AMAK - A Framework for Developing Robust and Open Adaptive Multi-agent Systems

Alexandre Perles, Fabrice Crasnier, Jean-Pierre Georgé

AMAK - A Framework for Developing Robust and Open Adaptive Multi-Agent Systems

Alexandre Perles[1], Fabrice Crasnier[1], Jean-Pierre Georgé[1]

IRIT, University of Toulouse, Toulouse, France
{Alexandre.Perles,Fabrice.Crasnier,Jean-Pierre.George}@irit.fr

Abstract. Multi-agent systems are commonly used in various research fields such as artificial intelligence, operational research, simulation, biology, ... However, this diversity often requires that the system and agents in it are created from scratch for each new research project. In addition to the fact that this forces the developers to code similar elements anew each time, this can introduce non-negligible biases (e.g. an information accessible to every agent which shouldn't be or a scheduler executing twice due to a user interface design failure). To avoid this, we propose in this paper AMAK, a framework developed in Java™ to facilitate the design and development of a multi-agent system. First, we present the particularity of Adaptive Multi-Agent Systems. Secondly, a state of the art of the main tools and software aiming at facilitating the development of such systems is discussed. Then, we develop the architecture of the framework and the main features. The use of the framework is illustrated with an application for socio-technical ambient systems. And finally, we conclude with the perspectives of this work.

Keywords: Multi-Agent System; Cooperation; Framework; Development; Java

1 Introduction

Today, companies develop ad-hoc Artificial Intelligence solutions often based on machine-learning with methods such as Gradient Boosting, Random Forest, Deep Learning and Genetic Algorithms. These methods show really great performance when the problem is well-defined and data are numerous and centralized.

But in modern society, we tend to have dynamic problems that are not completely identified/specified and involving numerous interacting entities. To solve these types of complex problems, it is necessary to reconsider the way we handle problems, notably by decentralizing the control and by adapting the solving process during time. This motivates the Multi-Agent Systems (MAS) approaches and more specifically the ones aiming for self-adaptation.

An Adaptive Multi-Agent System is a MAS in which agents interact with their neighborhood in a cooperative way [1] so as to adapt their behavior to disturbances they perceive. The main idea is that when agents are all cooperative at the microlevel, the global system solves the problem at the macrolevel [6].

These kinds of approaches tend to be more and more used in private and public research. The development of these systems requires to follow a bottom-up approach. It means that the focus is put on the behavior of agents [5].

To avoid common errors in the development of Adaptive Multi-Agent Systems and given the fact that the only differences between this kind of systems is in the architecture of agents and their behavior, we propose a framework integrating the common bricks useful for all adaptive multi-agent system projects.

This paper presents the framework we developed and highlights its characteristics. In the first part, more details are given on the Adaptive Multi-Agent System theory. Secondly, a state of the art of the main tools and software aiming at facilitating the development of such systems is discussed. Then, the characteristics of the framework are presented. And finally, the use of the framework is illustrated with an application for socio-technical ambient systems.

2 Adaptation in Multi-Agent Systems

The need for adaptation comes from the fact that complex problems are generally not completely specified. Indeed, complex problems or systems involve numerous entities interacting dynamically and are therefore hard to solve, control or predict.

As agents are autonomous and have a limited perception (they can only perceive a small part of their environment), the adaptation in Multi-Agent Systems can come from three factors:

- The tuning of parameters: Agents can change the way they act by modifying their internal parameters based on their perception;
- The changes of interaction: An agent initially interacting with another can decide to break this link and interact with other agents;
- The addition/removal of an agent: If an agent is useless, it can decide to remove itself. Conversely, an agent can decide to create another agent if it thinks it will benefit the system.

The adequate functioning of a multi-agent system requires a cooperative behavior of agents. Cooperation is a social attitude in which agents coordinate and help each other. An agent is considered as cooperative if he tries to reduce the *criticality* in its neighborhood. The *criticality* of an agent can be defined as "the state of dissatisfaction of an agent regarding its local goal"[11]. This information can be seen as the main motivator for an agent to act, leading to a continuous self-organizing process between the agents, which is the engine for self-adaptation at the system level.

3 Existing tools

The development of adaptive multi-agent systems can be made with almost any programming language. However, a code basis can be useful as it allows to focus

on the specific features and to avoid common errors. The most used tools or frameworks are Jade (Java Agent DEvelopment Framework), GAMA (Gis & Agent-based Modelling Architecture), NetLogo, SARL and MadKit.

Jade is a Java™ framework first released in 2000. This framework allows to distribute agents in various containers. Basically, the Jade framework contains a class named Agent that must be extended by any developed agent. These agents are then controlled by two special agents with a global view of the system: The DF Agent which provides a directory with the full list of available agents and the AMS Agent which controls the platform. The main advantages of this framework are that it has been used in many projects, that it generalized the concept of behaviors and that it is compliant with the FIPA standards [2]. However, the development of systems using Jade may not be intuitive. Moreover, even if it seems adapted to the development of multi-agent systems, adaptive multi-agent systems require a more decentralized control and is notably incompatible with agents with a global view.

GAMA is a platform dedicated to agent-based modeling and simulation. The development is high-level and therefore allows to focus on the behaviors of agents. However, it requires to learn a specific programming language called GAML [15].

NetLogo is particularly useful for teaching. Packaged as a full-featured Integrated Development Environment, it provides a really simple way to experiment with multi-agent systems. It differentiates from other by its simplicity and has been used for many scientific articles. However, despite its simplicity, it also requires to learn a specific programming language [16].

SARL is a statically-typed agent-programming language. It is a specific language aiming at filling in the gaps of the Java™ language. It is based on events and has a syntax similar to a mix between Java™ and Python. However, it lacks easy rendering capacities and forces its user to learn a new language [14].

MadKit is a generic multi-agent system platform based on a organizational model. It has been developed in Java™ and has a clear structure. It is relatively easy to use however it is not really intuitive [10].

4 The AMAK framework

The framework AMAK is an all-in-one Java™ library aiming at facilitating the development of Adaptive Multi-Agent Systems. It is presented as a jar file that must be added to Java™ projects [12]. The programming language Java™ has been chosen as it is widely used in both research and education fields. In addition to this, this language is object-oriented and programs developed with it are portable [9].

During the design and development of Adaptive Multi-Agent Systems, the focus should be made on the agents' behaviors. However, to be able to code the behavior of an agent, it is necessary to code all the related concepts which are the scheduling of agents, the plotting of data and specific algorithms (for example, determining the most critical agent). Given the bottom-up approach used in the development of such system, one small change in any part of the code can

have non-negligible impact on the system overall functioning and can notably introduce biases and therefore lead to wrong results. The use of a framework allows to avoid common errors and also to save time.

By providing a set of inheritable classes and methods, the framework AMAK gives a solid basis to develop such systems. The three main abstract classes are AMAS, Agent and Environment. These concepts present in every multi-agent system have specific characteristics and comply with the Adelfe methodology designed to assist the development of adaptive multi-agent systems[4]. The Figure 1 represents the relations between the main classes. Inheriting the AMAS class allows to develop a multi-agent system compliant with the scheduler given in AMAK. It also supports the addition and removal of agents safely at runtime. The abstract class Agent mainly contains three overridable methods: onPerceive, onDecide and onAct. These methods are common to all kind of agents as defined in [7]. Also, it can be necessary to execute specific code on each agent for example when all agents have finished their cycle. The framework contains such methods. It can be used for example to debug or to log agent states. Finally, the class Environment is abstract and must be extended by the environment of the multi-agent system. The extended class must provide direct access to any information the agents may require. Each agent belonging to the multi-agent system has a pointer to this environment and use it to perceive its part of the environment.

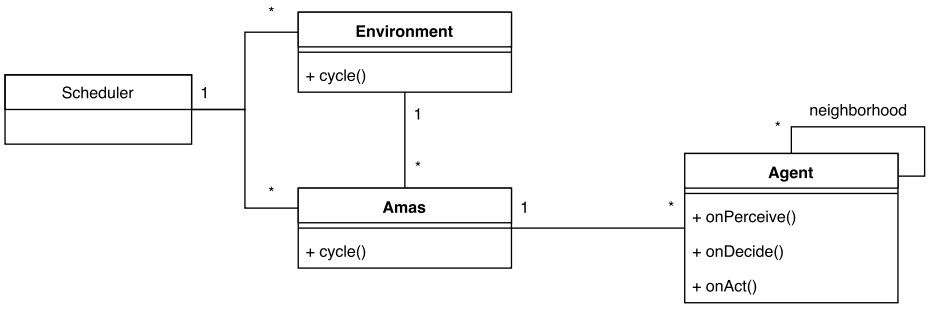

Fig. 1. Relations between the main classes

The scheduling is ensured by a specific class Scheduler which enables to execute the cycle of the AMAS and therefore the cycle of each agent synchronously (or not) with the environment. Adaptive Multi-Agent Systems have specific characteristics. Notably the fact that agents have a criticality and often use it to determine which agent should be helped. This criticality value is computed at various times directly by the framework to ensure that the value complies with the real state of the agent and that the computation is made as few often as possible. Moreover, specific algorithms are added. For example, it exists a predefined method which allows any agents to directly know which agent in its neighborhood is the most critical.

During the design and development of multi-agent systems, it is often necessary to visualize data and notably agents locations and interaction links. The module DrawableUI is an abstract class that must be extended by any rendering class you may have. This class contains a method called at each cycle (synchronized with the multi-agent system scheduler or not) that must be implemented.

The framework AMAK is also packaged with various tools generally used in such projects. The tool *AVT* (Average Value Tracker), developed by Sylvain Lemouzy [11] for the SMAC team, is a tool aiming at finding a potentially-dynamic value in a specific range given simple feedback (smaller, bigger and almost good). The problems faced by Adaptive Multi-Agent Systems are often dynamic and not completely specified. Such tool is particularly adapted.

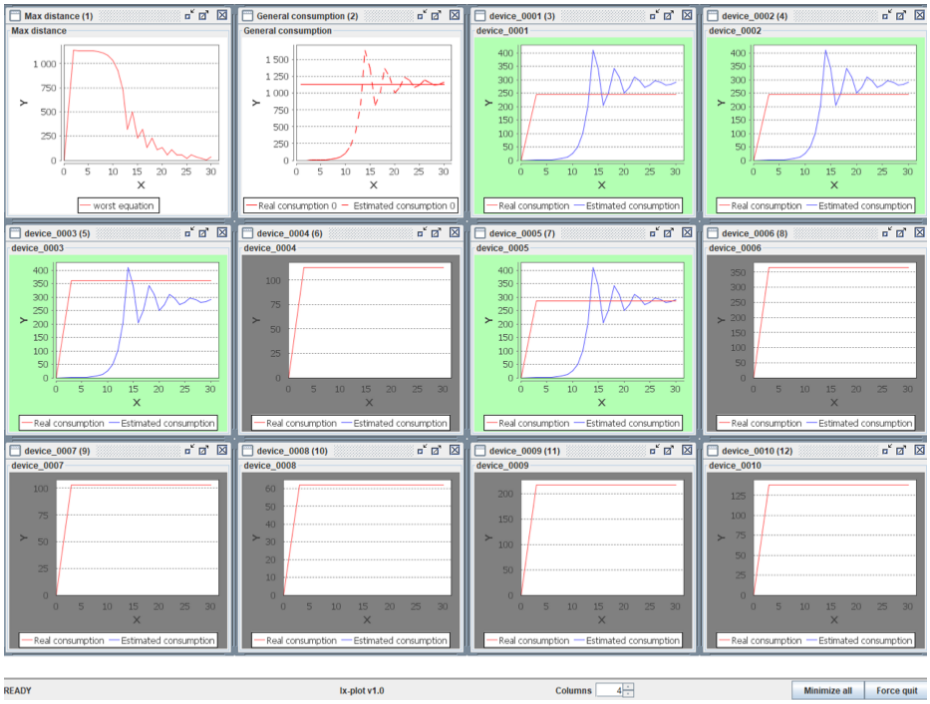

Fig. 2. Screenshot of the *LxPlot* tool used in AMAK

The second integrated tool is *LxPlot* (`https://bitbucket.org/perlesa/lx-plot`). It exists multiple libraries for plotting data in Java™. The most used is probably JFreeChart (`http://www.jfree.org/jfreechart/`). However, it requires to write a lot of code to manage to display simple points. The library *LxPlot* integrates the library JFreeChart and simplifies it to allow to draw points or lines using one simple line of code. The Figure 2 presents a screenshot of the tool *LxPlot* used in a socio-technical ambient system presented later.

Sometimes, simple piece of code can be executed a lot of times. If this code is not optimized, it can have non-negligible impact in term of computation time. To face this problem, the framework includes a simple tool called *Profiler* which allows to measure the exact time in nanoseconds taken by the execution of a piece of code.

Java™ natively includes convenient classes to read or write files. However, as for the plotting library, this requires to write a lot of code to read or write a simple file. The tool *FileHandler* allows to simply read or write text, CSV and JSON files.

Finally, a multi-agent system can handle an important amount of agents (for example, the drone application is able to handle thousands of agents on a computer with a medium configuration). It, therefore, can be hard to observe what is happening during the execution. AMAK integrates convenient tools to display data and a logging system with various log levels and a tag system to filter and display only some information based on a regular expression. Moreover, by default, the logging system writes to the standard output but it can easily be rerouted to write log to a file or even send logging data through the network.

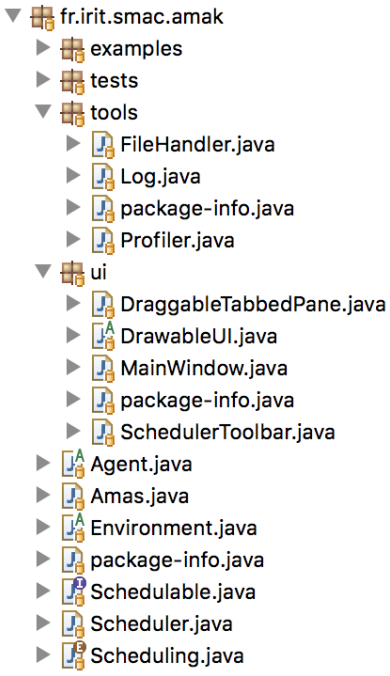

Fig. 3. Main classes of the proposed framework

The Figure 4 represents the execution time of the system over the number of agents in the system. For this evaluation, agents do random actions at each

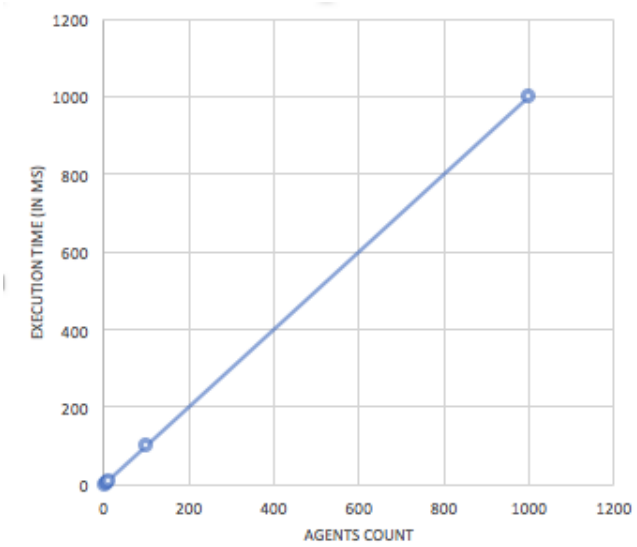

Fig. 4. Evolution of the execution time of a system developed with AMAK over the number of agents

cycle. It can be seen in this figure that the complexity in time evolves linearly with the number of agents in the system. The complexity of communication can't be observed as it mainly depends on the agent behavior. However, following the AMAS theory, agents are cooperative. Therefore, they are not supposed to send useless messages.

The framework AMAK provides an easy way to develop and maintain a multi-agent system. Such developed systems are open (agents can be added or removed at runtime), efficient, reliable and monitorable. The Figure 3 presents the main classes. The framework is provided for free under the LGPL (Lesser General Public License) license [8].

5 Application to Socio-Technical Ambient Systems

Since the beginning of its development in 2017, AMAK has been used in various projects. First of all, AMAK is used in education in last year of multiple master's degrees. Given its simplicity of use and the fact that it is based on the commonly used language Java™, it is particularly adapted to education. It has been used in education on two examples: the solving of the philosopher's dinner problem [12] and the adaptive management of a drones fleet [13]. It allows to easily understand concepts related to the design and development of multi-agent systems. Secondly, each year, new internships and PhD students integrate the IRIT laboratory. To avoid losing too much time understanding the process of conceiving multi-agent systems, AMAK gives a robust and clear structure to start with [3].

AMAK has been used for the development of a socio-technical ambient system aiming at making emerge an ambient welfare.

An ambient environment consists of a multitude of devices, some of which measure its physical characteristics such as heat sensors, light sensors, olfactory sensors and hearing sensors while others act directly on it to modify its state such as the electric motors of the shutters to control the opening and closing of the curtains, or the switches to control the switching "on" and "off" of the ceiling or fixture. AMAK has been used for the development of the ambient socio-technical system aimed at bringing out the notion of ambient well-being of immersed humans in this environment. By simulating connected devices such as sensors and effectors, the framework allows such a system to discover the optimal combination to maximize user comfort and bring out our goal of well-being. In this research framework, learning about the environment linked to well-being is carried out using multi-agent systems related to the four characteristics of human physical comfort i.e. thermal comfort, visual comfort, olfactory comfort or the acoustic comfort we will call "MAS Comforts". The latter ones are initially responsible for learning the thermal, luminous, olfactory or auditory environments and in a second phase for providing their criticality levels to the multi-agent systems representing the ambient socio-technical system consisting of sensors and effectors scattered in the environment we call "MAS Devices". To meet the socio-economic objectives and to realize a functionality of eco-citizenship, a multi-agent system that we call "MAS Conso" must meet this objective. The Figure 5 presents the overall architecture of the presented system.

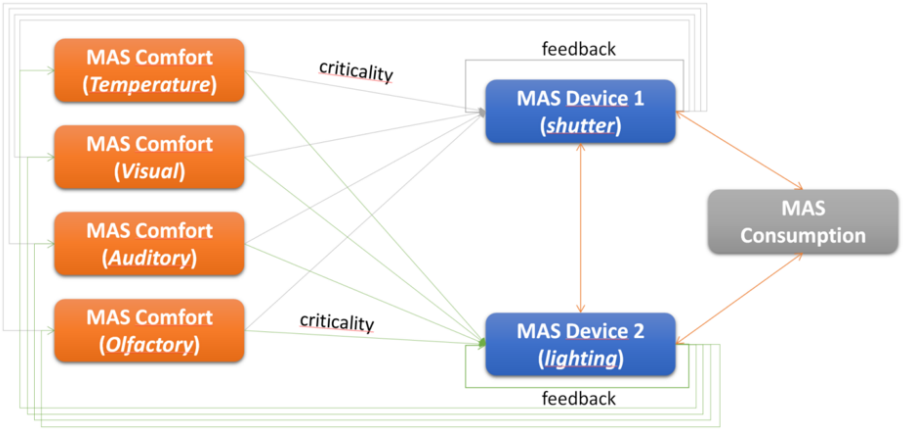

Fig. 5. Ambient system architecture

In this article, we will show the use of the AMAK framework to meet the goal of learning the power consumption of the environment using the construction of a "MAS Conso". The class "Environment" allows to build the world in which

the multi-agent system will operate. Here we can set up all the elements that will interact with the system, in our case we have built a simulator to generate the necessary inputs to the system such as the power consumption of the environment as well as the states of devices present in the environment. In addition, it is easy to be able to connect graphical output interfaces to evaluate or compare the work of the "MAS Conso". The Figure 6 presents a screenshot of a graphical output interface developed for the project.

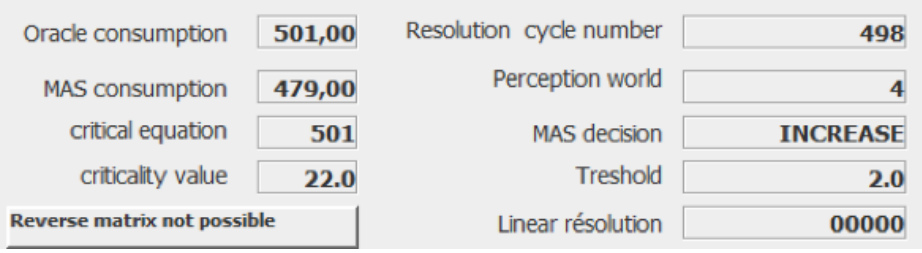

Fig. 6. Multi-Agent system state

The class "AMAS" allows to define the elements of our multi-agent system and to create the agents that will intervene. Using the method "onInitialAgentsCreation()", we create our "Estimation" agent and the "Device" agents that represent the devices of the environment. The Figure 7 presents the architecture of the "MAS Conso" and notably the links between agents. The "AMAS" class is scheduled using a "Scheduler" class that provides the ability to track the life cycles of the system, the beginning of cycles using the method "onSystemCycleBegin()" and the end using the method "onSystemCycleEnd()".

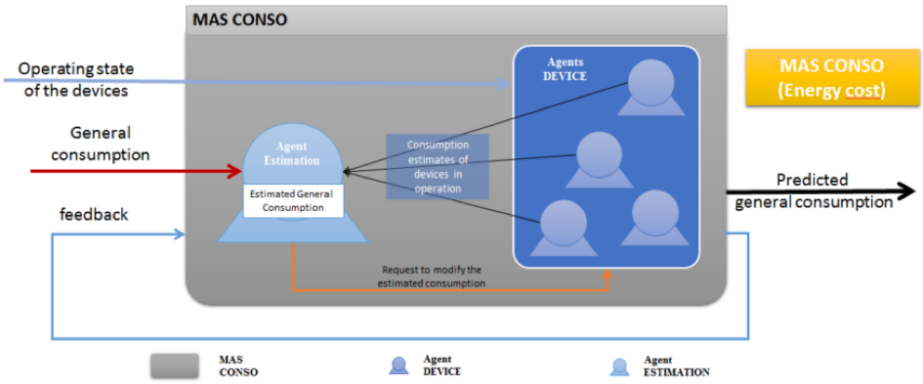

Fig. 7. MAS Conso architecture

Finally, each "Agent" is independent and respond to a schema of cooperative multi-agent systems, the class has an onPerceive() perception method that allows the agent to perceive a part of its environment, an onDecide() decision-making method making it possible to forge its decision on the solicitation of the neighborhood but also according to its own needs, and finally an action method onAct() allowing to act on its environment. In our "MAS CONSO", the "Estimation" agent aims at evaluating the power consumption of the environment. To maintain a cooperative state, he memorizes the states of the world he perceives at each change and looks for the most critical state to improve the predictions. To fulfill this objective, it perceives the total electricity consumption of the environment and solicits the device agents to obtain their prediction of consumption. It is then able to evaluate the tendency to take for Device Agents to improve their prediction. Considering these, they perceive the request of the Agent Estimate but, according to their own knowledge of their tendency, they will follow his recommendation or decide not to answer his request.

The flexibility of the framework makes it possible to develop these different agents while keeping them coherent. Also, this system requires to observe the evolution of different values such as the instantaneous energy consumption that is made possible thanks to the built-in tools *LxPlot*. Figure 2 is a screenshot of the data displayed using *LxPlot*. We evaluate here an environment composed of 10 devices, with each cycle of perception it is possible to observe the cooperating devices with a background of green color, those which do not follow the demand of the agent Estimation with a background of orange color and those who are not in working condition during this observation of the environment. In addition, the graphs show the consumption to be achieved for each device (red line) and the evolution of the predictions (blue line). In addition, the first graph allows the observation of the criticality defined by the distance between the actual consumption and the estimated consumption, as for the second graph, it shows the general evolution of the system on electricity consumption.

6 Conclusion

Table 1. Comparison with Existing Tools

	Known language	Intuitive	Flexible	Rendering capacities
Jade	+++	-	++	-
GAMA	-	++	+	++
NetLogo	-	++	-	+
SARL	-	++	++	-
MadKit	+++	+	+	+
AMAK	+++	++	+	-

The Table 1 shows the differences between the main existing tools and AMAK. As it can be seen, AMAK is less adapted to rendering data. However,

it is as much (or more) intuitive as the others thanks to its simple structure, the simplicity of the method names and the use of Java™. It is also flexible and doesn't require to learn a new language.

Despite the large choices of tools and frameworks to assist the development of Adaptive Multi-Agent Systems, the proposed framework seems relevant as it provides an effective and intuitive way to develop such systems. However, it lacks the possibility to reliably execute agent simultaneously. Moreover, the rendering capacities are currently limited. Also, it has been shown that the framework respects major generic concepts of multi-agent systems development and that it maintains a linear complexity evolution over the number of agents which means that multi-agent system developed with AMAK are scalable. The perspective of this work will be to improve the rendering capacities of the framework and to use it on various projects with different natures and requirements.

References

1. Axelrod, R., Hamilton, W.: The evolution of cooperation. Science 211(4489), 1390–1396 (mar 1981), http://www.ncbi.nlm.nih.gov/pubmed/7466396http://www.sciencemag.org/cgi/doi/10.1126/science.7466396
2. Bellifemine, F., Poggi, A., Rimassa, G.: JADE – A FIPA-compliant agent framework (1999), https://pdfs.semanticscholar.org/19f5/4048201ce8e416b74f3325266c34ae203f74.pdf
3. Blanc-Rouchossé, J.B.: Régulation auto-adaptative de réseau électrique intelligent. Tech. rep. (2017)
4. Bonjean, N., Mefteh, W., Gleizes, M.P.P., Maurel, C., Migeon, F.: Adelfe 2.0 pp. 1–45 (2012)
5. Crespi, V., Galstyan, A., Lerman, K.: Top-down vs bottom-up methodologies in multi-agent system design. Autonomous Robots 24(3), 303–313 (2008), https://link.springer.com/content/pdf/10.1007{\%}2Fs10514-007-9080-5.pdf
6. Di Marzo Serugendo, G., Gleizes, M.P., Karageorgos, A.: Self-organising Software. From Natural to Artificial Adaptation (2011), http://link.springer.com/chapter/10.1007/978-3-642-17348-6{_}5
7. Ferber, J.: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, vol. 222 (1999), http://jasss.soc.surrey.ac.uk/4/2/reviews/rouchier.html
8. GNU: GNU Lesser General Public License, https://www.gnu.org/licenses/lgpl-3.0.en.html
9. Gosling, J., McGilton, H.: The Java TM Programming Language Environment. No. May (1996), http://www.oracle.com/technetwork/java/intro-141325.html
10. Gutknecht, O., Ferber, J.: MadKit: A generic multi-agent platform. Proceedings of the fourth international conference on Autonomous agents - AGENTS '00 pp. 78–79 (2000), http://portal.acm.org/citation.cfm?id=336595.337048{\%}5Cnpapers2://publication/doi/10.1145/336595.337048{\%}5Cnhttp://portal.acm.org/citation.cfm?doid=336595.337048
11. Lemouzy, S.: Systèmes interactifs auto-adaptatifs par systèmes multi-agents auto-organisateurs: application à la personnalisation de l'accès à l'information (2011), http://thesesups.ups-tlse.fr/1303/
12. Perles, A.: AMAK, https://bitbucket.org/perlesa/amak

13. Perles, A.: AMAS Exercises, https://bitbucket.org/perlesa/amas-exercises
14. Rodriguez, S., Gaud, N., Galland, S.: SARL: A general-purpose agent-oriented programming language. In: Proceedings - 2014 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Workshops, WI-IAT 2014. vol. 3, pp. 156–165. IEEE (aug 2014), http://ieeexplore.ieee.org/document/6928174/
15. Taillandier, P., Vo, D.A., Amouroux, E., Drogoul, A.: GAMA: A simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 7057 LNAI, pp. 242–258. Springer, Berlin, Heidelberg (nov 2012), http://link.springer.com/10.1007/978-3-642-25920-3{_}17
16. Tisue, S., Wilensky, U.: Netlogo: A simple environment for modeling complexity. Conference on Complex Systems pp. 1–10 (2004), http://ccl.sesp.northwestern.edu/papers/netlogo-iccs2004.pdf