Intelligent Agents: Multi-Agent Systems

Alfredo Garro[1]

Department of Informatics, Modeling, Electronics and Systems Engineering, University of Calabria - via P. Bucci, cubo 41/C, 87036 Rende (Italy)

Max Mühläuser, Andrea Tundis

Department of Computer Science, Technische Universität Darmstadt - Hochschulstrasse 10, 64289 Darmstadt (Germany)

Matteo Baldoni, Cristina Baroglio

Department of Computer Science, University of Turin - Corso Svizzera 185, 10149 Turin (Italy)

Federico Bergenti

Department of Information Engineering, University of Parma - Parco Area delle Scienze 181/A, 43124 Parma (Italy)

Paolo Torroni

Department of Computer Science and Engineering, Alma Mater Studiorum - University of Bologna - Via Zamboni 33, 40126 Bologna (Italy)

Abstract

This chapter provides an overview of the fundamentals related to agent-oriented paradigm and Multi Agent Systems (MASs). Specifically, in the first part, the definition of agent and its main features are reported. The second section discusses multi-agent systems and their main features by focusing on the concepts of automation, coordination, norms and emerging behavior which characterize a MAS. Finally, in the sections third and forth, two main ways to approach and use such paradigm for developing respectively agent-based simulations and agent-based software, by breaking down the leading and most renowned refer-

URL: `www.unical.it` (Alfredo Garro), `www.tu-darmstadt.de` (Max Mühläuser, Andrea Tundis), `https://www.unito.it/` (Matteo Baldoni, Cristina Baroglio), `http://www.unipr.it/` (Federico Bergenti), `https://www.unibo.it` (Paolo Torroni)

[1]Corresponding author

ence software platforms, are discussed.

Keywords: Agent, Multi Agent System, Agent Based Modeling and Simulation, Agent Oriented Software Engineering, Autonomous Systems, Distributed Systems

---

1. Introduction

The agent is a metaphor that was natively originated in AI (Artifical Intelligence) (see works from Newell, and others). Its characteristics changed when the emphasis was put on multi-agent systems, rather than on single agents. A key event in the history of multi-agent systems was the recognition that agents can be used fruitfully to model and implement distributed systems. The creation of FIPA (Foundation for Physical Intelligent Agents) originates from the recognition that multi-agent systems are a novel and promising approach for the implementation and deployment of distributed systems. It is important to recognize that many applications of agents exist outside AI, because tools and techniques, which are intended to support one, are not necessarily good for the other. In particular, nowadays, multi-agent systems have two major applications outside of their traditional role in AI:

- Engineering of distributed systems, with the introduction of agent-oriented software engineering (AOSE) (e.g. agent-oriented methodologies, agent-oriented programming languages).
- Agent-based modeling and simulation (ABMS), which is what this chapter is mostly about.

Agents are entities that observe their environment and act upon it so as to achieve their own goals Russell and Norvig (2003); Wooldridge (2009). Two fundamental characteristics of agents are *autonomy* and *situatedness*. Autonomy means that agents have a sense-plan-act deliberative cycle, which gives them control of their internal state and behavior. Whereas, agents are situated

because they can sense, perceive, and manipulate the environment in which operate. The environment could be physical or virtual and it is understood by agents in terms of (relevant) data. Autonomy implies proactivity, i.e., the ability of an agent to take action towards the achievement of its objectives, without being solicited to do so.

From a programming perspective, agent-oriented programming was introduced by Shoham as "a specialization of *object-oriented programming*" Shoham (1993). The difference between agents and static objects is clear. Citing Wooldridge (Wooldridge, 2009, Section 2.2): (1) objects do not have control over their own behavior (this is summarized by the well-known motto "Objects do it for free; agents do it because they want it"), (2) objects do not exhibit flexibility in their behavior, and (3) in standard object models there is a single thread of control, while agents are inherently multi-threaded.

The agent-based paradigm also differs from the Actor Model Hewitt et al. (1973) (and from Active Objects, largely inspired by the latter). Actors, in fact, do not have neither goals nor purposes, even though their specification includes a process. Agents, instead, exploit their deliberative cycle (as control flow), possibly together with the key abstractions of belief, desire, and intention (as logic), so as to realize algorithms, for example processes for acting in their environment to pursue their goals. In other words objects "do it" for free because they are data, agents are processes and "do it" because it is functional to their objectives.

The environment, in which agents are situated, does not exhibit the kind of autonomy that is typical of agents although it may evolve, also thanks to an internal process. Its activity, however, is not meant to pursue a goal and this makes environments more similar to active objects.

The binomial agent–environment is formalized by modeling approaches like Demazeau (1995), where the environment is seen as providing "the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources", and in particular by the Agent & Artifact meta-model Omicini et al. (2008).

A system comprising a number of possibly interacting agents is called a *multiagent system*. At this level, it is widely recognized that further abstractions become handy, like organizations and interactions, aimed at enabling a meaningful and fruitful coordination of the autonomous and heterogeneous agents in the system. Thus, agents are not only situated in a physical environment, they are also situated in a social environment where they get into relationships with other agents and are subject to the regulations of the society they belong to. A normative multiagent system is "a multiagent system together with normative systems in which agents on the one hand can decide whether to follow the explicitly represented norms, and on the other the normative systems specify how and in which extent the agents can modify the norms" Boella et al. (2007). The impact on the agent's deliberative cycle is that agents can reason about the social consequences of their actions.

2. Multi-agent Systems

There is no single definition for the word agent, and there is no single definition for the term multi-agent system (MAS). Notably, major accepted definitions share commonalities, such as the way the agents interact in a system: via the shared environment, via structured messages (ontologies, interaction protocols). Indeed, a MAS can be defined in terms of interacting entities, and in particular the agents. Communication may vary from simple forms to sophisticated ones. A simple form of communication is that restricted to simple signals, with fixed interpretations. Such an approach was used by Georgeff in multi-agent planning to avoid conflicts when a plan was synthesized by several agents. A more elaborate form of communication is by means of a blackboard structure. A blackboard is a shared resource, usually divided into several areas, according to different types of knowledge or different levels of abstraction in problem solving, in which agents may read or write the corresponding relevant information for their actions. Another form of communication is by message passing between agents.

Autonomy is another major characteristic of agents, when defining a MAS, also referred to as "self-organized systems", enables them to find the best solution for their problems "without intervention". The main feature which is achieved when developing multi-agent systems, is flexibility, since a multi-agent system can be added to, modified and reconstructed, without the need for detailed rewriting of the application. The MAS also tends to prevent propagation of faults, self-recover and be fault tolerant, mainly due to the redundancy of components.

It is extremely important to distinguish between Automatic and Autonomous systems. Automatic systems are fully pre-programmed and act repeatedly and independently of external influence or control. It can be described as self-steering or self-regulating and it is able to follow an externally given path while compensating for small deviations caused by external disturbances. However, it is not able to define the path according to some given goal or to choose the goal dictating its path. Whereas, Autonomous systems, as a MAS, are self-directed toward a goal in that they do not require outside control, but rather they are governed through laws and strategies that clearly make a difference between traditional and multi agent systems. If machine learning techniques are utilized, autonomous systems can develop flexible strategies for themselves by which they select their behavior.

More in detail, *norms* are a fundamental ingredient of multi-agent systems that govern the expected behavior towards a specific situation. Through the norms, the desirable behaviors for a population of a natural or artificial community is represented. Indeed, they are generally understood as rules indicating actions that are expected to be pursued that are either obligatory, prohibitive, or permissive based on a specific set of facts. According to Hollander and Wu Hollander and Wu (2011), norms have been used to indicate constraints on behavior Shoham and Tennenholtz (1992), to create solutions to a macrolevel problem Zhang and Leezer (2009), and to serve as obligatory Verhagen (2000), regulatory, or control devices for decentralized systems Savarimuthu et al. (2008). The most common norms are:

- Conventions, which are natural norms that emerge without any enforcement Villatoro (2011). Conventions solve coordination problems when there is no conflict between the individual and the collective interests; for example, everyone conforms to desired behavior. Essential Norms that are used to solve or ease collective action problems when there is a conflict between an individual and the collective interests Villatoro (2011)Villatoro (2010). For example, the norm not to pollute urban streets is essential in that it requires individuals to transport their trash, rather than dispose of it on the spot, an act that benefits everyone.

- Regulative Norms. Regulative norms are intended for regulating activities by imposing obligation or prohibition in performing an action.

- Constitutive Norms, which are affirmed to produce new goal norms or states of affairs, for example, the rules of a game like chess.

- Procedural Norms that are categorized as objective and subjective. Objective procedural norms represent the rules that express how decisions are really made in a normative system, while subjective procedural norms represent the instrument for individuals working in a system, for instance, back-office procedures.

Coordination is another distinguishing factor of a MAS. In fact, an agent exists and performs its activity in a society in which other agents exit. Therefore, coordination among agents is essential for achieving the goals and acting in a coherent manner. Coordination implies considering the actions of the other agents in the system when planning and executing one agents actions. Coordination allows agents to achieve the coherent behaviour of the entire system. Coordination may imply cooperation and in this case the agent society works towards common goals to be achieved, but may also imply competition, with agents having divergent or even antagonistic goals. In this later case, coordination is important because the agent must take into account the actions of the others, for example competing for a given resource or offering the same service.

Another characterizing feature of a MAS is its *emergent behavior*. Emergent behavior in agents is commonly defined as behavior that is not attributed to any individual agent, but is a global outcome of agent coordination Zhengping et al. (2007). This definition emphasizes that emergent behavior is a collective behavior. There are also other definitions. Emergent behavior is that which cannot be predicted through analysis at any level simpler than that of the system as a whole Emergent behavior, by definition, is whats left after everything else has been explained Dyson (1997). This definition highlights the difficulty in predicting and explaining emergent behavior. If the behavior is predictable and explainable, then it will not be treated as emergent behavior and approaches could be designed to handle the behaviors. Emergence is also defined as the action of simple rules combining to produce complex results Rollings A. (2003). This definition states that the rules applied to the individuals can be quite simple, but the collective behavior of the group may turn out to be quite complex and unpredictable. Researchers have designed experiment to demonstrate this kind of situation. While it is true that all behavior comes from individuals, the interactions are what make things difficult to understand. Emergent behavior is essentially any behavior of a system that is not a property of any of the components of that system, and emerges due to interactions among the components of a system. Borrowing from biological models such as an ant colony, emergent behavior can also be thought of as the production of high level or complex behaviors through the interaction of multiple simple entitles. Some examples of emergent behaviors: Bee colony behavior where the collective harvesting of nectar is optimized through the waggle dance of individual worker bees; Flocking of birds cannot be described by the behavior of individual birds; Market crashes cannot be explained by "summing up" the behavior of individual investors.

Further details about multi-agent systems can be founded in Baldoni et al. (2010).

3. Agent-based Modeling and Simulation

Agent Based Modelling and Simulation (ABMS) refers to a category of computational models invoking the dynamic actions, reactions and intercommunication protocols among the agents in a shared environment, in order to evaluate their design and performance and derive insights on their emerging behaviour and properties Abar et al. (2017). Agents and multi-agent systems are entities that can be effectively used to model complex systems made of interacting entities. This is why they have been adopted to study biological and chemical systems, especially when the systems become too complex for the analytic tools available from Chemistry, Physics and Mathematical Physics. The fact that agents and multi-agent systems are abstractions with executable counterparts, the agents and the multi-agent systems that many tools support, contributed to suggest the use of agent technology to simulate biological and chemical systems. The size of simulated systems, and the high level of accuracy of simulated phenomena, calls for dedicated tools capable enable domain expert to describe simulations with little, or no, interest on the engineering issues related to distributed systems. Agent-based technology provides such tools and it supports domain experts in the construction of effective distributed systems with minimal emphasis on the inherent issues of large- scale distributed systems. Notably, even if dedicated tools are available, it is common to adopt tools designed to support agent-oriented software engineering in the scope of ABMS. In particular, a number of agent-based tools to support modeling and simulation of complex and/or distributed systems are available Allan (2009):

1. Netlogo: it is a multi-agent programmable modeling environment which allows the simulation of natural and social phenomena. It is particularly well suited for modeling complex systems developing over time. Indeed, modelers can give instructions to hundreds or thousands of "agents" all operating independently. This makes possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from their interaction. It comes with a large library

of existing simulations, both participatory and traditional, that one can use and modify in different domains such as social science and economics, biology and medicine, physics and chemistry, and mathematics and computer science. In the traditional NetLogo simulations, the simulation runs according to rules that the simulation author specifies. A further feature of NetLogo is HubNet, a technology that lets you use NetLogo to run participatory simulations. HubNet adds a new dimension to NetLogo by letting simulations run not just according to rules, but by direct human participation.

2. FLAME: it is a generic agent-based modelling system which can be used to development applications in many areas. Models are created based upon a model of computation called (extended finite) state machines. The framework can automatically generate simulation programs that can run models efficiently on HPCs. It produces a complete agent-based application which can be compiled and built on the majority of computing systems ranging from laptops to HPC super computers. Furthermore, FLAME provides a Model Library which is a collection of relatively simple models that illustrate the use of FLAME in different applications.

3. AnyLogic: it is a simulation tool that supports all the most common simulation methodologies in place today: System Dynamics, Process-centric (AKA Discrete Event), and Agent Based modeling. Its visual development environment significantly speeds up the development process. It has applicational models in ManufacturingLogistics, Supply ChainsMarkets, Competition Business Processes Modeling Healthcare, Pharmaceuticals Simulation Pedestrian Traffic Flows Information, Telecommunication Networks Simulation Modeling Social Process, Marketing Simulation Asset Management, Financial Operations with Simulation Modeling Warehouse Operations and Layout Optimization.

4. Repast: The Repast Suite is a family of advanced, free, and open source agent-based modeling and simulation platforms that have collectively been

under continuous development for many years. Repast Simphony is a richly interactive and easy to learn Java-based modeling system that is designed for use on workstations and small computing clusters. An advanced version is called Repast for High Performance Computing, which is a lean and expert-focused C++-based modeling system, that is designed for use on large computing clusters and supercomputers.

5. Jason: It is an interpreter for an extended version of AgentSpeak, that has been one of the most influential abstract languages based on the BDI architecture. Jason implements the operational semantics of that language, Strong negation, so both closed-world assumption and open-world are available. Annotations in beliefs are used for meta-level information and annotations in plan labels. One of the best known approaches for the development of cognitive agents is the BDI (Beliefs-Desires-Intentions) architecture. It provides the possibility to run a multi-agent system distributed over a network.

6. Framsticks: it is a three-dimensional life simulation project. Both mechanical structures (bodies) and control systems (brains) of creatures are modeled. It is possible to design various kinds of experiments, including simple optimization, co-evolution, open-ended and spontaneous evolution, distinct gene pools and populations and modeling of species and ecosystems. Users of this software work on evolutionary computation, artificial intelligence, neural networks, biology, robotics and simulation, cognitive science, neuro-science, medicine, philosophy, virtual reality, graphics, and art.

7. Gephi: it is an interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs. It allows (i) Exploratory Data Analysis: intuition-oriented analysis by networks manipulations in real time; (ii) Link Analysis: revealing the underlying structures of associations between objects, in particular in scale-free networks;(iii) Social Network Analysis: easy creation of social data connectors to map community organizations and small-world networks; (iv)

Biological Network analysis: representing patterns of biological data; (v) Poster Creation: scientific work promotion with hi-quality printable maps. It works mainly with metrics related to centrality, degree (power-law), betweenness, closeness, density, path length, diameter, HITS, modularity, clustering coefficient.

8. Stanford Network Analysis Platform (SNAP): it is a general purpose, high performance system for analysis and manipulation of large networks. It easily scales to massive networks with hundreds of millions of nodes, and billions of edges. It efficiently manipulates large graphs, calculates structural properties, generates regular and random graphs, and supports attributes on nodes and edges.

This is a list of the most popular simulation tools that can be used to model, simulate and analyzes complex systems by adopting agent oriented paradigm.

4. Agent-Oriented Software Engineering

Since its early days in late 1960s, *software engineering* has been constantly facing the problem of better understanding the sources of complexity in software systems. Over the last three decades, the complexity of interactions among parts has been progressively identified as one of the most significant sources of such a complexity. Software systems that contain a—possibly large—number of interacting parts are critical, especially when the graph of interactions changes dynamically, parts have their own thread of control, and parts are engaged in interactions governed by complex protocols (see, e.g., Wooldridge and Ciancarini (2001) for an in-depth discussion). As a consequence, a major research topic of software engineering has been the development of techniques and tools to understand, model, and implement systems in which interactions is the major source of complexity. This has led to the search for new computational abstractions, models, and tools to reason and to implement MASs, which have been recognised as prototypical examples of such systems. *AOSE* (*Agent-Oriented*

Software Engineering) is an emerging paradigm of software engineering that has been developed to target the inherent complexity of analysing and implementing MASs (see, e.g., Bergenti et al. (2004) for a comprehensive reference on the subject). A number of AOSE methodologies have been proposed over the last two decades, and Kardas (2013) provides a recent survey of the state of the art of such methodologies.

Besides methodologies, the research on AOSE have been constantly interested in delivering effective tools to implement MASs. *Agent platforms* are examples of such tools intended to offer generic runtime environments for the effective deployment and execution of MASs. A number of platforms have been proposed over the years, and Kravari and Bassiliades (2015) proposes a recent attempt to enumerate the platforms that survived the test of time. One of the most popular agent platforms is *JADE* (*Java Agent DEvelopment framework*), as described in Bellifemine et al. (2007), which consists of a middleware and a set of tools that help the development of distributed, large-scale MASs. JADE is widely used for industrial and academic purposes and it can be considered as a consolidated tool. Just to cite a notable industrial example, JADE has been in daily use for service provision and management in Telecom Italia for more than six years, serving millions of customers in one of the largest and most penetrating broadband networks in Europe (see Bergenti et al. (2015) for further details). In order to effectively address the inherent issues of the high-profile scenarios that agent platforms target, specific tools are needed to assist the development of complex functionality, and to promote the effective use of the beneficial features of agent technology as a software development technology. Nevertheless, approaching AOSE with the help of agent platforms alone is often perceived as a difficult task for two main reasons. First, the continuous growth of agent platforms has been increasing their inherent complexity, and the number of implementation details that the programmer is demanded to master for the construction of MASs is equally grown. Second, the choice of mainstream programming languages as the unique option to use agent platforms is now considered inappropriate in many situations because such languages do not natively

offer the needed abstractions for the effective concretization of AOSE.

The interest in *agent programming languages* dates back to the introduction of agent technologies and, since then, it has grown rapidly. As a matter of fact, agent programming languages turned out to be especially convenient to model and develop complex MASs. Nowadays, agent programming languages represent an important topic of research and they are widely recognized as important tools in the development of agent technologies, in contrast with mainstream languages, that are often considered not suitable to effectively implement AOSE. Agent programming languages are usually based on specific agent models and they aim at providing specific constructs to adopt such models at a high level of abstraction. The features of the various agent programming languages proposed over the years may differ significantly, concerning, e.g., the selected agent mental attitudes (if any), the integration with an agent platform (if any), the underlying programming paradigm, and the underlying implementation language. Some classifications of relevant agent programming languages have already been proposed to compare the characteristics of different languages and to provide a clear overview of the state of the art. Bădică et al. (2011) classifies agent programming languages on the basis of the use of mental attitudes. According to such a classification, agent programming languages can be divided into: *Agent-Oriented Programming* (*AOP*) languages, *Belief Desire Intentions* (*BDI*) languages, hybrid languages—which combine the two previous classes—and other languages—which fall outside previous classes. It is worth noting that such a classification recognizes that BDI languages follow the *agent-oriented programming* paradigm, as defined in Shoham (1993), but it reserves special attention to them for their notable relevance in the literature. Bordini et al. (2006) proposes a different classification, where languages are divided into declarative, imperative, and hybrid. Declarative languages are the most common because they focus on automatic reasoning, both from the AOP and from the BDI points of view. Some relevant imperative languages have also been proposed, and most of them were obtained by adding specific constructs to existing procedural programming languages. Finally, the presence (or absence) of a host language is

an important basis of comparison among agent programming languages.

Even if early proposals dates back to late 1990s, AOSE is still at an early stage of evolution. While there are many good arguments to support the view that agents represent an important direction for software engineering, there is still need of actual experience to underpin these arguments. Methodologies and tools to support the deployment of MAS are beginning to become accepted for mission-critical applications, but slowly. Although a number of agent-oriented analysis and design methodologies have been proposed, there is comparatively little consensus among them. In most cases, there is not even agreement on the kinds of concepts the methodology should support. But, the research on AOSE is still active, and it has been recently revitalized by the view of AOSE in terms of *model-driven development*, as summarized in Kardas (2013).

5. Guidelines/Perspectives

Agents and multi-agent systems have been used to study and to simulate complex systems in different application domain where physical factor are present for energy minimizing, where physical objects tend to reach the lowest energy consumption possible within the physically constrained world. Furthermore, MAS have been intensively exploited to analyzed through simulation biological and chemical systems. The literature reports on various successful uses of the abstractions and of their executable tools. Notably, the study of the benefits and the costs of adopting agents and multi-agent systems to support scientific studies of biological and chemical systems has not been approached extensively.

A well-known application field, where agents and MAS have been successful exploited regarding the study of biological phenomena, is the protein synthesis, a common and relevant phenomenon in nature. In this field, several approaches for predicting the three-dimensional structure of proteins are, for instance, available in literature Jennings et al. (1998). Several of them exploit the chemical or physical properties of the proteins (e.g. Standley et al. (1998) Dudek et al.

(1998) work on energy minimization whereas Galaktionov and Marshall (1995), Sabzekar et al. (2017) uses intra-globular contacts); other ones use evolutionary information Piccolbon and Mauri (1998). Some tools for the prediction of the three-dimensional structure of proteins have been presented in Douguet and Labesse (2001)Gough et al. (2001)Meller and Elber (2001).

This shows that the autonomy of agents, their normed freedom of interaction, and the possibility of deploying large-scale systems with minimal care on issues related to distributed systems are few of the major benefits of approaching modeling and simulation of complex systems with agents. Furthermore, they closely represent how natural systems work by distributing a problem among a number of reactive, autonomous, deliberative, pro-active, adaptive, possibly mobile, flexible and collaborative entities. All these properties make agent technology more suited than other ones (e.g., a distributed system, an artificial intelligent system) for facing the challenges and high level of complexity of biological system and related phenomena.

References

Abar, S., Theodoropoulos, G.K., Lemarinier, P., OHare, G.M. (2017). Agent based modelling and simulation tools: A review of the state-of-art software. Computer Science Review 24, 13 – 33.

Allan, R. (2009). Survey of agent based modelling and simulation tools .

Bădică, C., Budimac, Z., Burkhard, H.D., Ivanovic, M. (2011). Software agents: Languages, tools, platforms. Computer Science and Information Systems 8, 255–298.

Baldoni, M., Baroglio, C., Mascardi, V., Omicini, A., Torroni, P. (2010). Agents, multi-agent systems and declarative programming: What, when, where, why, who, how?, in: Dovier, A., Pontelli, E. (Eds.), A 25-Year Perspective on Logic Programming: Achievements of the Italian Association for Logic Programming, GULP, Springer. pp. 204–230.

Bellifemine, F., Caire, G., Greenwood, D. (2007). Developing multi-agent systems with JADE. Wiley Series in Agent Technology, John Wiley & Sons.

Bergenti, F., Caire, G., Gotta, D. (2015). Large-scale network and service management with WANTS, in: Industrial Agents: Emerging Applications of Software Agents in Industry, Elsevier. pp. 231–246.

Bergenti, F., Gleizes, M.P., Zambonelli, F. (Eds.) (2004). Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook. Springer.

Boella, G., van der Torre, L.W.N., Verhagen, H. (2007). Introduction to normative multiagent systems, in: Boella, G., van der Torre, L.W.N., Verhagen, H. (Eds.), Normative Multi-agent Systems, 18.03. - 23.03.2007, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

Bordini, R.H., Braubach, L., Dastani, M., Seghrouchni, A.E.F., Gomez-Sanz, J.J., Leite, J., O'Hare, G., Pokahr, A., Ricci, A. (2006). A survey of programming languages and platforms for multi-agent systems. Informatica 30.

Demazeau, Y. (1995). From interactions to collective behaviour in agent-based systems, in: Proceedings of the 1st. European Conference on Cognitive Science, Saint-Malo. pp. 117–132.

Douguet, D., Labesse, G. (2001). Easier threading through web-based comparisons and cross-validations. Bioinformatics 17, 752–753.

Dudek, M., Ramnarayan, K., Ponder, J. (1998). Protein structure prediction using a combination of sequence homology and global energy minimization: Ii. energy functions. Journal of Computational Chemistry 19, 548–573.

Dyson, G.e.B. (1997). Darwin Among the Machines:The Evolution of Global Intelligence.

Galaktionov, S.G., Marshall, G.R. (1995). Properties of intraglobular contacts in proteins: An approach to prediction of tertiary structure, pp. 326–335.

Gough, J., Karplus, K., Hughey, R., Chothia, C. (2001). Assignment of homology to genome sequences using a library of hidden markov models that represent all proteins of known structure. Journal of Molecular Biology 313, 903–919.

Hewitt, C., Bishop, P., Steiger, R. (1973). A universal modular ACTOR formalism for artificial intelligence, in: Nilsson, N.J. (Ed.), Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, August 1973, William Kaufmann. pp. 235–245.

Hollander, C.D., Wu, A.S. (2011). The current state of normative agent-based systems. J. Artificial Societies and Social Simulation 14.

Jennings, N., Sycara, K., Wooldridge, M. (1998). A roadmap of agent research and development. Autonomous Agents and Multi-Agent Systems 1, 7–38.

Kardas, G. (2013). Model-driven development of multiagent systems: A survey and evaluation. The Knowledge Engineering Review 28, 479–503.

Kravari, K., Bassiliades, N. (2015). A survey of agent platforms. Journal of Artificial Societies and Social Simulation 18, 11.

Meller, J., Elber, R. (2001). Linear programming optimization and a double statistical filter for protein threading protocols. Proteins: Structure, Function and Genetics 45, 241–261.

Omicini, A., Ricci, A., Viroli, M. (2008). Artifacts in the A&A meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems 17, 432–456. Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.

Piccolbon, A., Mauri, G. (1998). Application of evolutionary algorithms to protein folding prediction. Lecture Notes in Computer Science (including

subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 1363, 123–135.

Rollings A., A.E. (2003). Andrew Rollings and Ernest Adams on Game Design.

Russell, S.J., Norvig, P. (2003). Artificial Intelligence: A Modern Approach. Pearson Education. 2 edition.

Sabzekar, M., Naghibzadeh, M., Eghdami, M., Aydin, Z. (2017). Protein -sheet prediction using an efficient dynamic programming algorithm. Computational Biology and Chemistry 70, 142–155.

Savarimuthu, B., Purvis, M., Purvis, M. (2008). Social norm emergence in virtual agent societies, pp. 1485–1488.

Shoham, Y. (1993). Agent-oriented programming. Artificial Intelligence 60, 51–92.

Shoham, Y., Tennenholtz, M. (1992). On the synthesis of useful social laws for artificial agent societies (preliminary report), pp. 276–281.

Standley, D., Gunn, J., Friesner, R., McDermott, A. (1998). Tertiary structure prediction of mixed / proteins via energy minimization. Proteins: Structure, Function and Genetics 33, 240–252.

Verhagen, H. (2000). Norm autonomous agents.

Villatoro, D. (2011). Self-organization in decentralized agent societies through social norms, pp. 1297–1298.

Villatoro, D.e.a. (2010). Of social norms and sanctioning: A game theoretical overview. International Journal of Agent Technologies and Systems (IJATS) 2, 115.

Wooldridge, M., Ciancarini, P. (2001). Agent-oriented software engineering: The state of the art, in: Agent-Oriented Software Engineering, Springer-Verlag. pp. 1–28.

Wooldridge, M.J. (2009). Introduction to multiagent systems, 2nd edition. Wiley.

Zhang, Y., Leezer, J. (2009). Emergence of social norms in complex networks, pp. 549–555.

Zhengping, L., Cheng, H., Malcolm, Y. (2007). A survey of emergent behavior and its impacts in agent-based systems, pp. 1295–1300.