g. agent position, agent role, agent group).
- Communication: it's an interface of the communication system of the framework (bus) that allows the agent to exchange messages with the others agents.
- Communication proxy: it's an embedded message proxy that can be used to build an overlay network among the agents.
- Environment: all the methods involved in the interaction between the agent and the environments.
- Geometry: these primitives enables the definition of the agent shape and the related reference system on the 3D scene.
- Proximity sensor: to make each agent aware of the environment a proximity sensor (laser scanner like) is simulated with a variable radius.

The status of any agent is represented by a set of variables that are managed by the status primitives, which allow to each agent to send its own status to the peers through the communication bus. Class `Agent` may be extended by the programmer through the addition of specific methods.

A (shared) data structure among agent instances is used to store the history of the agent position or to set the path of

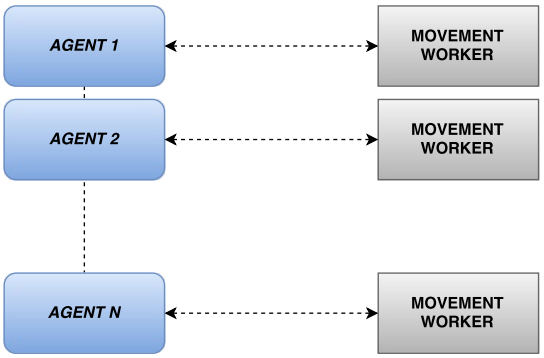

Fig. 2. Multiple movement workers

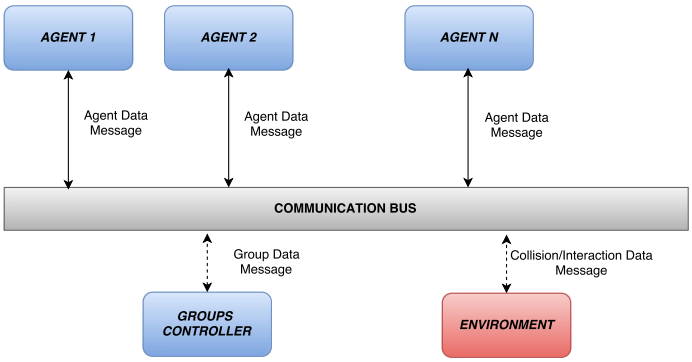

Fig. 4. Communication bus

the agents. Path planning can be performed by setting high level strategies in the group controller. A fine tuning may be performed by means of the Algorithm API and methods available for setting the movement of the single agent. In particular, the position methods of the agent instances – that compute/update the position of every agents – may be invoked run on a dedicated worker (movement worker) for each agent, as depicted in figure III-A.

For a high number of agents the movement worker can be centralized, i.e. a singleton instance may be used, in this case all the operations related to the computation of the agent positions will be managed by a single worker. The worker (Figure 3) has the responsibility to evaluate the position of each simulated agent at each simulation step, by computing the next position of the agent by means of the motion equation in a fixed time interval. In order to update the agent position, the worker may use the native web-worker communication interface or the bus communication of the framework.

In the centralized approach the movement worker will publish a message containing the agent position and ID for each agent; then every agent will receive the message and will update it's position if the Id of the message match with its own ID, otherwise the agent will discard the message.

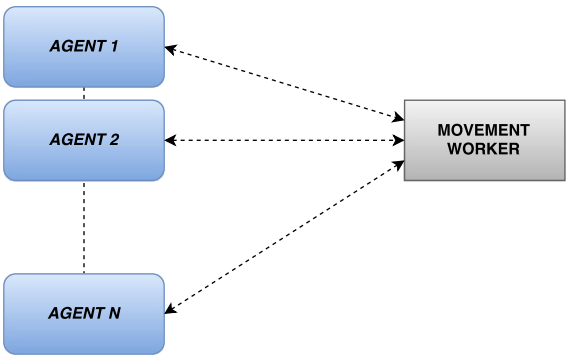

Fig. 3. Singleton worker

B. Communication Bus and Overlay network

The communication bus is an important component of the simulation framework. Through the communication bus each agent can exchange information with other agents in a centralized way; the communication bus can be also used by other components of the framework to exchange simulation data.

The type of messages exchanged in the communication bus can be listed as follow:

- Agent Data Message: this type of message are exchanged among the agents to share information about position, objects, environments, task etc.; each message will contain the Id of the sender and the Id of the receiver.
- Group Data Message: this type of message are sent from the group controller to each member of the group (a sub-set of agents); each message will contain the Id of the group that must receive the message.
- Agent-Environment Collision/Interaction: this type of message contains information about the interaction between the agents and the environment. The information are computed by the environment and sent to a specific agent.

As depicted in Figure 4, the communication bus can handle several kind of messages sent by actors of different levels, e.g. a message can travel from agent to agent or from a group controller to an agent. It enables also a remote integration between several simulations that runs in different machine. Figure 5 shows that through the web-api integrator several different actors, e.g. group controller and agents, can exchange simulation data while running in different machine. By the web-api integrator, agents will be able to send and receive message with the same approach.

Each agent is capable to join an overlay network with the other agents by communicating with within the maximum transmission range. In this case the communication bus is not needed, as only the agent communication proxy primitives of agents are used. In order to build an overlay network the message sent by the agent's will contain the Id of the sender and the number of agents that have to read the message. As in a real overlay network, if an agent is outside the range of another agents, it will not receive any message from it. Each agent that receive the message will perform the following operation:

- check if its own Id is included in the Id list of the agents that have already read the message. If not, the agent must add its own Id into the list and eventually

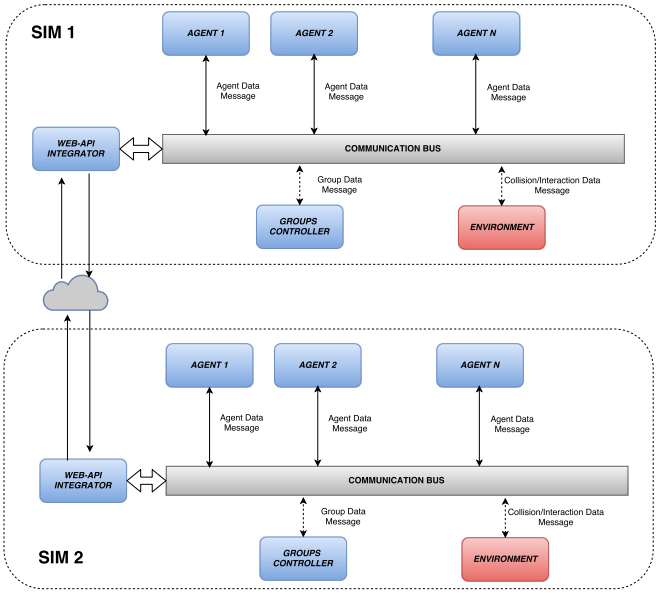

Fig. 5. AgentSimJs Communication bus

store the information and re-send the message to the near agents.

- If the Id is already included in the list, the agent will discard the message.

C. Environment

A component named *Environment* is responsible to create the 3D scene. Its geometry can be defined by a set of 3D primitives (that can be enhanced by the user). It is also capable to place the objects on the scene in a specific position chosen by the user and compute the potential collision among the agents and between the agents and the objects. The collision are evaluated by a batch process that compute the distance between every objects on the scene and then compare these distances with a fixed threshold. If the distance is lower than a given threshold a collision between the selected objects will occur.

Once a collision is detected the Environment will send a message to the involved agent, the message will set the agent status as off-line and will contain the Id of the scene object that has caused the collision. The simulation of the collision can be avoided by setting a specific parameter. The environment is the only component that is aware of the positions and status of every object on the scenes. The object on the scene can be placed also in a random way through a specific method during the scene initialization; the static objects can be added also at run-time. If a simulation is executed on several machines, each environment component will set a sharing of the information about all the objects in the scene during the initialization phase. This means that, at runtime, each environment will compute the collision and then will broadcast a message to the involved agents and the others environments through the communication bus. The environments that will receive the message will update the scene and the information about the involved agents and objects. In this way each environment that will detect the collision at the same time will send a message to the agents involved and this redundancy will make the simulation robust also when performed remotely. The slowest environment will receive the information and will align the scene to the others environment.

D. Group Controller

The group controller is used to create groups among the agents, to assign them several different tasks and manage the agents of the group. In particular, the group controller expose an interface with methods to select the agents that will form a specific group by assigning a GroupID to agents and group and manage the groups during the simulation.

Groups can be modified at runtime by reassigning the agents to different groups. Through the communication bus the group controller can send a broadcast message to the agents of a single group in order to share specific information with them, as well as a heartbeat periodic message to monitoring their status. In particular, an agent can be considered off-line/not available if the environment detects a collision between the agent and another item (i.e. an object or another agents). In this case the environment will send a message to the agent that will set its status as off-line, and the agent can send a message to the group controller to notify its new status. As alternative, it may ignore the heartbeat message by the group controller which, after a timeout, will consider the agent as off-line. The group controller is capable to manage only the agents that are running in the same machine, as it cannot send or receive message by the communication bus to the agents running in another machine. The main reason of this choice is represented by the network latency which will may become a bottleneck.

E. Algorithms Library

The algorithms API provides an interface that can be used to implements a specific algorithm that can be used during the simulation. The interface defines how the agents have to send its status and how they'll receive the result. This approach allows to separate the agent low level model and primitives from its behaviors.

The methods exposed by the Algorithms Library can be invoked by each agent, such that the agents behaviors will run in a distributed way through a shared knowledge (i.e. the algorithms), which will represent the common logic of the agents. The Algorithms Library will define also a library that can be enhanced during time by all the users that will use the AgentSimJs framework.

F. IndexedDB Manager

The design of the framework requires a DB which will store a cache of that can be hosted inside a web-browser. The IndexedDB manager, which relies on the IndexedDB API [28], is responsible to store all the simulation data on an embedded database that can be used as a data-cache for the client. The information stored in the database can be sent to any database on cloud (eg. through a web-api) or can be exported in CSV format to be further analysed by the user. The embedded data-cache capability ensures that the simulation can be launched without any remote connection requirement through a simple html page.

G. Web-API Integrator

In order to add the capability of performing a simulation in several machine through a web-browser a specific component has been developed. The Web-API integrator is responsible to build a remote communication bus that can exchange messages through remote machines. The approach builds a shared communication bus composed by each embedded bus (that is hosted in each remote machine), each bus is connected to the others through a web-api or a socket. This means that each message in each bus will be sent in a bus overly network. To perform a remote connection the web-api integrator can use a custom web-api or a specific socket (cenno sui socket e su come si possono gestire tramite JavaScript).

The web-api integrator will map each remote machine, will store simulation Id and status. The embedded data-cache can be connected to the web-api integrator to share data also in an asynchronous way or as a data-buffer. The remote machines are managed also through an "heartbeat" message containing the information about the status of the single remote machine and connection data (e.g. IP address, timestamp, etc.). If the heartbeat is not received by a given time threshold the machine will be set as off-line. The simulation will run without the agents of the disconnected remote machine (the Web-api integrator will send a message to the environment to eliminate from the simulation the external agents, the static object will persist on the scene).

IV. Case Study: Simulation of Sensor Releasing by UAVs

AgentSimJs has been recently employed to simulate a scenario on which a variable number of coordinated UAVs [14], [5] are employed to release sensors in a certain area of terrain, in order to accomplish the construction and management of a Wireless Sensor Network (WSN). The behavior of the UAVs has been set in order to perform a sensor releasing pattern which leads to a higher density towards the centroid of the zone. The simulator can be tested the following address: http://globus.dmi.unict.it/node.

The UI of the simulator provides a first interface useful to design a single circular area (hot zone, Figure 6), and an overall graphical view useful to manage all the created areas. In particular, for each area, it is possible to set some parameters that relate to the designed releasing model.

Furthermore, a simulation can be executed in two different modes: *graphical* and *batch*. In both cases, there is an option to stop the log stream which appears in the upper part of the simulation panel (see Figure 7). An interesting feature of the simulator is the generation of the set of coordinates where sensors have to be released. These data are used to perform the analysis needed to understand whether the deployment schema gives real advantages in terms of energy consumption and duration of the resulting wireless sensor network.

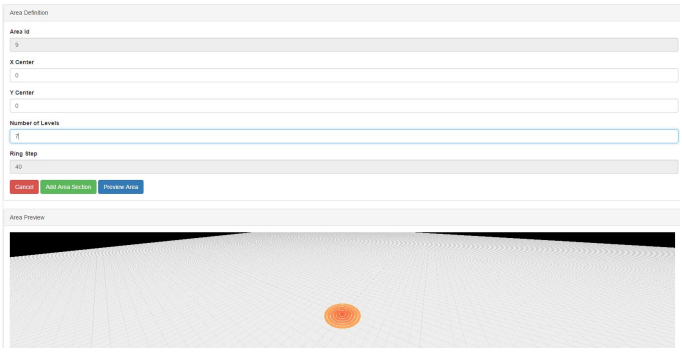

Fig. 6. Design the circular area

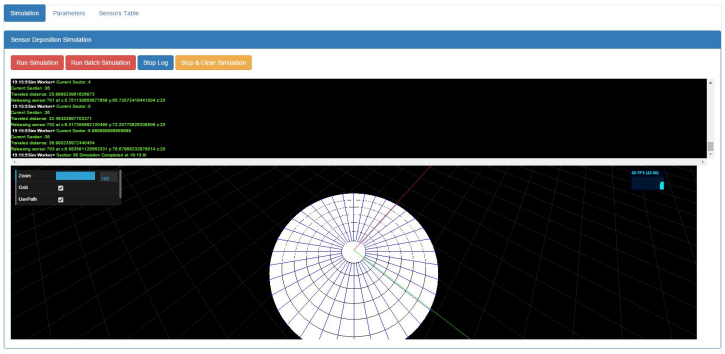

Fig. 7. A simulation running

V. Conclusions and Future Work

This paper has described the architecture of AgentSimJs, a web based based multi-agent simulation framework written in JavaScript. The simulator is capable to render a 3D scene with objects and agents and allows the programmer to arrange simulations by specifying the behavior of agents. The architecture of AgentSimJs is highly modular, as several different components have been designed to endow the simulator with the desired functionalities. With AgentSimJs the simulation can be distributed among different machines and/or different thread, agents can form groups for broadcast communications. Particular care was given to data storage, which has been addressed by a component relying on IndexedDB API.

We are working to realize a web portal to give the opportunity to use the framework to students and researchers, as well as releasing publicly the source code of the framework. The portal should provide the capability to store configurations

of simulations and code of agent behaviors developed by researchers. Furthermore, we planned to integrate the framework with a library for real time physics simulation. At the moment there is an attempt of porting of Bullet physics engine to JavaScript, named *ammo.js*. Finally, we have planned to integrate the framework with the JavaScript interface provided by ROS (Robotic Operating System) [4].

VI. Acknowledgements

This work is partially supported by projects PRISMA PON04a2 A/F and CLARA funded by the Italian Ministry of University.

References

[1] "The python language reference manual," 2016, https://docs.python.org/3/reference/index.html.

[2] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough, "Exploitation of high performance computing in the flame agent-based simulation framework," in *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*. IEEE, 2012, pp. 538–545.

[3] A. Comi, L. Fotia, F. Messina, G. Pappalardo, D. Rosaci, and G. M. Sarné, "An evolutionary approach for cloud learning agents in multi-cloud distributed contexts," in *2015 IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*. IEEE, 2015, pp. 99–104.

[4] ——, "Using semantic negotiation for ontology enrichment in e-learning multi-agent systems," in *Complex, Intelligent, and Software Intensive Systems (CISIS), 2015 Ninth International Conference on*. IEEE, 2015, pp. 474–479.

[5] M. De Benedetti, F. D'Urso, F. Messina, G. Pappalardo, and C. Santoro, "Uav-based aerial monitoring: A performance evaluation of a self-organising flocking algorithm," in *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. IEEE, 2015, pp. 248–255.

[6] M. De Benedetti, F. DUrso, F. Messina, G. Pappalardo, and C. Santoro, "Self-organising uavs for wide area fault-tolerant aerial monitoring," 2015.

[7] P. De Meo, F. Messina, D. Rosaci, and G. M. Sarné, "Recommending users in social networks by integrating local and global reputation," in *International Conference on Internet and Distributed Computing Systems*. Springer International Publishing, 2014, pp. 437–446.

[8] ——, "2d-socialnetworks: away to virally distribute popular information avoiding spam," in *Intelligent Distributed Computing VIII*. Springer International Publishing, 2015, pp. 369–375.

[9] ——, "An agent-oriented, trust-aware approach to improve the qos in dynamic grid federations," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5411–5435, 2015.

[10] ——, "An agent-oriented, trust-aware approach to improve the qos in dynamic grid federations," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5411–5435, 2015.

[11] D. Flanagan, *JavaScript: the definitive guide*. " O'Reilly Media, Inc.", 2006.

[12] G. Fortino, W. Russo, and C. Santoro, "Translating statecharts-based into bdi agents: The dsc/profeta case," in *German Conference on Multiagent System Technologies*. Springer, 2013, pp. 264–277.

[13] T. French, N. Bessis, F. Xhafa, and C. Maple, "Towards a corporate governance trust agent scoring model for collaborative virtual organisations," *International Journal of Grid and Utility Computing*, vol. 2, no. 2, pp. 98–108, 2011.

[14] S. Gupte, P. I. T. Mohandas, and J. M. Conrad, "A survey of quadrotor unmanned aerial vehicles," in *Southeastcon, 2012 Proceedings of IEEE*. IEEE, 2012, pp. 1–6.

[15] M. Higashino, T. Hayakawa, K. Takahashi, T. Kawamura, and K. Sugahara, "Management of streaming multimedia content using mobile agent technology on pure p2p-based distributed e-learning system," *International Journal of Grid and Utility Computing 26*, vol. 5, no. 3, pp. 198–204, 2014.

[16] J. Klein, "Breve: a 3d environment for the simulation of decentralized systems and artificial life," in *Proceedings of the eighth international conference on Artificial life*, 2003, pp. 329–334.

[17] F. Klügl and A. L. Bazzan, "Agent-based modeling and simulation," *AI Magazine*, vol. 33, no. 3, p. 29, 2012.

[18] C. M. Macal and M. J. North, "Agent-based modeling and simulation," in *Winter simulation conference*. Winter simulation conference, 2009, pp. 86–98.

[19] F. Messina, G. Pappalardo, D. Rosaci, C. Santoro, and G. M. Sarné, "A trust model for competitive cloud federations," *Complex, Intelligent, and Software Intensive Systems (CISIS)*, pp. 469–474, 2014.

[20] ——, "A distributed agent-based approach for supporting group formation in p2p e-learning," in *Congress of the Italian Association for Artificial Intelligence*. Springer International Publishing, 2013, pp. 312–323.

[21] ——, "Hyson: A distributed agent-based protocol for group formation in online social networks," in *German Conference on Multiagent System Technologies*. Springer Berlin Heidelberg, 2013, pp. 320–333.

[22] F. Messina, G. Pappalardo, D. Rosaci, and G. M. Sarné, "An agent based architecture for vm software tracking in cloud federations," in *Complex, Intelligent and Software Intensive Systems (CISIS), 2014 Eighth International Conference on*. IEEE, 2014, pp. 463–468.

[23] F. Messina, G. Pappalardo, and C. Santoro, "Complexsim: a flexible simulation platform for complex systems," *International Journal of Simulation and Process Modelling 6*, vol. 8, no. 4, pp. 202–211, 2013.

[24] ——, "Integrating cloud services in behaviour programming for autonomous robots," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer International Publishing, 2013, pp. 295–302.

[25] ——, "Designing autonomous robots using golem," in *XV Workshop "Dagli Oggetti agli Agenti"*, vol. 1260. CEUR-WS, 2014.

[26] ——, "A goal-centric framework for behaviour programming in autonomous robotic systems," in *Mechatronic and Embedded Systems and Applications (MESA), 2014 IEEE/ASME 10th International Conference on*. IEEE, 2014, pp. 1–6.

[27] F. Messina, G. Pappalardo, C. Santoro, D. Rosaci, and G. M. Sarné, "An agent based negotiation protocol for cloud service level agreements," in *2014 IEEE 23rd International WETICE Conference*. IEEE, 2014, pp. 161–166.

[28] M. D. Network, "Indexeddb api," https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API.

[29] M. R. Ogiela and U. Ogiela, "Linguistic approach to cryptographic data sharing," in *2008 Second International Conference on Future Generation Communication and Networking*, vol. 1. IEEE, 2008, pp. 377–380.

[30] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle *et al.*, "Argos: a modular, multi-engine simulator for heterogeneous swarm robotics," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 5027–5034.

[31] P. Schwab and H. Hlavacs, "Palais: A 3d simulation environment for artificial intelligence in games," in *Proceedings of the AISB Convention*, 2015.

[32] S. Tisue and U. Wilensky, "Netlogo: Design and implementation of a multi-agent modeling environment," in *Proceedings of agent*, vol. 2004, 2004, pp. 7–9.

[4] http://www.ros.org/