Chapter 18

Agent-Based Models of Cellular Systems

Nicola Cannata, Flavio Corradini, Emanuela Merelli, and Luca Tesei

Abstract

Software agents are particularly suitable for engineering models and simulations of cellular systems. In a very natural and intuitive manner, individual software components are therein delegated to reproduce "in silico" the behavior of individual components of alive systems at a given level of resolution. Individuals' actions and interactions among individuals allow complex collective behavior to emerge. In this chapter we first introduce the readers to software agents and multi-agent systems, reviewing the evolution of agent-based modeling of biomolecular systems in the last decade. We then describe the main tools, platforms, and methodologies available for programming societies of agents, possibly profiting also of toolkits that do not require advanced programming skills.

Key words: Agent-based modeling and simulation, Agent-oriented software engineering, Behavioral models of biological systems, Multi-agent systems

1. Introduction

A biological system can be simulated in a natural way by a society of software agents. Actually, the goal of systems biology is that of providing mathematical and computational models of life, ranging from the molecular level to those of molecular complexes, cells, tissues, organs, organisms, communities of organisms, and ecosystems. Models permit analysis and simulation, aiming at reproducing "in silico" peculiar aspects of life. Among the computational models, behavioral models are intuitively closer to the reality than pure mathematical models. In fact, they rely on software entities whose characteristics resemble those of individual "components" of alive systems at some chosen resolution. The simulated behavior and interactions of those individuals at some given level of abstraction should let emerge at higher level the properties of the collective behavior of the individuals' population. For instance, building a model with interacting enzymes and metabolites (i.e., the individuals) should let emerge biochemical kinetic laws of a well-mixed solution (i.e., the population) of enzymes and metabolites. Modeling the behavior and the interactions of the individuals in a community of organisms should reproduce their social attitudes and rules. An effective model should allow to "zoom-in" and "zoom-out," letting emerge at a higher level of abstraction the structure and properties determined by lower levels.

1.1. Software Agents and Multi-agent Systems

Complex software systems can be designed around autonomous interacting components. Robin Milner suggested to call each of the several parts of a software system as agent, with its own identity, which persists through time (1). The name agent derives from the Latin "agere," to act. A software agent is definitively a piece of software that acts for a user or for another program in a relationship of agency. We can metaphorically think to some kind of agreement to act on one's behalf. Such action on behalf of implies that the agent has the authority to decide which action (if any) is appropriate. Wooldridge defined an agent as a computer system *situated* in some environment and capable of *autonomous*, *flexible* action in that environment in order to meet its design objectives (2).

The *autonomy* property ("autonomous agent") implies that the agent has the control over its internal state and over its own behavior. A comparison between agent-based programming and object-oriented programming (OOP) can better explain this property. Both the methodologies permit to define an abstraction of an "internal state." In the case of OOP, the state encapsulated in an object can be controlled from the software entity controlling the object and invoking its methods. In the case of an agent, instead, the agent itself has the full control on the state it encapsulates.

The *situatedness* property ("situated agent") implies that the agent perceives its environment through some "sensors" and is able to act in the environment through some "actuators." Usually sensors and actuators are software ones but we can think also to robotic agents in which they actually are physical devices. The environment in which the agent is situated is typically dynamic, likely open, unpredictable, and populated from other agents (i.e., multi-agent).

The *flexibility* property ("flexible agent") can be declined in different ways. A *reactive* agent responds in timely fashion to environmental change. An *adaptive* agent responds to environmental change according to its internal state. A *proactive* agent acts in anticipation of future goals.

An additional property of agents is mobility. A mobile agent is able to move from one to another distributed environment.

A multi-agent system (MAS) consists of a number of agents interacting with each other in a dynamic environment (3). Agents of a MAS will be acting with their different goals and motivations. Furthermore, agents will require the ability to communicate, cooperate, and perceive other agents. A well-designed MAS is the one that achieves a global task through the tasks of the single agents and their interactions. The first design step consists in decomposing a global task into distributable subcomponents, yielding tractable tasks for each agent. Once the agents responsible for all the different tasks have been depicted, communication channels among them must be established. In this way sufficient information can be provided to each agent in order for it to achieve its task. Finally the agents must be coordinated in a way that they cooperate on the global task, or at the very least, in order to prevent them to pursue conflicting strategies in trying to achieve their tasks. The fundamental problem, in analyzing/designing a MAS, is in determining how the combined actions of a large number of agents lead to "coordinated" behavior on the global task.

1.2. Agent-Based Modeling

Biological systems are complex ones, i.e., many of their components are coupled in a nonlinear fashion (4). They are characterized by variables having complicated, discontinuous behaviors over time. Furthermore, they exhibit the emergence property, i.e., complex patterns do emerge from simpler interaction rules. The global behavior of such a system can be determined by defining the lower-level interaction rules among its components. Developing software for agent-based systems can profit of modern software engineering techniques, including decomposition, abstraction, and organization. A problem can be divided into smaller, manageable subproblems. Some details of a problem can be chosen to be modeled, while others can be ignored. The relationships among the various system components can be identified and managed. Software agents are situated in space and time and have some properties and some sets of local interaction rules. Though "intelligent," they cannot by themselves deduce the global behavior resulting from their dynamic interactions. An agent-based system usually evolves from the microlevel to the macrolevel. Usually agent-based modeling (ABM) adopts a bottom-up design strategy rather than a top-down one. Agents are commonly assumed to have well-defined bounds and interfaces, as well as spatial and temporal properties, including such dynamic properties as movement, velocity, acceleration, and collision. Agent-based systems also allow for easy modification of interaction rules or behavior, as well as for viewing agents or groups of agents at different levels of abstraction. Modeling a system with a bottom-up approach requires that every individual agent's behavior be described. The greater the number of details that go into describing the behavior of the system, the greater is the computational power that is required to simulate the behaviors of all constituent agents. This is a limitation in modeling large systems using ABM (5). A reasonable approach is to provide several levels of abstraction and granularity, which can be chosen depending on the level of detail needed and the computational resources available.

In general, ABMs are a class of computational models for simulating actions and interactions of multiple entities in an attempt to re-create and predict the appearance of complex phenomena. Modelers can observe the emergence of phenomena from the lower level of systems to higher level. The result can sometime be an equilibrium or an emergent pattern (e.g., cycles). Simple behavioral and interaction rules are able to generate complex behavior. Individual agents are usually characterized as rational, acting in what they perceive as their interest, e.g., economic benefit or social status, using heuristic or simple decision-making rules. They can also experience "learning."

ABM has been used since the 1990s, for instance, to analyze financial and business issues, to describe social phenomena, to model populations and ecosystems evolution, wars, epidemies spreading, and in general to solve technological problems. Examples of applications include human organizations, consumer behavior, logistic, distribution, supply chains, stock markets. Other examples concern crowd behavior, e.g., in public places and in emergency situations, vehicles movement and traffic congestion, growth and decline of ancient civilizations, migrations, and social network effects.

1.3. Review of ABM of Cellular Systems

Computational models of cellular Computational models of cellular systems began to appear even before the rapid establishment of systems biology. This recent—its "manifesto" (6) has been published in 2001—multidisciplinary research area builds on the unprecedented availability of biomolecular data (characterized by the suffix "-omes"). The high-throughput technologies changed the focus from the identification and analysis of a single molecule at time (e.g., gene, protein, metabolite) to the systematic simultaneous characterization of whole populations of molecules (e.g., genome, proteome, metabolome). The systemic approach can be applied at different levels of abstraction; therefore, considering the nature, properties, and interaction of the entities at different levels of the structural hierarchy of life, from molecules to ecosystems. For instance, molecular systems biology concentrates its attention at the molecular level taking into account metabolic networks, signal transduction networks, and genetic regulatory networks.

Several computational approaches have been proposed to model cellular signaling pathways (e.g., Boolean networks, Petri nets, Artificial Neural Networks, Cellular Automata) (7). In Cellular Automata (CA) (8) information is inherently processed in parallel. With CA the interactions between cells or molecules can be modeled in a matrix, where the state of an element of the matrix depends on the states of neighboring elements.

Also agents permit to model a cell as a society of autonomous agents acting in parallel. Agents communicate between them through messages and have the "cognitive" capabilities to interact with the surrounding environment. In Cellulat each agent communicates with the others through the creation or modification of signals on a shared data structure named "blackboard" (7). The blackboard permits a very abstract representation of cellular compartments related to the signaling pathways, whereas the different objects created on the blackboard represent signal molecules, activation or inhibition signals, or other elements belonging to the intracellular medium.

Cellular systems can be generally simulated at three different scales of resolution: the nanoscale (10^{-10}), the mesoscale (10^{-8}), and the continuum or macroscale (10^{-3}) (9). At the atomic level (nanoscale), molecular dynamics (MD) and Brownian dynamics (BD) are typically used to model the behavior of a limited number of atoms over relatively short periods of time and space. MD is fully deterministic and remarkably accurate over the short temporal and spatial scales that are normally simulated. Because of their accuracy, MD techniques are well suited to simulate state or conformational changes, predict binding affinities, investigate single molecule trajectory, and model stochastic or diffusive interaction between small numbers of macromolecules. In order to model molecular events involving large numbers of molecules or macromolecules over extended periods of time and space, continuum approaches are usually adopted. At the macroscale, molecules essentially lose their discreteness and become infinitely small and infinitely numerous. The system of interest can be described with ordinary (ODE) or partial (PDE) differential equations. However not all sets of differential equations are solvable nor are all systems suitable to be described by differential equations. Due to their continuum nature, the solutions to differential equation always generate smooth curves or surfaces that fail to capture the true granularity or stochasticity of living system. Discontinuities, state changes, irregular geometries, or discreteness with low number of molecules are not easily described by differential equations. Cellular systems can be effectively and efficiently modeled at the mesoscale (10^{-8}–10^{-7} m). At that level, macromolecules can still be treated as discrete objects occupying a defined space or volume. The possibility of representing single macromolecules allows mesoscale models to display the stochasticity or granularity found in real molecular systems. At this scale Brownian motion dominates over the other forces and therefore significant dynamic simplification are possible, allowing very long time scale and very large number of entities or reaction to be modeled. Wishart et al. (9) propose to perform mesoscale simulation by means of dynamic cellular automata (DCA), an hybrid between the classical CA and agent systems. DCA rely on a 2D grid to roughly resemble cell compartments. Simulating Brownian motion and using simple pairwise interaction rules, DCA can be used to model spatial and temporal phenomena that include macromolecular diffusion, viscous drag, enzyme rate processes, metabolic reactions, and genetic circuits. SimCell (9) was an easy-to-use graphical simulator able to model a set of molecular components and a collection of interaction rules. The five represented components are small molecules (metabolites, ligands), membrane proteins, soluble proteins or RNA molecules, DNA molecules (non-mobile), and membrane (non-mobile). Membranes describe boxes or borders and may be permeable or impermeable to certain molecules. They may be used to define cell compartments, including the nucleus or other organelles.

Troisi et al. (10) define an agent as a computer system that decides for itself. After sensing the environment, it takes decisions based on some rules. ABM was applied to the theoretical problem of molecular self-assembly. In the problem, a system evolves from a separated to an aggregated state following a combination of stochastic, deterministic, and adaptive rules. Agents are identified with a molecule or a group of molecules. Molecules have rigid shapes formed by four contiguous cells of a 2D square lattice. Cells can be of three types (neutral, positive, negative) and their interactions are nearest-neighbor only. The interactions mimic van der Walls attraction between all cell types and Coulomb repulsion/attraction between similarly/different charged cells. The published results show that it is possible to devise a combination of stochastic, deterministic, and adaptive rules that lead a disordered system to organize itself in an ordered low-energy configuration.

Using agent-based technology, Emonet et al. developed AgentCell (11), a model to study the relationships between stochastic intracellular processes and behavior of individual cells. Korobkova et al. showed that behavioral variability of an individual cell could be the result of the stochastic nature of molecular interactions in molecular signaling pathways (12). Consequently, even genetically identical cells can exhibit different behaviors. Stochastic molecular events in signaling pathways play a significant role in single-cell behavior. AgentCell studies how molecular noise influences the behavior of a swimming cell in a 3D environment. The model is able to reproduce experimental data for bacterial chemotaxis, one of the best characterized biological system. In the model, each bacterium is an agent equipped with its own chemotaxis network, motors, and flagella. A piece of software (scheduler) is responsible for stepping the system through time. Scheduling consists of keeping a global clock, updating the clock to the next event, and maintaining a sorted list events. Each agent inserts its own future events inside the scheduler's list of events.

From this flashback we may observe how the community of cellular ABM was already trying to answer to the necessity that was then arising in the systems biology community. ABMs, establishing a correspondence between a population of autonomous, interacting, more or less "intelligent" software components and a population of different species of biomolecules, permit more realistic simulations than models based on differential equations. Phenomena dictated from low numbers of molecules or arising from spatial effects can be easily represented.

The necessity of representing space emerges as the "final frontier" (13) in systems biology. Lemerle et al. underline that only at that time the technological and theoretical advances begun to allow the simulation of detailed kinetic models of biological systems reflecting the stochastic movement and reactivity of individual molecules within cellular compartments. At that time some model specification languages like CellML (14) and SBML (15) based on the XML markup language began to establish. SBML rapidly became a de facto standard, permitting the interchange of models and reproducibility of simulation on compatible simulation environments and tools. Lemerle et al. (13) consider also a number of issues relevant to cellular simulation. The first of all is obviously space representation. Is it continuum space supported, e.g., in systems based on differential equation? Is space discretized, i.e., represented as a 2D regular lattice or as 3D voxels ("voxelizations")? Another fundamental issue is the representation of molecular entities individuals. Are species represented as concentrations, like in ODE's systems, or as population (i.e., numbers), or taking into account each individual (like with agents)? Another issue concerns the enabling conditions for biomolecular reactions to happen in the simulated cellular environment. Depending on the spatial approach adopted, the reaction can be simulated when a collision between two suitable species is detected or when the two species are in the same voxel (or in neighboring ones). Important issues are also the geometry of the model and the movement itself.

Also Takahashi et al. (16) highlight the importance of a spatial representation in systems biology. In "particle" space, molecules are represented as individual particles with positions in a continuum space. Particles are usually given motions according to some kind of force equations that are numerically integrated to advance time. Reactions are represented as collisions between particles. "Discrete" space representation ("mesoscopic") discretizes the space either by subvolumes (voxels) of an identical shape (typically cubic) or by mean of a regular lattice. Some methods allow at most one particle to occupy a lattice site. Others allow multiple particles to reside in a single lattice site. Detailed space representation permits to deal with very important issue that otherwise cannot be taken into account, e.g., molecular crowding. Extremely high protein density in the intracellular space can actually alter protein activities and break down classical reaction kinetics.

A goal in biomodeling research is to understand the linkage from molecular level events to the emerging behavior of the system (17). This task requires having plausible, adequately detailed design plans for how components (single and composite) at various system levels are thought to fit and function together. Experimentation is then used to reconcile different design plan hypotheses. To actually demonstrate that a design plan is functionally plausible, it is however necessary to assemble individual components according to a design, and then show that the constructed device exhibits behaviors that match those observed in the original experiment. Tang et al. (17) apply this methodology presenting a multilevel, agent-based, model that represents the dynamics of rolling, activation, and adhesion of individual leukocytes in vitro.

Vallurupalli and Purdy present an agent-based 3D model of phage lambda, with its two characteristic phases: lysogenic and lytic (4). This widely studied gene regulatory system has been modeled using the unified modeling language (UML) and simulated using the breve (18) 3D visualization engine. In their article it is stressed how a complex behavior emerges from local agent interactions. The agent approach allows also to study how individual parameters affect overall system behavior.

Differently from Emonet et al., the model of bacterial chemotaxis proposed by Guo et al. (19) is a hybrid one. Biological cells are modeled as individuals (agents) while molecules are represented by quantities. This hybridization in entity representation entails a combined modeling strategy with agent-based behavioral rules and differential equations, thereby balancing the requirements of extendible model granularity with computational tractability. In their assay of 10^3 cells and about 10^6 molecules they are able to produce cell migration patterns that are comparable to laboratory observations.

A valuable issue raised by Guo and Tay (20) is that of event scheduling. The update scheme of a MAS model refers to the frequency of agent state updates and how these are related in temporal order. In contrast to verifiable agent behavioral rules at the individual level, the update scheme is a design decision made by the model developer at the systems level that is subject to realism and computational efficiency issues that directly affect the credibility and the usefulness of the simulation results. Usually simulation adopts uniform time-step update scheme. Guo and Tay, in modeling immunological phenomena, characterized by multi-timescales, suggest to adopt event-scheduling based asynchronous update scheme. The scheme allows arbitrary smaller timescales for realism and avoids unnecessary execution and delays to achieve efficiency. In their article the application of the event-scheduling update scheme to realistically model the B cell life cycle is presented. The simulation results show a significantly reduced execution time (40 times faster) and also reveal the conditions where the event-scheduling update scheme is superior.

Indeed remarkable has been the research activity on ABM of cellular systems at the University of Sheffield. A proof-of-concept was the Epitheliome (21), an ABM, in which there is a one-to-one correspondence between biological cells and software agents. The model is able to predict the emergent behavior resulting from the interaction of cells in epithelial tissue. There is no fixed "scaffold" that determines the structure achieved during tissue morphogenesis. The organization into complex tissues and organs is an emergent property of the constituent cells of an organism and of the genes that determine the behavior of those cells. The Epitheliome model addresses explicitly the concept of structure as an emergent property of the interaction or "social behavior" of a large number (10^6–10^7) of individual cells. All software development has been carried out using object-oriented code in Mathworks MATLAB (22). The model adopts rule-based modeling to describe cell cycle progression and the modeled cell types are stem cells (which can undergo growth and division, bonding, spreading, lateral migration, and apoptosis), transit amplifying cells, mitotic cells, post-mitotic cells, and dead cells.

ABM has been then generalized to model intracellular chemical interactions (23). It is essential that an ABM is able to deal with individual interactions of molecule agents with the same accuracy as reaction kinetics. The authors argue that any reasonably random movement within an agent's confines is sufficient for the model to operate properly. Agents must at least move around enough to regularly collide. The model must of course agree with the corresponding reaction kinetics model in the circumstance where reaction kinetics can reasonably be applied (i.e., with large numbers of molecules of well-mixed chemicals).

The review from Walker and Southgate (24) examines individual-based models (cellular automata or agent-based methodologies) of cellular systems to explore multi-scale phenomena in biology. Such models, where individual cells are represented as equivalent virtual entities governed by simple rules, are inherently extensible and can be integrated with other modeling modalities (e.g., partial or ordinary differential equations) to model multi-scale phenomena. Alternatively, hierarchical agent models may be used to explore the functions of biological systems across temporal and spatial scales.

Recently, Adra et al. (25) developed a 3D multi-scale computational model of the human epidermis which is composed of three interacting and integrated layers: (1) an ABM which captures the biological rules governing the cells in the human epidermis at the cellular level and includes the rules for injury induced emergent behaviors, (2) a COmplex PAthway SImulator (COPASI) (26) ODE model which simulates the expression and signaling of the transforming growth factor (TGF-β1) at the subcellular level, and (3) a mechanical layer embodied by a numerical physical solver responsible for resolving the forces exerted between cells at the multicellular level.

Other recently published ABM of cellular systems concern parasitology (evolution of Chagas disease (27)) and immunology (competition between lung metastases and the immune system (28)). On the field of immunology too, an interesting recent review on ABM related to host–pathogen interaction and disease dynamics is that from Bauer et al. (29). The authors emphasize the well-known feature of generating surprisingly complex and emergent behavior from very simple rules, including periodic behaviors or intricate spatial and temporal patterns. Nonlinearities and time-delays are not difficult to treat empirically since they can be incorporated into the agent's rules or they may even emerge naturally as a consequence of the system's collective dynamics. Another remarked advantage of ABM is that their computational structure is inherently parallel and therefore can be implemented on parallel computers very efficiently.

The recent advent of programming for graphical processing units (GPU) together with the introduction of multi-core CPU actually enabled an easier, cheaper, and resolute parallelization of cellular simulation algorithms. In particular, GPUs do not have to perform many of the generalized tasks that a CPU must perform and therefore they have become highly optimized to perform tightly coupled data-parallel processing with hundreds of independent processor units and specialized memory addressing (30). In the past few years GPUs have been already used for tasks such as sequence analysis and molecular dynamics. Software toolkits like CUDA and OpenCL have greatly eased the complexity of GPU programming. ABM has numerous implementation challenges on the GPU to handle dynamic agents and their interactions. The methodology article of Christley et al. presents a pedagogical approach to describing how methods for multicellular modeling are efficiently implemented on the GPU. Aspects like memory layout of data structures and functional decomposition are discussed. The authors deal also with various programmatic issues and provide a set of design guidelines for GPU programming that are instructive to avoid common pitfalls as well as to extract performance from the GPU architecture.

The review from Demattè and Prandi (31) takes stock of some recent efforts in exploiting the processing power of GPUs for the simulation of biological systems. General purpose scientific computing on graphics processing units (GPGPU) actually offers the computational power of a small computer cluster at a cost of a few hundred dollars. However, computing with a GPU requires the development of specific algorithms, since the programming paradigm substantially differs from traditional CPU-based computing.

Other computational infrastructures, like the Grid, developed for seamless sharing computational power and other resources like memory, data, and knowledge among virtual organizations, enable ambitious collaborative projects to take off. The ImmunoGrid project, for instance, aims at developing agent-based simulations of the human immune system at a natural scale (32). ImmunoGrid has the task of modeling one of the most challenging components of the virtual physiological human (VPH). The VPH (33) is currently developed through several initiatives that are expected to enable an integrative and analytical approach to the study of medicine and physiology and to drive the paradigm shift in health care. The key benefits that the VPH aims to deliver are a holistic approach to medicine, personalized care solutions, a reduced need for animal experiments, and a preventive approach to the treatment of disease.

Virtual tissues (VT) are clearly of paramount importance for toxicology. Predicting the risk of chemical-induced human injury is a major challenge in toxicology (34). The translational issues in elucidating the complex sequence of events from chemical-induced molecular changes to adverse tissue level outcomes, and estimating their risk in humans, provide a practical context for developing VT. They aim to predict histopathological outcomes from alterations of cellular phenotypes that are controlled by chemical-induced perturbations in molecular pathways. As already emerged from our excursus in the state of the art, the behaviors of thousands of heterogeneous cells in tissues can be naturally simulated with ABM. Further, Shah and Wambaugh state that to extrapolate toxicity across species, chemicals, and doses, VT require four main components: (a) organization of prior knowledge on physiologic events to define the mechanistic rules for agent behavior, (b) knowledge on key chemical-induced molecular effects, including activation of stress sensors and changes in molecular pathways that alter the cellular phenotype, (c) multiresolution quantitative and qualitative analysis of histologic data to characterize and measure chemical-, dose-, and time-dependent physiologic events, and (d) multi-scale, spatiotemporal simulation frameworks to effectively calibrate and evaluate VT using experimental data.

We close our introduction to ABM of cellular systems by referring the readers also to the recent review on Internet resources for ABM by Devillers et al. (35).

2. Materials

Nowadays, ABMs can be designed on dedicated simulation platform or coded with specialized programming tools and frameworks. Gilbert and Bankes depicted ABM in a symbiotic relationship with computing technology (36). Modeling with agents became feasible only with the advent of personal workstations. Following the technological development, the scale and sophistication of the software available for modelers have greatly increased. Very sophisticated modeling is now viable thanks to complex algorithms, toolkits, and libraries. The earliest models were developed on mainframe computers. From the early 1990s most models have been developed in conventional programming languages such as C++, JAVA, and SMALLTALK. But the disadvantages of using a general purpose language were evident: basic algorithms must be always re-implemented, graphics libraries could be hardly coupled with dynamic modeling, and the resulting code was easily accessible only to those familiar with the language and the compiler needed to run it. The first development was represented by the emergence of several standardized libraries easily includable in developed ABM programs. REPAST (37), originally only a set of JAVA libraries, allowed programmers to build simulation environments (e.g., regular lattices), create agents in social networks, collect data from simulations automatically, and build user interfaces easily. Its features and design owe a lot to SWARM (38), one of the first ABM libraries (36). These libraries required nevertheless a good working knowledge of the programming language that they are aimed at, usually JAVA.

In the meanwhile, "agent-oriented programming languages" began to establish. JADE (39), in development since at least 2001 and adopted by a wide international community, is a very good example. Being developed over JAVA, it is consequently portable on all the operating systems supporting JAVA. JADE includes both the libraries (i.e., the JAVA classes) required to develop application agents and the run-time environment that provides the basic services and that must be active on the device before agents can be executed. Each instance of the JADE run-time is called container (since it "contains" agents). The set of all containers is called platform and provides a homogeneous layer that hides to agents (and to application developers too) the complexity and the diversity of the underlying tires (hardware, operating systems, types of network, JAVA Virtual Machine) (40). In this way, it realizes a middleware, laying "in the middle" between application software that may be working on different operating systems on different computers. Purpose of a middleware is interoperability, i.e., a set of services is provided, allowing multiple processes (e.g., agents) running on one or more machines to interact. In MAS, interoperability should also be granted between heterogeneous agent-based systems. The issue of agent systems standardization has been tackled by FIPA (41). FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. Originally formed in 1996 as a Swiss based not-for-profit organization to produce software standards specifications for heterogeneous and interacting agents and agent-based systems, in 2005 it was officially accepted by the IEEE (42) as its 11th standards committee.

JADE is compliant with the FIPA specifications. As a consequence, JADE agents can interoperate with other agents, provided that they comply with the same standard. Other FIPA-compliant agent systems includes Java Intelligent Agent Compontentware (JIAC) (43), Smart Python multi-Agent Development Environment (SPADE) (44), which is written in the Python programming language, and JACK Intelligent Agents (45). JACK is a mature, cross-platform environment for building, running, and integrating commercial-grade MASs. It is built on the sound BDI (Beliefs/Desires/Intentions) logical foundation. BDI is an intuitive and powerful abstraction that allows developers to manage the complexity of the problem. In JACK, agents are defined in terms of their beliefs (what they know and what they know how to do), their desires (what goals they would like to achieve), and their intentions (the goals they are currently committed to achieving).

Returning to JADE, we can observe that from the functional point of view it provides the basic services necessary to distributed peer-to peer applications in the fixed and mobile environment (40). Each agent can dynamically discover other agents and communicate with them according to the peer-to-peer paradigm. From the application point of view, each agent is identified by a unique name and provides a set of services. It can register and modify its services and/or search for agents providing given services, it can control its life cycle and, in particular, communicate with all other peers. Agents communicate by exchanging asynchronous messages, a communication model almost universally accepted for distributed and loosely Internet Wireless environment coupled communications, i.e., between heterogeneous entities that do not know anything about each other. In order to communicate, an agent just sends a message to a destination. Agents are identified by a name (no need for the destination object reference to send a message) and, as a consequence, there is no temporal dependency between communicating agents. The sender and the receiver could not be available at the same time. The receiver may not even exist (or not yet exist) or could not be directly known by the sender that can specify a property (e.g., "all agents interested in football") as a destination. Because agents identifies each other by their name, hot change of their object reference are transparent to applications.

The platform includes a naming service (ensuring each agent has a unique name) and a yellow pages service that can be distributed across multiple hosts. Another very important feature consists in the availability of a rich suite of graphical tools supporting both the debugging and management/monitoring phases of application life cycle. By means of these tools, it is possible to remotely control agents, even if already deployed and running: agent conversations can be emulated, exchanged messages can be sniffed, tasks can be monitored, agent life cycle can be controlled.

Gilbert and Bankes recognize also that, similarly to what happened with statistical computing, the real breakthrough in ABM was the development of "packages," collections of routines assembled with a common standardized user interface (36). Different from "agent-oriented programming languages" these packages or platforms do not require programming skills and permit direct manipulation or "visual programming" of the models. They provide "total immersion" in an environment in which building blocks can be assembled. The recently developed platforms for ABM offer also facilities for other phases of a model's life cycle, model evaluation, and model maintenance.

A list of ABM software, including simulation platforms and programming systems, highlighting their primary domain of application, the programming language (if any), the compliance with FIPA, and other capabilities (e.g., GIS, 3D) is constantly kept updated in Wikipedia (46).

Even very basic and widely used software tools can support ABM. For instance in ref. 47 it is shown how to realize a simulation of a shopper agent model using a spreadsheet like Microsoft Excel. In the last years a lot of specialized agent-based platforms have been developed. Their use, after an initial effort for learning the particular concepts and characteristics of the platform, can dramatically speed up not only the process of defining and running the simulations but also the ways in which the results are handled and analyzed, by connecting with largely used data analysis and visualization software tools and/or data formats. Another important aspect to take into consideration is the scalability of the software. The number of agents that have to be simulated for obtaining significant results can be taken as a rough measure of the needed computational power. While this number is under the order of 10^2 or 10^3 the simulation can usually be carried out on a desktop machine. If the number is higher there is the need of more powerful computing architectures for parallel and/or distributed computation. Some platforms give native support for these architectures (48). Recently, also GPUs with nVIDIA CUDA programming environment (49) are starting to be exploited by some platforms (50). In the latter case the high performance computing power can be obtained with lower costs with respect to classical parallel architectures.

Nicolai and Madey (51) give a survey of tools, classifying them using different characteristics: the programming languages on which the tool is based, the type of license, the operating systems for which the tool is available, the domain (including distributed simulation) for which the platform is specialized (or if it is general purpose), and the type of support offered to the final user. Other useful reviews, organized with different views, can be found in refs. 52 and 53.We now briefly describe those tools that historically have been mostly used and discussed. For a complete list we refer to the surveys and Web pages given above.

Swarm (38), developed by the Swarm Development Group, was one of the first general purpose ABM systems. It uses the Objective-C programming language, but by using a middleware it is possible to use also Java. A lot of documentation and tutorial material is available, and the platform can be used on top of the most diffused operating systems.

The Recursive Porous Agent Simulation Toolkit (Repast) (37) was derived from Swarm, but incorporate a lot of facilities to interact with other software for databases, data analysis, scientific calculus, and data visualization. It also supports distributed computation, and its specification of agents is based on Java, Groovy (54), and flowchart diagrams.

The Multi-Agent Simulator Of Neighborhoods... or Networks... or something... (Mason) (55) is a fully Java programmable library for agent-based simulations that can be integrated in larger applications. It can be easily connected with (possibly 2D or 3D) visualization components.

Graphical and accessible programs for creating ABMs are AgentSheets (56), NetLogo (57), and SeSAm (58).

NetLogo was originally designed for simple educational purposes but now it is largely used for research as well. Many colleges have used this as a tool to teach their students about ABM. A similar program, StarLogo (59), has also been released with similar functionality.

The SeSAm simulator with graphical modeling interface, optimizing model compiler and plug-in system, is used in research projects and educational environments. Many plug-ins are available (Evolution, Event-Based Simulation, Communication, GIS, Graph, FIPA, Import/Export, etc.), and the availability of the Java source code and plug-in infrastructure allows for further customization.

AnyLogic (60) is a commercial ABM tool. In AnyLogic the user can combine ABM with discrete-event (process-centric) modeling and system dynamics. Visual languages such as state charts, action charts, process flowcharts, and Stock and flow diagrams are used to define the behavior of agents.

Concerning ABM platform specifically addressed to the biological domain, we mention SimBioSys (61), based on C++ language that allows programming evolutionary simulations.

3. Methods

Being ultimately a software product, the design and implementation of a MAS requires software engineering methodologies to guide the whole production process. The use of a methodology helps in correctly and effectively realizing the system of interest in order to use it for the intended objectives. A methodology should give guidelines for all the typical phases of the software life cycle: requirement analysis, design, development, testing, and/or validation, deployment, and maintenance. Modern software engineering techniques are mainly focused in software methodologies for the object-oriented paradigm. Standard formal and semi-formal graphical specification languages exist for this paradigm (e.g., UML (62)) and several standardized software engineering processes have been defined (e.g., Rational Unified Process (63)).

Concerning agent-based systems, agent-oriented software engineering (AOSE) (64) and agent-based modeling and simulation (ABMS) (65) are recognized as very interesting emerging paradigms that will have a major impact on the quality of science and society over the next years. On the one hand, AOSE is concerned in finding the best way to design and implement an agent-based system, in order to create a MAS application that is able to manage and/or resolve a complex problem. On the other hand, ABMS has been defined as "a third way of doing science" in addition to traditional deductive and inductive reasoning (66). The novelty is that the scientist can face the understanding of the complexity of natural phenomena using the notion of agent to represent simple components of the real world and program them making hypotheses on their (simple) behaviors. Then, putting all together and running the simulation, the global system can be observed and, if the hypotheses were correct, a (known or unknown) emergent behavior should appear, that is more than the sum of the simpler individual behaviors of the components.

In the last years there has been an increasing number of initiatives to develop methodologies for the development of agent-based software systems. We mention GAIA (67, 68), Tropos (69), Prometheus (70), INGENIAS (71), PASSI (72), ADELFE (73), PASSIM (74). Each of these methodologies defines a different meta-model, that is to say, a set of general concepts, and relations among them, that are considered appropriate to model a MAS. There are several reviews and comparisons between these different methodologies, see, for instance, refs. 64 and 75. Unfortunately, only a few of the methodologies are "complete," in the sense that they cover all the phases of the life cycle of an agent-based software. All methodologies start from the requirements engineering phase and most of them stop after the design phase, when a detailed model of the system is available. Some of them continue towards the phase of implementation, deployment, and testing, but none of them goes beyond, to the maintenance phase. An attempt to unify all the existing different methodologies under a general meta-model has been done by FIPA (41), with the objective of defining a standard AUML (Agent UML) language (76). However, as the AUML Web site admits, the process has stopped at the moment because the new versions of UML seem to already incorporate notations that could be suitable for designing agent-based systems. Recently, a unified graphical notation for AOSE (77) has been proposed.

Let us briefly describe the meta-models of some of the methodologies.

In GAIA, the basic building blocks of a MAS are: agents, roles, activities, and protocols. An agent can have several roles, and for each role it exhibits a different behavior. Roles are defined in terms of permissions, responsibilities, and activities (i.e., the procedural abilities of the agents) as well as of interactions with other roles. Services are given by the agents when playing the associated roles and protocols are thought as general stubs for permitting communications between agents. Moreover, the meta-model contains social aspects of the MAS that are useful to model open agent systems. Agents constitute organizations, aggregated into an organizational structure, and both agents and roles observe organizational rules.

Tropos is an AOSE methodology that is based on three key ideas: a notion of agent with cognitive skills (goal, plan, and belief) used along all the software development phases, an approach called "requirement driven" (a sort of goal-oriented analysis) that is used along all the phases as well and the construction of a conceptual model in subsequent steps of refinement until the level of code is reached. In Tropos the elements of the basic ontology are the following: actors, goals, planes, resources, dependencies, capabilities, and beliefs. The actor is a strategic entity that exhibits a will. Analyzing these actors and their dependencies with respect to other actors, by using techniques such as means-end analysis, and/or decomposition, it is possible to specify the agents and the relative capabilities in order to model the system and solve the problem.

ADELFE is mainly concerned to the development of Adaptive Multi-Agent Systems. Much account is given to cooperation. Agents are defined with a limited (local) view of the environment and they have beliefs on the environment, on the other agents, and on themselves, on which they rely to decide their behaviors. Every agent can update its beliefs and can share them with other agents. The ADELFE agent is essentially cooperative, it always tries to solve a local goal and keep cooperative relationships with others. Its basic cycle is of the type perceive-decide-act. When a Non-Cooperative-Situation is detected by an agent (for instance, it received a message it could not understand) it tries to solve it and to stay cooperative with the others (for instance, it sends the incomprehensible message to other agents that it believes could understand it).

INGENIAS is the result of trying to integrate the best parts of other methodologies into one and it is also a set of tools to develop MAS (i.e., INGENIAS Development Kit (78)). The meta-model is composed of the following concepts:

- Organization, an autonomous entity that has its own goal. It can be structured in groups and contain workflows (procedural information to organize tasks, resources, and the participants to the procedure). The groups can be made of roles, agents, resources, or applications.

- Agent, that is, again, an autonomous entity. It can play different roles and pursue different goals. The agent has a mental state made of mental entities (goals, facts, beliefs) that is managed by a mental state manager (to create, modify, delete mental states) and by a mental state processor (to determine how mental states evolve and to select the action that the agent should execute at any moment).
- Interactions that can be initiated by an agent and that may involve more than two agents.
- Tasks and goals that are related. Every task is assigned an input and an output and it is specified how they affect the environment or an agent's mental state.
- Environment defines what is the perception of every agent and also identifies the resources of the systems and who is responsible to manage them.

PASSI (Process for Agent Societies Specification and Implementation) is an iterative and incremental process for developing a MAS. It also supports implementation and testing with the development toolkit PTK, the Passi Toolkit (79). The PASSI metamodel is organized in three different domains:

- Problem Domain. It encompasses scenarios, requirements, ontology, and resources. A scenario is a sequence of interactions that may happen among actors and the system. Requirements are usually described using a UML use case diagram. The ontology defines a set of concepts (categories of the domain), actions (executable in the domain and that can affect the status of concepts), and predicates (relating some domain elements). The resources are those that can be accessed by the agents.
- Agency Domain. The Agent is defined in this domain. Every agent is responsible of satisfying a set of requirements coming from the Problem Domain, using its capabilities. Any agent can play different roles along its life. These roles are parts of an agent's behavior that, depending on the specific role, can be goal driven or can consist of services that the agent gives, possibly accessing to available resources. There is a service component that represents the service provided by a role in terms of functionalities, associated to pre- and post-conditions and possibly other details to be called properly by other agents that may require it for their goals. Agents can use tasks or communication to realize the aims of a role. A task is a portion of behavior that can be considered atomic. Communication consists on agents exchanging one or more messages, each of them expressed according to an Agent Interaction Protocol that is used to give a certain level of predefined semantics to the message content.

- The Solution Domain is where the implemented systems will be deployed. It describes the code structure in the FIPA-compliant implementation structure that has been chosen for the implementation.

Several academic and industrial experiences have already shown that the use of MAS offers advantages in many different areas such as manufacturing processes, e-Commerce, and network management (80). Since MAS in such contexts need to be tested before their deployment and execution in real operating environment, methodologies that support system validation through simulation (e.g., discrete-event simulation, agent-based simulation, etc.) are highly required. Verification involves debugging the model to ensure it works correctly. Validation ensures that you have built the right model. In fact, simulation of MAS cannot only demonstrate that MAS correctly behaves according to its specifications but can also support the analysis of emergent properties of the MAS under test (74).

PASSIM makes a step towards this direction. PASSIM is an agent-oriented software development methodology that uses simulation in two different moments: at an early stage to prototyping the MAS being developed and at the late stage for validating the requirements of the developed MAS. It is based on parts coming from PASSI and from the distilled state charts (DSC)-based simulation methodology (81). The life cycle proposed by PASSIM is iterative and incremental and is based on the following steps: requirement specification, design, simulation, coding, and deployment. After the simulation step the designers can either proceed to the real implementation or use the results of the simulation as a feedback on the design and requirement specification phases. All the phases but simulation are supported by PTK, the PASSI Toolkit (79). For the simulation phase the DSC Visual Toolset (82) can be used.

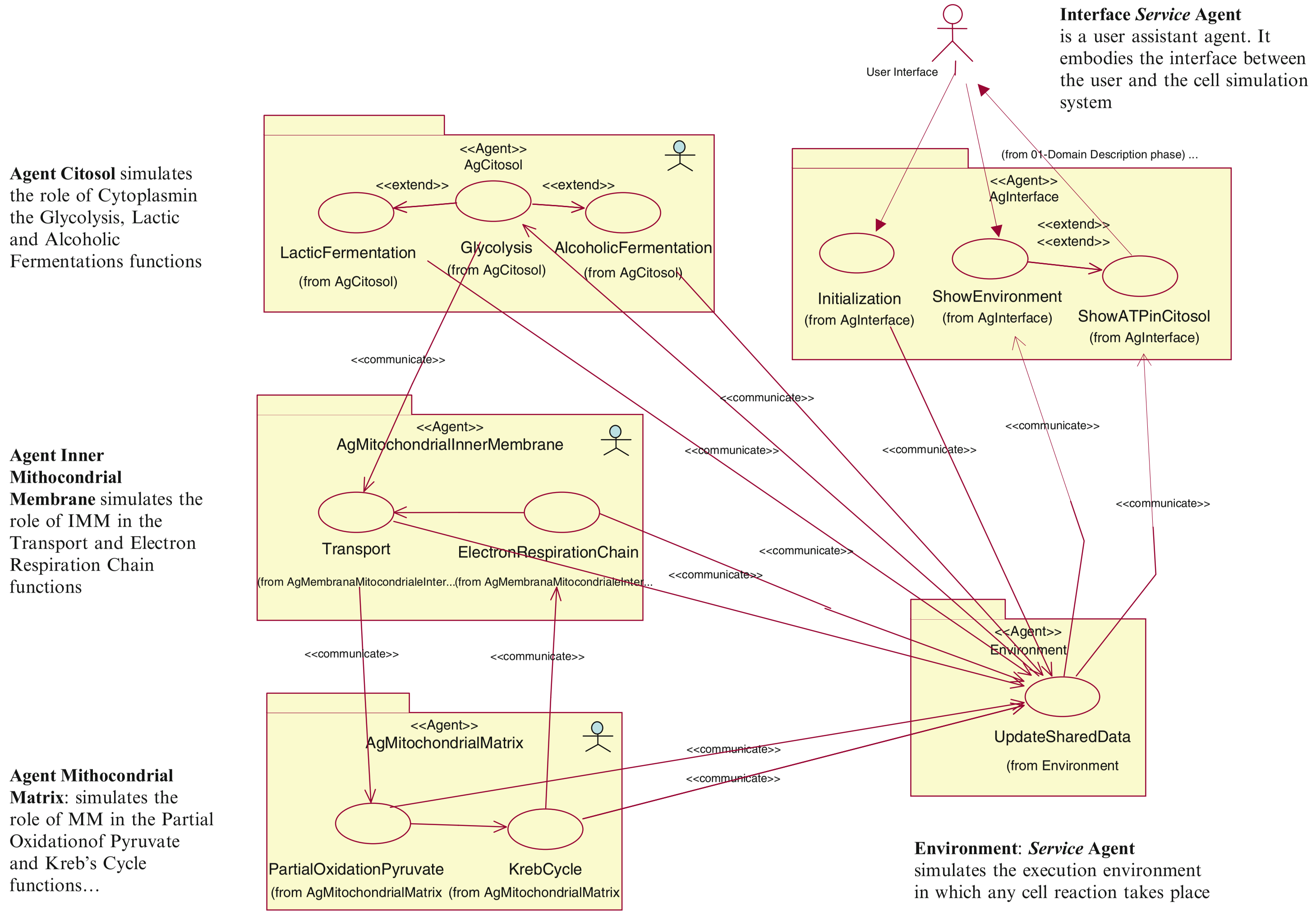
4. Examples

Here we report on the lessons we have learnt in programming ABMs of cellular systems. We find it very educational and at the same time we realize that our experience parallels very well the evolution of this research area.

We proposed a conceptual framework for engineering an agent society to simulate the behavior of a biological system (83). The framework is intended to support life scientists in building models and verifying experimental hypotheses by simulation. We believe that the use of an agent-based computational platform and of agent coordination infrastructures—along with the adoption of formal methods—will make it possible to harness the complexity of the biological domain by delegating software agents to simulate bio-entities. Relevant issues are those of information management, as well as the phases of model construction, analysis, and validation (84). The proposed conceptual framework takes into account the four steps suggested by Kitano (6): (1) system structure identification, (2) system behavior analysis, (3) system control, (4) system design. For each step, our framework exploits agent-oriented metaphors, models, and infrastructures to provide systems biologists with the suitable methodologies and technologies.

As a recurrent benchmark in our "experiments" with ABM we have chosen the very well-known and studied process of carbohydrate oxidation (CO). Our first focus was mainly on the model engineering issues. In refs. 85 and 86 we adopted the PASSI AOSE methodology (72). A simulator was designed with the help of the PASSI Toolkit (79) and implemented on the Hermes agent middleware platform (87). The PASSI Toolkit provides the system's specification by UML diagrams, very helpful for the implementation (e.g., for the automatic generation of stubs of code) and for documentation purposes, concerning different design level. PASSI, proceeding in a top-down way, naturally leads to the identification of the structure and the behavior of the cell and its components. Following a macrodescription of CO, we first identified the main functions and then each cell compartment (or subcompartment) involved into the process. Any identified compartment was modeled as an active autonomous entity (and each function as its specific role. The resulting multi-agent model consisted of the three agents: Cytoplasm, Inner Mitochondrial Membrane, and Mitochondrial Matrix, as depicted in the agent identification diagram shown in Fig. 1. Two other auxiliary agents were introduced to support the interface with the user (Interface Service Agent) and to simulate the execution environment in which all the reactions take place (Environment Service Agent). The environment represents a centralized coordinator, keeping track of the available quantities of molecules. The diagram shows also the roles played by each agent in the identified functions.

Software engineering distinguishes between verification—"did we build the system right?"—and validation—"did we build the right system?" (88). Merelli and Young considered the problem of model validation for simulation models whose structure as well as behavior mimics the modeled biological systems (89). Intentional insertion of faults is a well-known software testing technique ("mutation analysis"). The authors proposed to bring some modifications on the model of CO in order to mimic some known or plausible mutations in the subject system. When the modified model is executed, then a behavior is expected corresponding to that of the natural system with the same mutation. If the modification does not correspond to a known natural mutation we expect a



Interface *Service* Agent is a user assistant agent. It embodies the interface between the user and the cell simulation system

Agent Citosol simulates the role of Cytoplasmin the Glycolysis, Lactic and Alcoholic Fermentations functions

Agent Inner Mithocondrial Membrane simulates the role of IMM in the Transport and Electron Respiration Chain functions

Agent Mithocondrial Matrix: simulates the role of MM in the Partial Oxidationof Pyruvate and Kreb's Cycle functions…

Environment: ***Service* Agent** simulates the execution environment in which any cell reaction takes place

Fig. 1. The PASSI agent identification UML diagram for our CO example.

biologically plausible change in the behavior. As an example it was considered a mutation of the gene that produces one of the enzymes involved in the partial oxidation of pyruvate, thus allowing its passage from the cytoplasm to the mitochondrion. The previously developed model was not designed with such modification in mind. It did not permit to zoom into the cellular compartments and to "see" the enzymes at work. Therefore, it was not possible, for instance, to disable a specific enzyme. On the same model the mutation analysis technique exposed other weaknesses in the structural and behavioral correspondence with the modeled system. This analysis leads to a re-implementation of the model to permit a finer grain level of detail, so zooming into the cytoplasm. The new implementation was reorganized to address the structural correspondence, making it easy to model a class of mutations that delete or suppress the activity of particular enzymes. We experimentally applied a change corresponding to a pyruvate kinase deficiency and observed resulting changes in the processing of glucose and glycogen, which could be assessed for plausibility by biologists.

The previously described ABM of CO (87) has been built with a top-down methodology, analyzing the cell at a high level of abstraction and considering CO as a function at this level. In this vision we correctly identified the cellular compartments responsible for it and therefore modeled the cytoplasm and the mitochondria as interacting agents. Such a model does not permit to zoom into a deeper level of detail, thus having the possibility to observe how the cellular behavior emerges from the behavior of the molecular species hidden at the higher level. The agent society notion can be used here for defining an ensemble of cellular agents and the coordination artifacts involved in the cellular task characterizing the cellular agent society. The notion of agent society can be suitably adopted also for scaling with complexity, identifying different levels of descriptions of the same system. What can be described at one level as an individual agent, can be described as a society of agents (zooming in) at a more detailed level—as an ensemble of agents plus their mediating artifacts—and vice versa (zooming out). Consequently, the model of CO was refined (90), zooming into the cytoplasm and taking into account the pool of enzymes. In the implementation, a basic class enzyme was defined to represent a reactive entity. Then the enzyme was specialized for any of the enzymes involved in the CO and acting in that compartment. In particular, for each enzyme subclass the set of affinity to some metabolites was introduced. In this way, glycolysis can be seen as a function performed by a society of interacting cellular agents—the enzymes, whereas the metabolites are considered product of the environment.

The next step was to provide physical characteristics (shape, weight, size, position) to agents (both enzymes and metabolites), to place them in the space and to allow them to autonomously move and perceive the spatial neighborhoods, reacting accordingly. We modeled the cellular spatial environment as a finite continuous space filled by a hierarchy of entities ("bioagents") of different volumes. Enzymes and complexes are much bigger than metabolites and therefore they move much more slowly, but they have adaptive and reactive capabilities to recognize, in the dynamic environment of the cytoplasm, their metabolic counterparts in biochemical reactions. According to kinetic laws and metabolic maps they will be able to transform substrate bioagents into the corresponding product of the executable reaction. To have an idea of the numbers of agents involved, we estimated that in a small portion (10^{-15} L) of the cytoplasm, we need to simulate the movement and the interactions of several millions of autonomous entities. Reasoning at a suitable level of details—at the mesoscale as suggested by Wishart et al. (9)—we modeled every biomolecule involved in a metabolic process as an agent and any biological compartment as a coordination environment where molecular bioagents are situated, move and react. We developed a prototype MAS to test the practicability of our approach focusing on glycolysis (91). Cytoplasm is characterized by a three-dimensional occupancy and a number of physical properties like temperature, pH, fluid viscosity, and concentration—which we suppose uniform—of ions. The cytoplasm keeps track of the position of all the molecular bioagents wandering inside it. In our model, the cytoplasm is intended as a coordination environment and all the enzymes, metabolites, and their intermediate complexes are represented as moving agents. All the moving agents are characterized by a shape, which at the mesoscale can be considered spherical, and by a molecular weight, which can be derived from biochemical databases (e.g., Kegg LIGAND (92)). The weight of the intermediate complexes is intuitively calculated adding the weights of the molecular components forming the complex. The radius of the spheres can be easily calculated from their molecular weight. All the molecular bioagents (Fig. 2) have a three-coordinate position in the cytoplasmic space and move according to the Brownian motion, which is the predominant law at this scale. From Stokes and Smoluchowski laws and Einstein equation on Brownian motion we can assume that the diffusion of spherical particles in a viscous fluid is depending on their radius and on the temperature and viscosity of the fluid. The latter takes also into account the local concentration of molecules around the moving agent. We divided the molecular bioagents into active and passive ones. The first (i.e., enzymes and complexes of enzymes and metabolites), besides the capability to move and perceive their environment, can also perform biochemical reactions. The latter (i.e., metabolites) have only the capability to move and to be manipulated—like in the reality—by active bioagents. The metabolites substrates (i.e., "inputs") of an enzymatic reaction undergo to specific chemical transformations, which transform them into

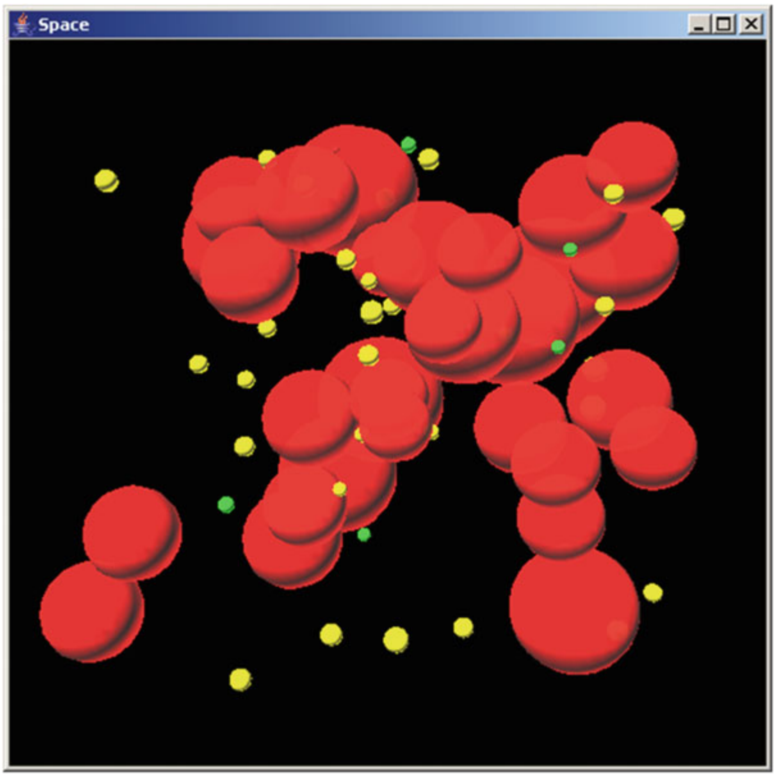


Fig. 2. A snapshot of our metabolic reaction simulation, showing enzyme, metabolite, and complex agents moving and acting in a portion of cytoplasm.

different chemical entities, products (i.e., "outputs") of the reaction. The enzyme instead remains unaltered by the completed reaction and can therefore try to "capture" new instances of the substrates into its active site, thus starting a new reaction. The net result of a completed reaction will be the disappearing of the instances of substrate bioagents previously captured by the enzyme and the appearing of new instances of the product bioagents, exactly as described by the chemical formula.

5. Notes

We have given a large view of the existing methodologies and tools to realize an agent-based system. It might be difficult for a researcher, possibly not coming from a computer science field or a related one, to start facing the task of realizing a MAS for his/her purposes. For this reason we suggest a useful, in our opinion, starting place. Inside the wiki pages of Swarm, there is a page presenting software templates (93). In this page, the so-called StupidModel, templates are available for using in Swarm, MASON, Repast, and NetLogo. They tackle 16 typical situations of agent-based programming, each of which is equipped with sample code realizing the relative function in the four tools. Situations range from creating a grid world in which agents can move to making agents born or die and obtaining statistic information about the MAS evolution as charts or output files. The page contains also other material more specific to Swarm and to other tools. We think that these templates are a good way to start familiarizing with the tools and the concepts of ABMS. The code can be copied, made run as it is in the platforms, and then modified and extended according to one's particular needs.

References

1. Milner R (1989) Communication and concurrency. Prentice-Hall, New York, London
2. Wooldridge M (1997) Agent-based software engineering. IEE proceedings on software engineering: 26–37
3. Wooldridge M (2002) An introduction to MultiAgent systems. Wiley, West Sussex, UK
4. Vallurupalli V, Purdy C (2007) Agent-based modeling and simulation of biomolecular reactions. Scalable Computing: Practice and Experience 8(2):185–196
5. Bonabeau E (2002) Agent-based modeling: methods and techniques for simulating human systems. Proc Natl Acad Sci USA 99 (Suppl 3):7280–7287
6. Kitano H (2001) Foundations of systems biology. MIT Press, Cambridge, MA
7. González PP, Cárdenas M, Camacho D et al (2003) Cellulat: an agent-based intracellular signalling model. Biosystems 68 (2–3):171–185
8. Wolfram S (1984) Cellular automata as models of complexity. Nature 311:419–424
9. Wishart DS, Yang R, Arndt D et al (2005) Dynamic cellular automata: an alternative approach to cellular simulation. Silico Biol 5 (2):139–161
10. Troisi A, Wong V, Ratner MA (2005) An agent-based approach for modeling molecular self-organization. Proc Natl Acad Sci U S A 102(2):255–260
11. Emonet T, Macal CM, North MJ et al (2005) AgentCell: a digital single-cell assay for bacterial chemotaxis. Bioinformatics 21 (11):2714–2721
12. Korobkova E, Emonet T, Vilar JM et al (2004) From molecular noise to behavioural variability in a single bacterium. Nature 428 (6982):574–578
13. Lemerle C, Di Ventura B, Serrano L (2005) Space as the final frontier in stochastic simulations of biological systems. FEBS Lett 579 (8):1789–1794
14. Lloyd CM, Halstead MD, Nielsen PF (2004) CellML: its future, present and past. Prog Biophys Mol Biol 85(2–3):433–450
15. Hucka M, Finney A, Sauro HM et al (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19(4):524–531
16. Takahashi K, Arjunan SN, Tomita M (2005) Space in systems biology of signaling pathways–towards intracellular molecular crowding in silico. FEBS Lett 579 (8):1783–1788
17. Tang J, Ley KF, Hunt CA (2007) Dynamics of in silico leukocyte rolling, activation, and adhesion. BMC Syst Biol 1:14
18. The breve simulation environment. http://www.spiderland.org/. Accessed 19 Nov 2010
19. Guo Z, Sloot PM, Tay JC (2008) A hybrid agent-based approach for modeling microbiological systems. J Theor Biol 255(2):163–175
20. Guo Z, Tay JC (2008) Multi-timescale event-scheduling in multi-agent immune simulation models. Biosystems 91(1):126–145
21. Walker DC, Southgate J, Hill G et al (2004) The epitheliome: agent-based modelling of the social behaviour of cells. Biosystems 76 (1–3):89–100
22. MATLAB—The language of technical computing. http://www.mathworks.com/products/matlab/. Accessed 19 Nov 2010
23. Pogson M, Smallwood R, Qwarnstrom E et al (2006) Formal agent-based modelling of intracellular chemical interactions. Biosystems 85 (1):37–45

24. Walker DC, Southgate J (2009) The virtual cell–a candidate co-ordinator for 'middle-out' modelling of biological systems. Brief Bioinform 10(4):450–461
25. Adra S, Sun T, MacNeil S et al (2010) Development of a three dimensional multiscale computational model of the human epidermis. PLoS One 5(1):e8511
26. Hoops S, Sahle S, Gauges R et al (2006) COPASI–a COmplex PAthway SImulator. Bioinformatics 22(24):3067–3074
27. Galvão V, Miranda JG (2010) A three-dimensional multi-agent-based model for the evolution of Chagas' disease. Biosystems 100 (3):225–230
28. Pennisi M, Pappalardo F, Palladini A (2010) Modeling the competition between lung metastases and the immune system using agents. BMC Bioinformatics 11(Suppl 7):S13
29. Bauer AL, Beauchemin CA, Perelson AS (2009) Agent-based modeling of host-pathogen systems: the successes and challenges. Inf Sci (Ny) 179(10):1379–1389
30. Christley S, Lee B, Dai X et al (2010) Integrative multicellular biological modeling: a case study of 3D epidermal development using GPU algorithms. BMC Syst Biol 4:107
31. Dematté L, Prandi D (2010) GPU computing for systems biology. Brief Bioinform 11 (3):323–333
32. Halling-Brown M, Pappalardo F, Rapin N et al (2010) ImmunoGrid: towards agent-based simulations of the human immune system at a natural scale. Philos Transact A Math Phys Eng Sci 368(1920):2799–2815
33. Viceconti M, Clapworthy G, Van Sint JS (2008) The virtual physiological human—a European initiative for in silico human modeling. J Physiol Sci 58(7):441–446
34. Shah I, Wambaugh J (2010) Virtual tissues in toxicology. J Toxicol Environ Health B Crit Rev 13(2–4):314–328
35. Devillers J, Devillers H, Decourtye A (2010) Internet resources for agent-based modelling. SAR QSAR Environ Res 21(3–4):337–50
36. Gilbert N, Bankes S (2002) Platforms and methods for agent-based modeling. Proc Natl Acad Sci USA 99(Suppl 3):7197–7198
37. REPAST - Recursive porous agent simulation toolkit. http://repast.sourceforge.net/. Accessed 19 Nov 2010
38. A resource for agent- and individual-based modelers and the home of Swarm. http://www.swarm.org/index.php/Main_Page. Accessed 19 Nov 2010
39. JADE—Java Agent DEvelopment Framework. http://jade.tilab.com/. Accessed 19 Nov 2010
40. JADE—A White Paper. http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf. Accessed 19 Nov 2010
41. IEEE foundation for intelligent physical agents. http://www.fipa.org/. Accessed 19 Nov 2010
42. IEEE—the world's largest professional association for the advancement of technology. http://www.ieee.org/index.html. Accessed 19 Nov 2010
43. Java-based intelligent agent componentware. http://jiac.de/. Accessed 19 Nov 2010
44. SPADE—Smart python multi-agent development environment. http://code.google.com/p/spade2/. Accessed 19 Nov 2010
45. JACK autonomous software. http://aosgrp.com/products/jack/index.html. Accessed 19 Nov 2010
46. Comparison of agent-based modeling software. http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software. Accessed 19 Nov 2010
47. Macal CM, North MJ (2008) Agent-based modeling and simulation: desktop ABMS. In proc. of winter simulation conference 2007:95–106
48. Sánchez D, Isern D, Rodríguez-Rozas A et al (2010) Agent-based platform to support the execution of parallel tasks. Expert Syst Appl. doi:10.1016/j.eswa.2010.11.073
49. CUDA Zone—Official Website. http://www.nvidia.com/object/cuda_home_new.html. Accessed 19 Nov 2010
50. Allan R (2009) Survey of agent based modelling and simulation tools. Technical report. computational science and engineering department, STFC Daresbury laboratory, Daresbury, Warrington. http://epubs.cclrc.ac.uk/work-details?w=50398. Accessed 19 Nov 2010
51. Nikolai C, Madey G (2009) Tools of the trade: a survey of various agent based modeling platforms. J Artif Soc Soc Simulat 12(2):2
52. Railsback SF, Lytinen SL, Jackson SK (2006) Agent-based simulation platforms: review and development recommendations. Simulation 82:609–623
53. Tobias R, Hofmann C (2004) Evaluation of free Java-libraries for social-scientific agent based simulation. J Artif Soc Soc Simulat 7 (1):6
54. Groovy—An agile dynamic language for the Java platform. http://groovy.codehaus.org/. Accessed 19 Nov 2010

55. Mason multiagent simulation toolkit. http://cs.gmu.edu/~eclab/projects/mason/. Accessed 19 Nov 2010
56. AgentSheets. http://www.agentsheets.com/. Accessed 19 Nov 2010
57. NetLogo Home Page. http://ccl.northwestern.edu/netlogo/. Accessed 19 Nov 2010
58. SeSAm—Integrated environment for multi-agent simulation. http://www.simsesam.de/. Accessed 19 Nov 2010
59. StarLogo on the Web. http://education.mit.edu/starlogo/. Accessed 19 Nov 2010
60. Why AnyLogic simulation software? http://www.xjtek.com/anylogic/why_anylogic/. Accessed 19 Nov 2010
61. SimBioSys class framework. http://www.lucifer.com/~david/SimBioSys/. Accessed 19 Nov 2010
62. Unified modeling language, resource page. http://www.uml.org/. Accessed 19 Nov 2010
63. Rational unified process: best practices for software development teams whitepaper. http://www.augustana.ab.ca/~mohrj/courses/2000.winter/csc220/papers/rup_best_practices/rup_bestpractices.html. Accessed 19 Nov 2010
64. Bernon C, Cossentino M, Pavón J (2005) Agent-oriented software engineering. Knowl Eng Rev 20(2):99–116
65. Macal CM, North MJ (2008) Agent-based modeling and simulation: desktop ABMS. In proc. of winter simulation conference 2007, pp. 95–106
66. Axelrod R (1997) Advancing the art of simulation in social sciences. In: Conte R, Hegselmann R, Terna P (eds) Simulating social phenomena. Springer-Verlag, Berlin
67. Wooldridge M, Jennings NR, Kinny D (2000) The Gaia methodology for agent-oriented analysis and design. J Auton Agent Multi-Agent Syst 3(3):285–312
68. Zambonelli F, Jennings N, Kinny D (2003) Developing multiagent systems: the Gaia methodology. ACM Trans Softw Eng Methodol 12(3):417–470
69. Bresciani P, Giorgini P, Giunchiglia F et al (2004) Tropos: an agent-oriented software development methodology. J Auton Agent Multi-Agent Syst 8:203–236
70. Padgham L, Winikoff M (2002) Prometheus: a methodology for developing intelligent agents. In proc. of 3rd international conference on agent-oriented software engineering III:174–185
71. Pavon J, Gomez-Sanz J, Fuentes R (2005) The INGENIAS methodology and tools. In: Henderson-Sellers B, Giorgini P (eds) Agent-oriented methodologies. Idea Group, London
72. Cossentino M (2005) From requirements to code with the PASSI methodology. In: Henderson-Sellers B, Giorgini P (eds) Agent-oriented methodologies. Idea Group, London
73. Bernon C, Camps V, Gleizes M-P et al (2005) Engineering adaptive multi-agent systems: the ADELFE methodology. In: Henderson-Sellers B, Giorgini P (eds) Agent-oriented methodologies. Idea Group, London
74. Cossentino M, Fortino G, Garro A et al (2008) PASSIM: a simulation-based process for the development of multi-agent systems. Int J Agent-Oriented Softw Eng 2(2):132–170
75. Henderson-Sellers B, Giorgini P (2005) Agent-oriented methodologies. Idea Group, London
76. The FIPA agent UML Web site. http://www.auml.org/. Accessed 19 Nov 2010
77. Padgham L, Winikoff M, Deloach S, Cossentino M (2009) A unified graphical notation for AOSE. In proc. agent-oriented software engineering IX. doi:10.1007/978-3-642-01338-6_9
78. García-Magariño I, Gutiérrez C, Fuentes-Fernández R (2009) The INGENIAS development kit: a practical application for crisis-management. In bio-inspired systems: computational and ambient intelligence. Lect Notes Comput Sci 5517:537–544
79. PASSI toolkit. http://sourceforge.net/pro1jects/ptk/. Accessed 19 Nov 2010
80. Cossentino M, Fortino G, Gleizes MP et al (2010) Simulation-based design and evaluation of multi-agent systems. Modell Pract Theory 18(10):1425–1427
81. Fortino G, Garro A, Russo W (2005) An integrated approach for the development and validation of multi-agent systems. Comput Syst Sci Eng 20(4):94–107
82. Fortino G, Garro A, Mascillaro S et al (2007) ELDATool: a statechart-based tool for prototyping multi-agent systems. Proc. of workshop on objects and agents (WOA '07)
83. Cannata N, Corradini F, Merelli E et al (2005) An agent-oriented conceptual framework for systems biology. Trans Comput Syst Biol, Lect Notes Comput Sci (3737):105–122
84. Finkelstein A, Hetherington J, Li L et al (2004) Computational challenges of systems biology. Computer 37(5):26–33
85. Corradini F, Merelli E, Vita M (2005) A multi-agent system for modelling carbohydrate oxidation in cell. Lect Notes Comput Sci 3481:1264–1273
86. Cannata N, Corradini F, Merelli E (2008) Multiagent modelling and simulation of

carbohydrate oxidation in cell. Int J Modell Ident Control 3(1):17–28

87. Corradini F, Merelli E (2005) Hermes: agent-based middleware for mobile computing. Lect Notes Comput Sci 3465:234–270
88. Boehm B (1981) Software engineering economics. Prentice Hall, New York, London
89. Merelli E, Young M (2007) Validating MAS with mutation. Int J Multiagent Grid Syst 3 (2):225–243
90. Berluti E, Corradini F, Leli S et al (2006) Glycolisis and fermetation in cellular energy production. Internal report unicam-cs01-2006, University of Camerino, Department of Computer Science
91. Cannata N, Corradini F, Merelli E et al (2008) A spatial simulator for metabolic pathways. International workshop on multi agents systems and bioinformatics (MAS & BIO 2008) http://www.cs.unicam.it/tesei/updown/CCMT08b.pdf. Accessed 19 Nov 2010
92. KEGG LIGAND Database. http://www.genome.jp/ligand/. Accessed 19 Nov 2010
93. Software Templates – SwarmWiki. http://www.swarm.org/index.php/Software_templates. Accessed 19 Nov 2010