On the Role of Simulation in the Engineering of Self-Organising Systems: Detecting Abnormal Behaviour in MAS

Luca Gardelli Mirko Viroli Andrea Omicini
DEIS, Alma Mater Studiorum–Università di Bologna,
via Venezia 52, 47023 Cesena, Italy
Email: {luca.gardelli, mirko.viroli, andrea.omicini}@unibo.it

***Abstract*— The intrinsic complexity of self-organising multi-agent systems calls for the use of formal methods to predict global system evolutions at early stages of the design process. In particular, we evaluate the use of simulations of high-level system models to analyse properties of a design, which can anticipate the detection of wrong design choices and the tuning of system parameters, so as to rapidly converge to given overall requirements and performance factors.**

We take abnormal behaviour detection as a case, and devise an architecture inspired by principles from human immune systems. This is based on the TuCSoN infrastructure, which provides agents with an environment of artefacts—most notably coordination artefacts and agent coordination contexts. We then use stochastic π-calculus for specifying and running quantitative, large-scale simulations, which allow us to verify the basic applicability of our ID and obtain a preliminary set of its main working parameters.

I. Introduction

The trend in today information systems engineering is toward an increasing degree of complexity and openness, leading to rapidly changing requirements and highly dynamic environments. Since the cost of system management is becoming comparable to the cost of the system itself [1] we need new engineering methodologies and tools. In that sense social and natural sciences are recognised as rich sources of inspiration: e.g. the Autonomic Computing initiative tries to face complexity applying self-regulating mechanisms typical of biological processes [1], [2].

Self-organisation is a promising theoretical framework to reduce complexity of systems engineering. A system if said to be *self-organising* if it is able to re-organise itself upon environmental changes, by local interaction of its parts without any explicit pressure from the outside [3]. A system built according to this principle is usually able to perform complex tasks even though its components are far simpler when compared to a monolithic solution.

In this paper we continue along the line discussed in [4] in order to explore methodological aspects of the engineering of self-organising MASs. Because of the complexity inherent in these systems, and the difficulties in predicting their behaviour and properties, we find it crucial to exploit formal tools for simulating systems dynamics at the early stages of design. In the case of self-organising MASs, in fact, this approach appears to be almost unavoidable in order to nurture evolving ideas and design choices, and to effectively tune parameters of the final system.

Among the various formal models to specify quantitative aspects of MASs we promote the use of the stochastic π-calculus process algebra [5]—see [4] for more details on that decision. This language is basically unexplored in the context of self-organising MASs: on the one hand, its simulation tools are relatively recent (see e.g. [6]), and on the other, it was primarily inspired by the need to model biological systems [7]. However, we show it can be fruitfully applied to the MAS paradigm as well: as far as stochastic aspects are concerned, the typical complexity of agent internal machinery can be suitably abstracted away, focussing instead on agent interactions and high-level activity changes.

For this purpose tools like SpiM (Stochastic PI-calculus Machine [6]) can be effectively used to track the dynamics of global system properties in stochastic simulations, validating design directions, inspiring new solutions, and determining suitable system parameters.

In this paper, we apply these ideas to the study of an intrusion detection (ID) infrastructure for open MASs. In particular our focus is on detecting anomalies in agents behaviour: the solution we describe here is inspired by principles of human immune system [8]. The infrastructure we devise is based on the TuCSoN technology [9][1]. This allows us to structure a MAS not only in terms of agents, but also with *tuple centres* [10] as *coordination artefacts* [11] and *agent coordination contexts* [12](ACC) as *boundary artefacts* [13]. Coordination artefacts are used to model resources in the environment on which agents act upon. ACCs specify and enact the access policies which each agent is subject to, and can be used to both *(i)* reify relevant information about the agent/artefacts interaction, and *(ii)* to deny malicious agents to access the MAS environment.

To evaluate the impact of different design choices and parameters of the ID infrastructure—such as inspection/detection rates, number of inspectors, and the like—we simulate the behaviour of different scenarios using SpiM specifications.

The rest of the article is structured as follows. In Section II we briefly highlight the main mechanisms and properties of intrusion detection and the human immune system. In Section

[1] `http://tucson.sourceforge.net`

III we describe our general architecture for a MAS based on TuCSoN, and show how to develop an anomaly detection application. Section IV motivates the use of π-calculus and its stochastic extension, providing a simulation related to our ID domain using SpiM. Finally, Section V concludes by providing final remarks, and by listing some of the main directions for our future research.

II. Intrusion Detection and Immune System

In this section we first depict the main aspects of intrusion detection systems (IDSs), and then describe the structure and main principles regulating the human immune system. We are not concerned about accurately modelling or mimicking an immune system, instead we gather from there inspiration and principles for the engineering of secure self-organising applications.

A. Security and IDSs in Information Systems

There are several mechanisms used to protect information systems, but usually only the basic ones are implemented: *(i) authentication*, the identity is proved by the knowledge of a secret (e.g. password) or a physical unique property (e.g. fingerprint, retina, voice); *(ii) authorisation*: user actions on the system are constrained by its role and the policy linked to that role.

However, applications flaws typically cause these methods not to be sufficient alone [8]. For instance, protection at the host level is achieved using additional software such as firewalls, antivirus and many other specific tools. Furthermore authorisation policies cannot account for all possible sequences of actions, and a specific sequence might exhibit unexpected side-effects. In particular, it is in general too expensive and impractical (or even unfeasible) to intercept all emergent harmful paths at design-time.

Hence automated tools are a very useful support for the detection of malicious behaviour. In this direction, many efforts have already been spent in developing IDSs. An IDS tries to detect intruders and misuse of a target software system by observing users behaviour and deciding wether actions performed are symptomatic of an attack.

Efficiency of an IDS is evaluated by three parameters: *accuracy* (rate of false-alarms), *performance* (rate of audit processing), and *completeness* (rate of missed detection). *Misuse-based IDSs* try to detect intruders matching the actual user behaviour with known signatures of malicious behaviour. *Anomaly-based IDSs* try to detect behaviours that are different from what it is considered to be the normal activity. There has been already a lot of work for both approaches to deal with security issues either at the application, host and network level [8], [14], [15]. We are more concerned about the neat impact of such techniques, expressed in a stochastic manner, and how they can influence the design of a protection layer for a multi-agent system.

B. Human Immune System Overview

The human immune system protects the body against antigens, i.e. foreign molecules that trigger an immune response. It is composed by reactive non-specific barriers such as the skin, and by active mechanisms, i.e. the *innate* and the *acquired* immune system. The innate immune system protects the body against known antigens, i.e. it is not adaptive, while the acquired immune system improves during individual life, discovering and memorising new antigens. The acquired immune system is composed of different types of cells: here we consider only *lymphocytes* since they are responsible for the main form of immune response. The mix of lymphocytes, which changes over time, defines the set of detectable antigens: this let the immune system cover a larger space of antigens—a phenomenon called *dynamic coverage*. Lymphocytes can become a “memory” if they bind to several antigens: this mechanism allow for a faster response if an antigen is met again.

It is easy to notice that we can define a parallel between human immune system and security for electronics systems. Static non-specific barriers are realised by authentication mechanism, firewalls etc. The innate immune system is mapped into authorisation policies, antivirus, trojan removers, and misuse-detection. The acquired immune system instead is mapped into anomaly-detection systems, which are able to discover new threats.

III. Security in MAS

In the following we describe our reference architecture for MASs, and discuss how to devise a security layer drawing useful concepts and techniques from previous works on IDS, as well as principles from the human immune system.

A. A General Architecture for MASs

In this section we describe a general architecture for MASs based on the TuCSoN coordination infrastructure [9], showing an approach to ensure security applying principles of the immune system.

We consider a system that provides agents with services encoded in terms of coordination artefacts, i.e. runtime abstractions encapsulating and providing a coordination service, to be exploited by agents in social contexts expressed by coordination rules and norms [11]. Following the general model for artefacts [16], a coordination artefact could be characterised by a usage interface, a set of operating instructions, and a coordination behaviour specification, which can be exploited by cognitive agents to rationally use a coordination artefact.

Accesses of agents to these resources is restricted by an authentication procedure. When an agent enters the system an authorisation policy limits its actions allowing the exploitation of a limited set of services and resources—e.g. those it has payed for. This is accomplished by the notion of Agent Coordination Context (ACC) [12], [13]. An ACC works as agent interface towards the environment: it is like a control room providing e.g. buttons and displays to an human, which are the only means by which he/she can interact with the environment. Thus, the ACC enables and rules the interaction between the agent and the environment [12], and it is then able to capture security and organisation aspects in MASs. In particular, the ACC is the right place to put authorisation

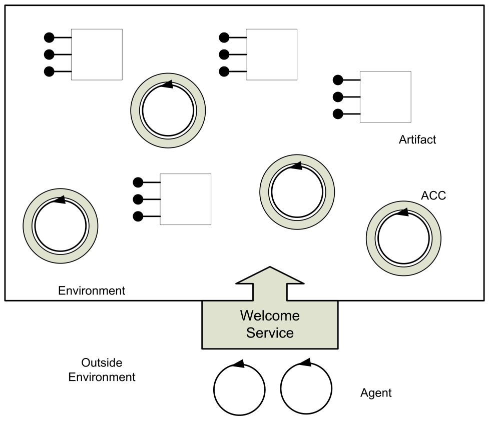

Fig. 1. A general architecture for a multi-agent system.

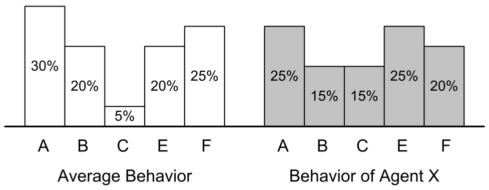

Fig. 2. The statistical approach for anomaly detection relies on the fact that the abnormal behaviour is distinguishable from the normal one. This can be restated in *the behaviour distribution of abnormal agents (right) is very different—at least for the critical actions—from the distribution of normal ones (left).*

policies, typically specified using a Role Based Access Control model (RBAC) [17]. The whole architecture is depicted in Figure 1.

Usually the two mechanisms of authentication and authorisation are considered to guarantee a sufficient degree of protection. However we promote the idea—as pointed out in the intrusion detection community—that a dynamic system is better protected by additional dynamic mechanisms. Correspondingly, we introduce a layer aimed at detecting anomalies in agents behaviour inspired by the immune system as well as by previous works on IDSs [8], [14].

B. Anomaly Detection in MASs

Let us consider agents willing to exploit a specific artefact: we can trace their behaviour "for a while" and then create an average distribution of actions over that resource. We can consider that distribution to be the "normal" way for agents to interact with a particular artefact (Figure 2 left). From now on it is possible to observe an agent in order to build its particular distribution of actions: the deviation from the average distribution might be a symptom of intrusion or abnormal activity. For instance, if the action C is critical then the agent X (Figure 2 right) should be inspected to decide wether he is acting properly or might cause problems.

In order to support the process described above, a mechanism to observe the agent behaviour is needed, e.g. by using logging tools. This approach is valid under two hypotheses:

1) the number of traces is such that the data is statistically significant
2) the percentage of agents exhibiting abnormal behaviour is sufficiently low during the initial observation stage

The former hypothesis is quite straightforward and must hold true every time dealing with statistical data. The latter is very difficult to prove because one would have to check every action and in most situations this might be unfeasible or simply not affordable without automated tools. But we can reasonably argue that most of users of a system are interested in exploiting a resource rather than to hack it. Furthermore the second hypothesis affects the threshold value used to decide wether an agent is dangerous or not: this value is as reliable and accurate as the number of anomalies is low during the initial stage. Since we are performing the detection task on-line, the actions distribution might changes over time, so we should also consider some tolerance ranges.

Referring to section III-A we describe now how the security layer fits into the general architecture. First we need a way to provide observability of the interaction agent-artefact: basically we have two choices:

- providing inspectors with access to ACCs
- reifying the action in a specific artefact for logging

Since we need two kinds of information, the average behaviour and the individual one, we can exploit both mechanisms: e.g. we can log the individual behaviour in the ACC while the average behaviour can be handled by another shared artefact. If agents privacy is not a main concern we can also reify both information using another artefact, which aggregates actions to define the average and individual signature.

Since observability mechanisms have been provided, now we need a set of agents whose goal is to observe periodically the use of resources. The actual *number of agents inspecting* and the *rate of inspection* are parameters of the security systems that should be dynamically tunable. When an agent detects abnormal behaviour, it should report it to the proper authority, which then decides wether to invalidate the ACC or not: invalidating the ACC means denying any further access to resources. This authority might be a human or artificial agent depending on the complexity and criticality of the decision process.

C. An Artefact for Logging Purpose

After we described the scenario, the architecture and the principles, we aim now at actually designing the artefact that could support the anomaly-detection task. For the sake of simplicity, our hypothesis is that there are no concerns of privacy for agents, i.e. it is not a problem to publish all agent actions: thus, in order to adopt the second approach described in section III-B, we only need to worry about the artefact design.

We use TuCSoN infrastructure [9] as our main source for artefacts: in particular, artefacts for logging are suitably-programmed ReSpecT tuple centres [10]. Hence what we need for a complete description of such an artefact are the templates for all the tuples used for the representation, and the ReSpecT specifications for handling the log tuples.

In particular, we only need three templates, respectively (i) for actions logging, (ii) for the individual behaviour signature, and (iii) for average behaviour signature.

```
%1) This reaction is executed only the first time the
%   action identified by ActionID is performed for the
%   first time over a specific artifact
reaction( out_r(action(AgentID, ActionID)), (
    in_r(action(AgentID, ActionID)),
    no_r(average_signature(ActionID, Count)),
    out_r(average_signature(ActionID, 1)),
    out_r(individual_signature(AgentID, ActionID, 1))
)).

%2) This reaction is triggered by an action identified by
%   ActionID has already been performed the agent
%   identified by AgentID
reaction( out_r(action(AgentID, ActionID)), (
    in_r(average_signature(ActionID, Count)),
    in_r(individual_signature(AgentID, ActionID, Counti)),
    in_r(action(AgentID, ActionID)),
    Count1 is Count + 1,
    Counti1 is Counti + 1,
    out_r(average_signature(ActionID, Count1)),
    out_r(individual_signature(AgentID, ActionID, Counti1))
)).

%3) This reaction is triggered by an action identified
%   by ActionID has already been performed from other
%   agents, but it's the first time for the agent
%   identified by AgentID
reaction( out_r(action(AgentID, ActionID)), (
    in_r(action(AgentID, ActionID)),
    in_r(average_signature(ActionID, Count)),
    no_r(individual_signature(AgentID, ActionID, Counti)),
    Count1 is Count + 1,
    out_r(average_signature(ActionID, Count1)),
    out_r(individual_signature(AgentID, ActionID, 1))
)).

%4) This reaction should never be triggered in normal
%   situation but it's useful to recover from inconsistencies
reaction( out_r(action(AgentID, ActionID)), (
    in_r(action(AgentID, ActionID)),
    no_r(average_signature(ActionID, Count)),
    in_r(individual_signature(AgentID, ActionID, Counti)),
    Counti1 is Counti + 1,
    out_r(average_signature(ActionID, 1)),
    out_r(individual_signature(AgentID, ActionID, Counti1))
)).
```

Fig. 3. ReSpecT specification for the logging artefact behaviour.

1) `action(agentID, actionID)`
2) `individual_signature(agentID,actionID, count)`
3) `average_signature(actionID, count)`

The system provides the first tuple each time an action is performed: this tuple triggers ReSpecT reactions which update signature tuples, then it is discarded. Each signature tuple is a counter for an action, which is incremented each time that action is performed by an agent: recording such information for each action makes is possible for an agent to build the actual signature.

Each time an artefact is introduced in the environment the signature of normal behaviour it is automatically built, which becomes significant only when the number of request exceeds a certain threshold. Figure 3 includes the whole specification[2].

Now that we have the anomaly detection support we must decide the parameters of the security systems: number of inspecting agents and rate of inspection—see section III-B for details. Given the computational cost of inspection and assuming a certain percentage of abnormally-behaving agents, we can simulate the system in order to predict the good values for system parameters. In the next section we describe Stochastic π-Calculus and how to exploit it for such purpose.

IV. Simulations in π-Calculus

In this section we briefly introduce π-calculus [18] and its stochastic extension [5]. Then we present the results obtained by simulating a Stochastic Pi-Calculus specification using the Stochastic Pi Machine (SpiM) [6].

[2]The source code for the experiments in this paper can be downloaded from `http://www.alice.unibo.it/download/spim/woa2005.zip`.

A. The π-Calculus

The π-calculus is a formal model developed to reason about concurrency [18]: it is a language for describing and analysing systems consisting of agents (or processes) which interact with each other. The basic entity is a *name*, which is used as an unstructured reference to a synchronous channel where messages can be sent and received. In its simpler version, a process is built from names according to the syntax:

$$P ::= 0 \mid \sum_{i \epsilon I} \pi_i.P_i \mid (P|Q) \mid !P \mid (\nu x)P \tag{1}$$

0 is the empty process. The summation $\sum_{i\epsilon I} \pi_i.P_i$ means that an agent might perform any prefix action π_i, and correspondingly continues as P_i behaviour: prefix forms π_i are of the kind $\bar{y}x$ (send the name x at channel y), $y(x)$ (wait for a name at channel y and rename it as x), and τ (perform a silent action). A composition $P|Q$ represents P and Q executed in interleaved concurrency. A replication $!P$ means that (infinitely) many copies of P can be executed concurrently (like $P|P|....$). Restriction $(\nu x)P$ creates the new name x and bounds its use in P. The semantics of π-calculus can be described by a transition system, where the transition relation $P \longrightarrow P'$—a process P moving to P' by the occurrence of an inner interaction—is defined by operational rules [18].

B. On Stochastic Models

In general, each formal model whose semantics is given by a transition system can be extended to a stochastic version, resorting to the idea of Markov transition system. There, each transition $P \xrightarrow{r} P'$ is labelled with a *rate* r, a non-negative real value which describes how the transition probability between P and P' increases with time. Stochastic models allow for quantitative simulations, for rates can be used to express aspects such as probability, speed, delays, and so on.

However, from an engineering perspective the choice of which language is used for describing processes is a crucial one, for the system to be simulated is to be effectively represented in the language.

Three basic options are available: *(i)* automata, like finite-state ones, where the system is described by state changes and by supporting data structures (such as stacks); *(ii)* nets, like Petri Net, where the system is described by a marking of tokens spread over a graph; and finally *(iii)* process algebras, like π-calculus, where the system is described by a composition of interacting entities. We find the third approach to be the best suited for describing quantitative aspects of complex MASs—such as self-organising ones. On the one hand, differently from automata, process algebras allow to express concurrent activities (agents in this case). On the other hand, differently from nets, process algebras allow for full compositionality: this property is a particularly relevant one, as it allows to express agents (and artefacts) with different roles separately, and then simply reuse such definitions to express the whole system model by composition (parallel composition, summation and replication).

C. Stochastic π-Calculus and π-Machine

Priami [5] introduced a stochastic extension to π-calculus. Each channel name is associated with an activity rate r: the delay of an interaction through that channel (representing the use of a resource [5]) is then a random variable with an exponential distribution defined by r. Exponential distributions are used because they enjoy the memoryless property, i.e. each transition is independent from the previous one. Given a channel name x, the probability p_i of a transition $P \xrightarrow{r_i} P_i$ representing an interaction through x is the ratio between its rate r_i and the sum of rates of the n transitions through x enabled by P:

$$p_i = \frac{r_i}{\sum_{j=1..n} r_j}, \quad 1 \leq i \leq n \ . \tag{2}$$

In the following, we consider the SpiM [6] implementation for the stochastic π-calculus interpreter.

D. Simulating Anomaly Detection

In this section we discuss how to exploit Pi-Calculus for simulating the previously described security system. We are not going to describe here how to write stochastic pi-calculus specifications since it has already been widely covered by the literature, e.g. see [7], [6]. In order to give an insight of the specification process, we have mapped an agent to a set of concurrent processes which are able to send/receive signal to/from other agents: a more detailed description of our approach is available in [4].

We consider an environment in which agents can enter and leave after being authenticated and authorised: since we want to keep the average number of agents constant we set the entering and leaving rates to be equal. The simulation parameters are *(i)* the number of agents within the system at $t = 0$, *(ii)* the "concentration" of normal vs. abnormal agents, *(iii)* the rates at which normal behaving agents enter the system, *(iv)* the rates at which abnormally-behaving agents enter the system, *(v)* the duration of the simulation.

Then we add inspector agents to the system, which observe the behaviour of external agents: each inspector performs the same task but independently from the others. The anomaly detection system parameters are

- the number of inspectors
- the rate of inspections

Both parameters should be dynamically adjustable, e.g. if the system is under attack it can raise its defences: for the whole duration of the simulation we consider them to be constant.

The results are plotted in Figure 4. With the chosen values for the parameters, this chart let us observe that the system is able to exclude agents behaving abnormally within about 400 time units. If that is acceptable—i.e. the system still working at the target quality level—we can choose those parameters value for tuning the actual system. Since these results have been obtained by simulation they must be validated by the actual system: this last step is going to be addressed soon in the future.

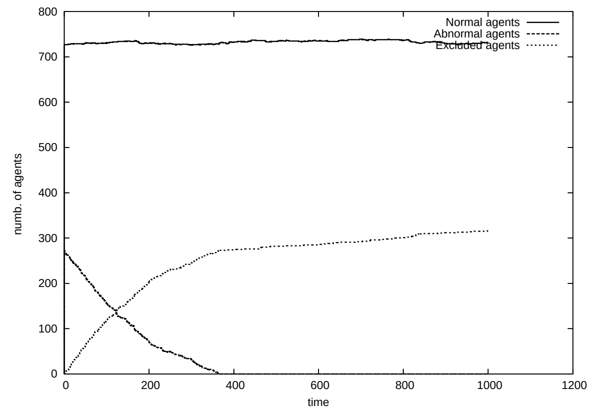

Fig. 4. A simulation of a simple system where normally- and abnormally-behaving agents can enter and leave. Inspectors limit the activity of abnormal agents.

V. Conclusion and Future Works

This paper is based on a previous work [4] where we started putting together the elements of a framework for engineering self-organising applications. While in [4] we focused on the applicability of stochastic pi-calculus, in this paper have given more details on domain specific issues, programming artefacts to support anomaly detection for MAS and targeting the simulations to the specific case. We consider MASs composed by agents and artefacts [13], [11], and we build simulations in a stochastic process algebra setting able to tune system parameters at design time. We are developing an anomaly detection system for TuCSoN and in parallel we consider this application as a case to assess the impact of simulation in engineering self-organising system.

The system depicted is based on the TuCSoN coordination infrastructure, it features the remarkable notion of ACCs, which enable to control agent actions, reify information on action sequences (to be read by the infrastructure and/or other agents), prevent agent actions from a given point in time. For the architecture and general principles we took inspiration from the human immune system and previous works on IDSs. For the methodology, we relied on formal simulation and modelling via stochastic π-calculus, which—even though is a quite new language in the context of the MAS community—showed its effectiveness as a design tool.

Whereas our experiments need to be further detailed, we believe they generally emphasise the ability of the proposed approach to help the MASs developers to anticipate design decisions and strategies at the early stages of design—before actually developing prototypes and testing them.

We have a basic prototype of anomaly detection systems on top of TuCSoN-based MASs, but we need to further detail and test it in order to validate simulation results. Other than testing security at the artefact level, we also plan to explore the implications of extending this approach to a network of nodes hosting the same sort of artefacts.

Finally, in this paper we have only been concerned with self-organisation mechanisms. In future works we intend to explore the dynamics that causes system properties to emerge. For example, the uniqueness of the human immune system provide the human species with a greater probability to survive to a specific antigen: this emergent property could be very important for distributed system.

References

[1] P. Horn, "Autonomic computing: IBM's perspective on the state of information technology," IBM Corporation, Tech. Rep., 15 Oct. 2001. [Online]. Available: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=642200

[3] F. Heylighen, "The science of self-organization and adaptivity," in *Knowledge Management, Organizational Intelligence and Learning, and Complexity*, ser. The Encyclopedia of Life Support Systems. EOLSS Publishers, 2003.

[4] L. Gardelli, M. Viroli, and A. Omicini, "On the role of simulations in engineering self-organizing MAS: the case of an intrusion detection system in TuCSoN," in *3rd International Workshop "Engineering Self-Organising Applications" (ESOA 2005)*, S. Brueckner, G. Di Marzo Serugendo, D. Hales, and F. Zambonelli, Eds., AAMAS 2005, Utrecht, The Netherlands, 26 July 2005, pp. 161–175.

[5] C. Priami, "Stochastic pi-calculus," *Computer Journal*, vol. 38, no. 7, pp. 578–589, 1995.

[6] A. Phillips, "The stochastic Pi machine (SPiM)," 2005. [Online]. Available: http://www.doc.ic.ac.uk/~anp/spim/

[7] A. Phillips and L. Cardelli, "Simulating biological systems in the stochastic pi-calculus," Microsoft Research, Cambridge, UK, Tech. Rep., July 2004.

[8] S. Forrest, S. A. Hofmeyr, and A. Somayaji, "Computer immunology," *Commun. ACM*, vol. 40, no. 10, pp. 88–96, 1997.

[9] A. Omicini and F. Zambonelli, "Coordination for internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, 1999.

[10] A. Omicini and E. Denti, "From tuple spaces to tuple centres," *Science of Computer Programming*, vol. 41, no. 3, pp. 277–294, Nov. 2001.

[11] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini, "Coordination artifacts: Environment-based coordination for intelligent agents," in *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, Eds., vol. 1. New York, USA: ACM, 19–23 July 2004, pp. 286–293. [Online]. Available: http://portal.acm.org/citation.cfm?id=1018409.1018752

[12] A. Omicini, "Towards a notion of agent coordination context," in *Process Coordination and Ubiquitous Computing*, D. C. Marinescu and C. Lee, Eds. CRC Press, Oct. 2002, ch. 12, pp. 187–200.

[13] A. Ricci, M. Viroli, and A. Omicini, "Agent coordination context: From theory to practice," in *Cybernetics and Systems 2004*, R. Trappl, Ed., vol. 2. Vienna, Austria: Austrian Society for Cybernetic Studies, 2004, pp. 618–623, 17th European Meeting on Cybernetics and Systems Research (EMCSR 2004), Vienna, Austria, 13–16 Apr.2004. Proceedings.

[14] A. Somayaji, S. Hofmeyr, and S. Forrest, "Principles of a computer immune system," in *1997 Workshop on New Security Paradigms (NSPW '97)*, T. Haigh, B. Blakley, M. E. Zurbo, and C. Meodaws, Eds. New York, NY, USA: ACM Press, 23–26 Sept. 1997, pp. 75–82. [Online]. Available: http://portal.acm.org/citation.cfm?id=283699.283742

[15] H. Inoue and S. Forrest, "Anomaly intrusion detection in dynamic execution environments," in *2002 Workshop on New Security Paradigms (NSPW '02)*, C. Serban, C. Marceau, and S. Foley, Eds. New York, NY, USA: ACM Press, 23–26 Sept. 2002, pp. 52–60. [Online]. Available: http://portal.acm.org/citation.cfm?id=844112

[16] M. Viroli, A. Omicini, and A. Ricci, "Engineering MAS environment with artifacts," in *2nd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2005)*, D. Weyns, H. V. D. Parunak, and F. Michel, Eds., AAMAS 2005, Utrecht, The Netherlands, 26 July 2005, pp. 62–77.

[17] A. Omicini, A. Ricci, and M. Viroli, "RBAC for organisation and security in an agent coordination infrastructure," *Electronic Notes in Theoretical Computer Science*, vol. 128, no. 5, pp. 65–85, 3 May 2005, 2nd International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo'04), 30 Aug. 2004. Proceedings.

[18] R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes, part I/II," *Information and Computation*, vol. 100, no. 1, 1992.