

# Validating MAS Models with Mutation<sup>\*</sup>

Emanuela Merelli<sup>1</sup> and Michal Young<sup>2</sup>

<sup>1</sup> Dipartimento di Matematica e Informatica,  
Università di Camerino  
Camerino, 62032, Italy,  
emanuela.merelli@unicam.it

<sup>2</sup> Department of Computer Science  
University of Oregon, Eugene, 97403, Oregon  
michal@cs.uoregon.edu

**Abstract.** A key reason for choosing agent-based simulation of a biological system over other kinds of simulation model is the potential for structural as well as behavioral correspondence between the simulation model and the modeled system. This correspondence both demands and makes possible new kinds of model validation. We propose evaluating model fidelity by introducing seeded faults that correspond to known or hypothesized mutations, thus tying software validation to biological mutation analysis. We illustrate with consideration of an agent-based model of carbohydrate oxidation in a cell.

## 1 Introduction

A biological system consists of several components interacting to perform complex functions [12]. This complexity of interaction is far too much to be fully comprehended by the unaided mind, and so computer simulation is an essential tool for testing and refining our understanding of structure and behavior. Of course, such models can also mislead, either through ordinary software faults (bugs) or through deeper mistakes in modeling. How should we test them?

We consider the problem of model validation for simulation models whose structure as well as behavior mimics the modeled biological system, down to some level of detail. A simulation model that is opaque or unstructured does not serve as a useful representation of theory for the bioscientist. A model that is as close as possible to the mental model of the bioscientist is easier to incrementally build, maintain, and refine, and provides a better check on understanding. One approach to building such models is as a collection of software agents. Agent technology [11] is an appealing approach to building such models because the model can consist of software components that correspond closely with the components and environment of the biological system, at whatever level of granularity is appropriate to understanding.

If one takes the approach of closely modelling structure, so that gross system behaviour and characteristics emerge from complex interactions, one then needs a way to

---

<sup>\*</sup> Work undertaken while the first author was on Fulbright leave at University of Oregon.

gain confidence in having built the “right” model. This involves not only finding simple software faults (“bugs” in the code), but also validating the overall design of the software and its fidelity to the model as understood by the bioscientist.

Of course, the fundamental approach to validating a simulation model is to test it on some scenarios for which we can distinguish correct (or at least plausible) behaviours from behaviours that reveal a flaw in design or coding. But what test scenarios can shed light on the structural and behavioural fidelity of a model? Our proposal is that, in addition to variations in environmental conditions, scenarios should include modifications of the system. In the case of biological models, these can be models of known and hypothetical mutations. If we can identify simple changes to the software system that correspond to known mutations, and if this change to the simulation model results in behavioural changes that correspond to what is known about the actual biological system, then we gain considerable confidence in model fidelity. The mutation approach is particularly suited to validation of agent-based models, but may also be useful for other modeling approaches in which a correspondence between model structure and biological system structure can be established.

In the remainder of this paper we begin with some background on agent-based modelling in systems biology, then describe a model validation approach based on mutation. The approach is illustrated with a report of our preliminary experience applying the mutation analysis approach to a model of carbohydrate oxidation in a cell [5].

## **2 Agent-based Simulation and Systems Biology**

Approaches to constructing computer simulation models vary widely depending on the characteristics of the modelled system and the intended uses of the model, and effective approaches to model validation depend in turn on the modeling approach. Systems biology is concerned with complex systems in which overall behaviour emerges from a myriad of interactions among entities in a dynamic environment. Multi-agent systems [18, 11], in which interactions among discrete software components (agents) correspond to interactions among entities in the modelled biological system, is an attractive approach in this domain [4].

An agent is an autonomous entity that encapsulates execution of independent activities within its environment. Autonomy is characterised by encapsulation of behaviour control. Like a software object in object-oriented software construction, an agent encapsulates state and behaviour. In contrast to conventional objects, in which operations are invoked only from outside, agents are pro-active and have full control of their behaviour, which proceeds concurrently with the activity of other agents in the environment.

As a situated entity, an agent is a persistent entity immersed within and interacting with an environment, which is typically open and dynamic. Interaction — broadly construed [17] — is a fundamental dimension of the agent paradigm. An agent interacts with its environment by means of actions and perceptions, which enable the agent to partially observe and control the environment. “Perception” in the sense used here need not imply intelligence or consciousness, but merely that an agent can react to relevant conditions. An enzyme that catalyzes a reaction can be said to “perceive” the presence of reagents in this sense, although clearly no conscious monitoring is involved.

The agent abstraction allows us to directly model both reactive and proactive behaviour of biological components and their environment. An environment abstraction can represent the resources found in the environment, and mediate interaction of other agents with these resources. The environment abstraction can be implemented as a blackboard, as proposed by Omicini et al. [15], or, as in the cell model we describe below, as another agent.

A multi-agent system can model a biological system at a desired level of abstraction by choosing component agent models at appropriate granularity. If agents represent individual components of the system, the overall MAS including environmental abstractions captures the overall set of the biological components and their environment, including the structures involved in their interaction.

Constructing such a model involves many design and implementation choices. It is important to have criteria that distinguish a “correct” from “incorrect” models, and highly desirable to also have objective criteria for judging the quality of design decisions (e.g., the division of the model implementation into agents). Some software faults are unambiguous: If a model “crashes” and cannot run to completion, then we can say with some confidence that it is not correct. A more complete definition of what it means for a model to be “right” or “good,” though, is more elusive. Validation of agent-based models of biological systems is largely unexplored.

Since a MAS simulation model is intended to mimic the simulated system in structure and behavior, and not only in overall output, one would like to validate not only predictions (e.g., by simulating well-understood scenarios), but also the relation between those predictions and the structure and function of the model. Just as sensitivity analysis is used to measure sensitivity of outputs to variance in input and assumptions, a kind of sensitivity analysis can be used to measure the thoroughness of test scenarios.

Multi-agent modelling of biological systems is at an early stage of development. Among recent work are several models of cell behavior. The Cellulat model proposed by Gonzalez et al. [8] fuses the behavioural-based paradigm and the blackboard architecture for intracellular signalling modelling. d’Inverno and Saunders [6] provide a multi-agent system to model and simulate stem cell behaviour, with a stem cell model formally defined in the Z specification language. Corradini et al. [5] describe a MAS to model carbohydrate oxidation in a cell. To date work in multi-agent modeling, including these models of cell behavior, has not addressed the problem of providing explicit evidence of fidelity of the model to the biological system, or the quality of the modeled system. That is our aim.

### 3 Mutation Analysis to Validate Agent-based Systems

In software engineering, a distinction is drawn between *verification*, which checks conformance between an implementation and some independent, formal description of accepted behavior, and *validation*, which is a broader but less clear-cut evaluation of actual fitness-for-purpose.<sup>3</sup> In the aphoristic formulation of Boehm [1], verification asks “did

---

<sup>3</sup> Note that “verification” is a much broader term than “formal verification,” or proof of correctness. Much of software testing, for example, is classified as “verification” rather than “validation.”

we build the system right,” and validation asks “did we build the right system” Both questions are relevant to MAS model; our focus here is primarily the latter.

Individual components of a MAS pose some of the same issues for testing as reactive system, some which can be addressed through adaptations of conventional specification-based or “black box” testing [10]. Additionally, some well-formedness properties of a multi-agent model may be addressed by considering it as a concurrent system and checking for well-understood properties of concurrent systems [13, 14]. These techniques address verification, and are complementary to validation, which is our focus here.

More interesting to us here is model *fidelity*, the question of whether the structure and function of the model accurately reflects the biological system. Given some particular scenario and expected outcome, we can execute the simulation to show that the model behavior is sufficiently close to the expected outcome (or to discover a discrepancy). This is a straightforward software testing exercise, not markedly different from other software testing except insofar as concurrency and non-determinism may add some complexity to control and observation. The new challenge comes in choosing the set of scenarios, because the question of model fidelity is ultimately not one of verification against a clear and precise specification but of validation against the intended purpose of the model.

Much of our understanding of biological systems has come from analysis of naturally and artificially induced deficits and modifications. For example, study of cognitive function in brain-injured individuals was crucial in associating brain structure with function, and induced mutation is a central tool in modern genetics. This leads naturally to the proposition that we can better understand a model — and in particular, compare the model to a natural system — by intentionally introducing faults and modifications. A simulation model that closely reflects the modeled system in both structure and function should be amenable to modifications that mimic natural injuries or mutations, and should show similar effects.

Intentional insertion of faults is a well-known software testing technique, part of an approach known (confusingly, in this context) as “mutation analysis” [9, 2]. The conventional fault-based testing approach is aimed at statistical measurement of the effectiveness of a test suite at exposing faults, analogous to a classical capture-recapture experiment [3]. The seeded faults in conventional software mutation analysis are simple syntactic modifications (for example, changing a comparison “<=” to a comparison “=”) that bear no relation to biological mutation. Since the aim of is purely verification rather than validation, examination of the resulting behaviors stops at distinguishing an incorrect behavior of the “mutant” program from correct behaviors of the unmodified program.

In our proposed approach, the “mutations” are not arbitrary syntactic variations in code. They are larger-grain modifications that mimic some known or plausible mutation in the subject system — which first of all imposes a requirement on the model that it possess a structure admitting of such modifications. When the modified model is executed, we expect behaviors that correspond to those of the natural system with a corresponding mutation, or if the modification does not correspond to a known natural mutation, we expect a biologically plausible change in behavior.

Two kinds of shortcomings may be exposed in a MAS model through this approach. First, we may find that there is no straightforward way to make a software change corresponding to the expression of the biological mutation. A software model necessarily omits some details and abstracts others, and attempts to make changes can illuminate those decisions of model detail. For example, a mutation might prevent a particular enzyme from being produced. Ideally, in a MAS model, the enzyme itself would be an agent, and preventing production of the enzyme would correspond to suppressing activation of the agent. Depending on the level of detail of the model, however, the enzyme may not exist as an identifiable structure in the model. Corresponding functionality might still be easily identified and disabled. Whether or not such an implicit representation of the enzyme through behavior were acceptable would be a judgment depending on the intended uses of the model over time.

A second kind of shortcoming in a model can be exposed through biologically-inspired mutation testing when behavior of the modified model does not correspond to behavior of a biological system with the corresponding change. This could be due to a simple software fault — a “bug,” in common parlance — or it could be due to deeper problems in the design of a model. It may expose, for example, parts of the model in which one component depends inappropriately on assumptions about the workings of another, or where some aspect of the environment that can affect behavior does not. For example, it might be the case that a certain chemical is always in abundant supply during normal functioning. Some behavior of the model may depend on that chemical, and yet it may never be tested. The omission of the test becomes obvious only when a modification introduces a scarcity of that chemical.

Even if neither of these shortcomings is exposed by several tests with several mutations, we can never definitively prove the fidelity of a model. We can nevertheless greatly improve our confidence in its robustness and in the likelihood that it can be further developed and refined incrementally, building up further details by making each software component a faithful representation of a corresponding biological component, rather than attempting to grasp all the possible interactions among parts of a monolithic model.

## **4 Validating a Carbohydrate Oxidation Model**

We are exploring and evaluating the mutation analysis approach to model validation by applying it to a model of oxidation in a cell [5]. In this section we briefly describe that model, and then our experience attempting to validate it.

A biological cell is an example of a complex system consisting of a large number of interacting components in a dynamic environment. Each cell is a self-contained and self-maintaining (autonomous) entity: it can take in nutrients, convert these nutrients into energy, and reproduce itself.

Cell metabolism is the process by which individual cells process nutrient molecules. The series of chemical reactions within a cell that realise cell metabolism are called metabolic pathways. Carbohydrate oxidation is a set of metabolic pathways by which a cell produces energy through chemical transformation of carbohydrates [7] like fructose, glucose, mannose, maltose, lactose, saccharose, glycogen and starch. Each kind of

carbohydrate has its metabolic pathway, i.e. a series of chemical reactions. Some occur only in presence of oxygen (aerobic respiration in mitochondria) and some in its absence (anaerobic metabolism, such as fermentation in cytoplasm). Since the CellMAS models the structure as well as the behavior of the biological system, the metabolic pathways are distributed among software agents that model the cellular components that collaborate in each pathway.

The carbohydrate oxidation process is performed by two main cell components: cytoplasm and mitochondria. Cytoplasm is the viscid, semi-fluid component between the cell membrane and the nuclear envelope, where the first stage of carbohydrate molecule transformation takes place. It consists primarily of water, but also contains various organelles (e.g. mitochondria) as well as salts, dissolved gasses and metabolites.

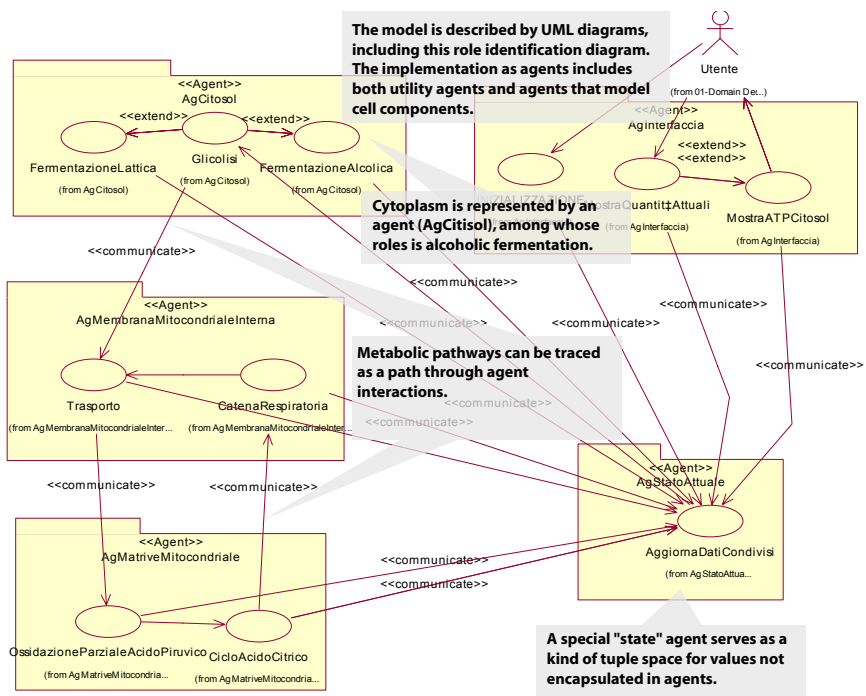
During the carbohydrate oxidation process, three metabolic pathways are active. Glycolysis converts one molecule of Glucose into two molecules of Pyruvate. Lactic Fermentation, in absence of oxygen, reduces the Pyruvate to Lactate. Lactic fermentation occurs in anaerobic microorganisms (sour milk) and animal cells, and is familiar to runner and other athletes from the muscle pain associated with build-up of lactate when exertion exceeds the level that can be supported by aerobic metabolism. Alcoholic fermentation, in the absence of oxygen, reduces pyruvate to ethanol. It occurs in yeast (*Saccharomyces*) causing the transformation of carbohydrates present in grapes and malted barley into ethanol.

We have chosen to validate the agent-based model produced by Corradini et al. [5]. Specification diagrams produced with that model describe the system's design from top-level architecture to detailed behavior, so we can distinguish between model design issues and simple faults in implementation coding. In principle the design diagrams should also facilitate changes to the model. The design consists of seven types of UML diagram: Domain Description, Agent Identification, Role Identification, Task Specification Ontology Description, Roles Description and Protocol Description Diagrams

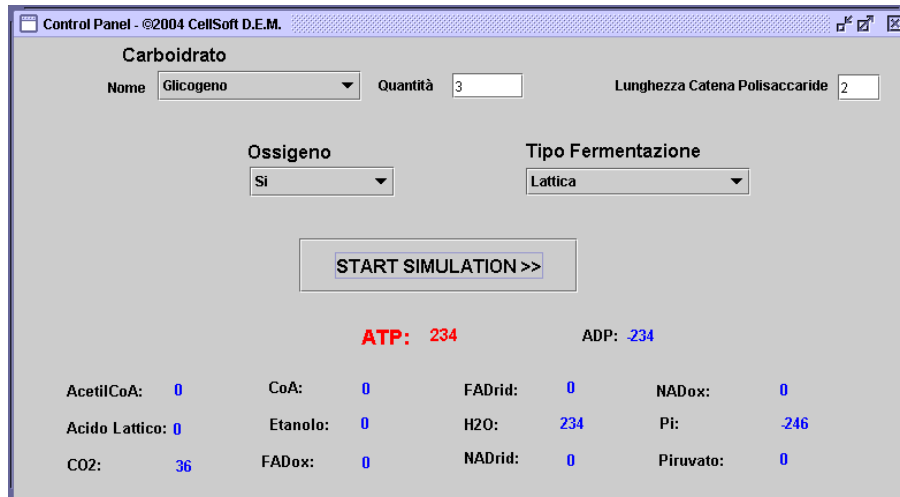
Components of the model include five agents corresponding to four cell components (cytoplasm, inner mitochondrion membrane, mitochondrion matrix and the environment) plus a user's assistant. Figure 1 shows the system's components represented as UML stereotyped packages.

The simulation model takes as input a carbohydrate fructose, glucose, mannose, maltose, lactose, saccharose, glycogen or starch, and simulates a cellular process, either aerobic respiration in mitochondria or anaerobic fermentation. Through the interface shown in Figure 2, simulation can be used to answer queries like: "How much ATP is formed when an enzyme is inhibited or is absent from the system?", "What are the levels of intermediate metabolites during execution of a metabolic pathway?," "Which carbohydrate source is most efficient for ATP production?," and "What is the ratio of NADrid/NADox and FADrid/Fadox during the oxidation process".

Our validation approach imposes a new requirement on the model: Not only should the outcomes of the queries described above be accurate, but we should in addition obtain reasonable results when the model is altered in ways that correspond to known or plausible mutations. Mutations with known effects are most useful, because their effects provide a standard to compare simulation results against. As noted, carbohydrate oxidation can take place within two different environmental conditions, in presence of oxygen



**Fig. 1.** Agent Identification: the Cell Components Stereotypes



**Fig. 2.** Agent Identification: the Cell Components Stereotypes

(aerobic) or in its absence (anaerobic). The former takes place in the mitochondria, the latter in the cytoplasm. There are some cases where as consequence of malfunction due to DNA mutation in the mitochondrion, the aerobic pathway is blocked and metabolism is forced to change behavior with respect to new condition.

A mitochondrion DNA mutation can lead to a lack or disappearance of an enzyme involved in a metabolic pathways. It can provide different ATP or in the worst case the block the process. In nature, an aerobic microorganism can become anaerobic as a consequence of a mutation which prevents use of cellular respiration in mitochondria. As an example, we consider a mutation of the gene that produces one of the enzyme involved in the partial oxidation of pyruvate, i.e. it allows the passage of pyruvate from the cytoplasm to the mitochondrion, through the mitochondrial membrane, by allowing the aerobic respiration. If the mutation provokes the disappearance of an involved enzyme the microorganism is forced to adapt to a new context by using the anaerobic pathway. In this case, the pyruvate is transformed in lactate by producing NADox, allowint the cell to maintain glycolysis and to produce ATP.

The Cell-MAS model was not designed with such modifications in mind, and (not too surprisingly), we found that the correspondence between the software system and components in the biological system is imperfect. In our initial attempts to simulate the mutation described above, we found that the enzyme is not explicitly represented, but we can easily simulate the effect of its absence by a simple modification to the agent representing the membrane.

More interesting was analysis of the mechanisms by which aerobic and anaerobic pathways are governed. At a chemical level, we would expect that in a cell with both aerobic and anaerobic pathways, activation of the anaerobic pathway would arise from chemical conditions, and the anaerobic pathway could be activated by failure of the aerobic pathway. We discovered in the software, however, that this pathway has instead been selected artificially as a (user-settable) model parameter. This must have seemed a reasonable design decision for choosing between modeling aerobic and anaerobic organisms, but it is not sufficient for finer-grained investigation of their operation and interaction. Thus, in the early stage of our exploration of mutation analysis for model validation, we identified a characteristic of the model that should be refined to improve the correspondence between software model and natural system.

Specification diagrams describing the model suggested a fairly straightforward modification that would allow aerobic and anaerobic pathways to proceed in parallel when both are present; this should have allowed anaerobic pathways to engage when the aerobic pathway was blocked by mutation. However, the corresponding modification to the implementation did not work because dependence on that aspect of the model was spread among other implementation components (agents). This is a deviation from agent-based design criteria and can be considered an implementation fault, but it became evident only in validation.

## 5 Conclusions and Future Work

A key reason for choosing agent-based simulation of a biological system over other kinds of simulation model is the potential for establishing fine-grained structural and behavioral correspondence. This raises the question of how such a correspondence can be validated. We have proposed that the validation can be based not only on the behavior of the model as it stands, but on the extent to which changes to the model correspond to changes in the natural system. Strong structural correspondence between an agent-based system should mean that distinct, local changes in the biological system can be represented by corresponding distinct, local changes in the model. Behavioral correspondence at a fine level should mean that corresponding small changes lead to corresponding changes in the behaviors that emerge through chains of interactions.

While our experience is limited to multi-agent systems, the basic approach should be applicable in other kinds of models to the extent that they likewise maintain a structural as well as behavioral correspondence between model and modeled system.

Our preliminary experience applying mutation analysis to an agent-based simulation of carbohydrate oxidation in a cell exposed weaknesses in the structural and behavioral correspondence between the simulation model and the modeled system, as well as flaws in the implementation. This is not too surprising, since we are essentially testing against requirements that were not given in an explicit, testable form during the initial model development.

Our experience suggests, in fact, that post hoc application of mutation analysis is too late. It would be more appropriate to consider the effects of plausible mutations as a way of evaluating the model at all stages of design and implementation. This would be in line with accepted principles of software engineering. It is well understood that the cost

of repairing a software defect grows enormously as the defect propagates from initial requirements through specification, design, and implementation [1], and that verification and validation must therefore be applied throughout development. Anticipating and designing for change is also a fundamental principle in the engineering almost any kind of software system [16], and mutation analysis as part of a refinement of the model design methodology can be seen at least partly as a specialization and application of that principle. Mutation analysis throughout model design and implementation will be evaluated in future work.

**Acknowledgements** The authors are grateful to Marco Vita for extensive consultations on cell system behavior, particularly concerning the effects of DNA mutations in mitochondria. The first author thanks University of Oregon for providing a conducive environment for this work during her Fulbright leave. She also expresses her appreciation to Flavio Corradini for introducing her to formal methods research.

## References

- [1] B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.
- [2] T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Theoretical and empirical studies on using program mutation to test the functional correctness of programs. In *POPL '80: Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 220–233, New York, NY, USA, 1980. ACM Press.
- [3] K. P. Burnham, D. R. Anderson, G. White, C. Brownie, and K. H. Pollock. *Design and analysis methods for fish survival experiments based on release-recapture*, volume 5. American Fisheries Society Monograph, 1987.
- [4] N. Cannata, F. Corradini, E. Merelli, A. Omicni, and A. Ricci. An agent-oriented conceptual framework for biological systems simulation. *Transaction on Computation System Biology*, 2005. to appear.
- [5] F. Corradini, E. Merelli, and M. Vita. A multi-agent system for modelling the oxidation of carbohydrate cellular process. In *Proceeding of Int. Conference on Computational Science and its Applications, Modelling Complex Systems*, volume 3481 of *Lecture Notes in Computer Science*, pages 1265–1273. Springer Verlag, 2005.
- [6] M. d’Inverno and R. Saunders. Agent-based modelling of stem cell organisation in a niche. In Springer, editor, *Engineering Self-Organising Systems*, volume 3464 of *Lecture Notes in Artificial Intelligence*, 2005.
- [7] R. Garret and C. Grisham. *Biochemistry*. Sunder College Publishing, 1995.
- [8] P. Gonzalez, M. Cardenas, A. D. Camacho, O. Rosas, and J. L. Otero. Cellulat: an agent-based intracellular signalling model. *BioSystem*, 68:171–185, 2003.
- [9] R. G. Hamlet. Testing programs with the aid of a compiler. *IEEE Trans. Software Eng.*, 3(4):279–290, 1977.
- [10] L. J. Jagadeesan, A. Porter, C. Puchol, J. C. Ramming, and L. G. Votta. Specification-based testing of reactive software: Tools and experiments. In S. Berlin Heidelberg New York, editor, *Proceedings of the 19st International Conferece on Software Engineering (ICSE'97)*, pages 525–537, 1997.
- [11] N. Jennings. An agent-based approach for building complex software systems. *Communication of ACM*, 44(6):35–41, Apr. 2001.
- [12] H. Kitano. Systems biology: A brief overview. *Science*, 295:1662–1664, march 2002.
- [13] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

- [14] R. Milner. *Communication and mobile system: the  $\pi$ -calculus*. Cambridge University Press, 1999.
- [15] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 286–293, New York, USA, 19–23 July 2004. ACM.
- [16] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.
- [17] P. Wegner. Coordination as constrained interaction. In P. Ciancarini and C. Hankin, editors, *Coordination Languages and Models – Proceedings of the 1st International Conference (COORDINATION’96)*, volume 1061 of *LNCS*, pages 305–320. Springer-Verlag, Cesena (I), April 15–17 1996.
- [18] F. Zambonelli and A. Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283, Nov. 2004.