On the Role of Simulations in Engineering Self-organising MAS: The Case of an Intrusion Detection System in TuCSoN

Luca Gardelli, Mirko Viroli, and Andrea Omicini

Alma Mater Studiorum–Università di Bologna,
via Venezia 52, 47023 Cesena, Italy
{luca.gardelli, mirko.viroli, andrea.omicini}@unibo.it

Abstract. The intrinsic complexity of self-organising MASs (multi-agent systems) suggests the use of formal methods at early stages of the design process in order to predict global system evolutions. 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 intrusion detection (ID) 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 artifacts—most notably coordination artifacts 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.

1 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. As a consequence, the cost of system management is becoming comparable to the cost of the system itself [1]. This phenomenon has led to the challenge of discovering new ways of engineering systems inspired by social and natural sciences. For instance the Autonomic Computing initiative tries to face complexity by taking inspiration from the self-regulating behaviour of the biological processes [1, 2]. Other visions, such as Amorphous Computing [3] and Spray Computers [4] have sprung sharing the same objective. Self-organisation is a promising approach to tackle these issues without explicit pressure or constraints from outside the system: self-organisation is usually the result of the interaction and coordination at a local level of a set of agents, each simpler in structure than the global task achieved [5].

In this paper we conduct a preliminary study on methodological aspects of the engineering of self-organising MASs. Because of the intrinsic complexity of these systems, and the difficulties in predicting their behaviour and properties,

S.A. Brueckner et al. (Eds.): ESOA 2005, LNAI 3910, pp. 153–166, 2006.

we find it crucial to exploit formal tools for simulating their 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—based on process algebras [6], Petri Nets [7], and automata [8]—we promote the use of the stochastic π-calculus process algebra [9]. While it allows for a great expressiveness in representing key aspects such as interactions and concurrent activities, π-calculus also features full compositionality and modularity properties, which are crucial to scale up with system complexity.

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. [10]), on the other, it was primarily inspired by the need of modelling biological systems [11]. However, we show that stochastic π-calculus 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. Then, tools like SpiM (Stochastic PI-calculus Machine [10]) 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 study an intrusion detection (ID) infrastructure for MASs, which detects malicious agents in an open context. The solution we consider is inspired by principles of human immune system [12], where agents resembling lymphocytes are dynamically created and updated with the goal of detecting malicious behaviours. The infrastructure we devise is based on the TuCSoN technology [13][1]. This allows us to structure a MAS not only in terms of agents, but also with *tuple centres* as *coordination artifacts* [14] and *agent coordination contexts* (ACC) as *boundary artifacts* [15]. Coordination artifacts are used to model resources in the environment on which (potentially) malicious 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/artifacts interaction, which the lymphocyte agents can inspect to detect abnormal sequences of actions, and *(ii)* to abruptly 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 lymphocytes, upgrade policy for lymphocytes, and the like—we simulate the behaviour of different scenarios using SpiM specifications.

The rest of the article is structured as follows. In Section 2 we briefly highlight the main mechanisms and properties of intrusion detection and the human immune system, and apply them to a general architecture for a MAS based on TuCSoN. Section 3 motivates the use of π-calculus and its stochastic extension, providing an application example related to our ID domain using SpiM. In Section 4 we examine performance of several scenarios of the MASs infrastructure

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

introduced. Finally, Section 5 concludes providing final remarks and listing main directions for future research.

2 Human Immune System and a MAS Architecture

In this section we first depict the main aspects of ID systems (IDSs), and then describe the structure and main functionalities of the human immune system. We are not concerned about accurately modelling or mimicking the human immune system, but we rather gather useful principles to engineer secure self-organising applications, which are then used to sketch a general architecture for MASs applying some of these concepts.

2.1 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.

However, applications flaws typically cause these methods not to be sufficient alone [16]. For instance, in operating systems there is a need to use additional software such as firewalls, antivirus and many other specific tools that must be configured and kept updated to prevent unplanned attacks. Furthermore authorisation policies cannot account for all possible sequences of actions: 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.

As a consequence, 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 abnormal behaviour and misuse of a target software system by observing it and deciding wether actions performed by a user/agent are symptomatic of an attack [17]. 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).

Usually IDSs are implemented as expert systems: they synthesise signatures of malicious behaviours from lists of sequences of actions. However, for generality, we do not focus here on specific techniques to handle signatures, for they mostly concern the IDS research community, and are typically very dependent on the application domain—they differ e.g. in the context of system calls [16, 12], TCP/IP connection addresses [18], memory accesses, and so on. Rather, we are concerned about the neat impact of such techniques, expressed in a stochastic manner, and how they can influence the design of a protection system.

2.2 Human Immune System Overview

The human immune system protects the body against foreign pathogens. We use the term *antigen* to refer to any foreign molecule that triggers an immune response by the human body. The human immune system consists of many layers each exploiting several mechanisms to increase the degree of protection. A first kind of protection is provided by the physiological barriers, i.e. skin, temperature and pH, that are reactive and non-specific—i.e. not triggered by a specific class of antigens.

The human immune system also provides active mechanisms: the innate and the acquired immune system. The first is there since birth, and is composed of scavenger cells (e.g. *fagi*) wandering in the lymphatic system, which are able to detect and kill only a fixed subset of antigens. It is not adaptive, hence it is not able to protect the body from new kinds of antigens. This issue is instead the main concern of the acquired immune system: it improves during individual life, learning and memorising new antigens. The acquired immune system is composed of different types of cells: we consider only lymphocytes for they are responsible for the main form of immunity.

Lymphocytes are produced in the bone marrow and are sent to mature in the thymus. There, they are exposed to self-cells (body): if they bind the self-cells lymphocytes are eliminated. When the process of maturation ends, lymphocytes are released in the lymphatic systems. Lymphocyte surface is covered with different sets of receptors that are able to bind to different classes of antigens. This phenomenon, called dynamic coverage, lets the immune system cover part of the space of antigens (10^{16}) with a sensibly smaller set of lymphocytes (10^{10}). Receptors are randomly generated by a process of variation and selection. Lymphocytes have a short life (about two days), but if during this period they bind to several antigens they become "memory" and their life is extended. This strategy, in combination with receptor generation process, allows to discover new antigens and to apply a faster response if the antigen is met again.

2.3 A General Architecture for MAS Applications

In this section we describe a general architecture for MASs based on the TuCSoN coordination infrastructure [13], showing an approach to ensure security applying principles of the immune system—for space reasons we only sketch main design details.

We consider a system that provides agents with services encoded in terms of coordination artifacts, i.e. runtime abstractions encapsulating and providing a coordination service, to be exploited by agents in social contexts expressed by coordination rules and norms [14]. The basic model of coordination artifacts is characterized by *(i)* a usage interface, *(ii)* a set of operating instructions, and *(iii)* a coordination behaviour specification, which can be exploited by cognitive agents to rationally use a coordination artifact.

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 realised by the notion of Agent Coordination Context (ACC) [19, 15]. 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

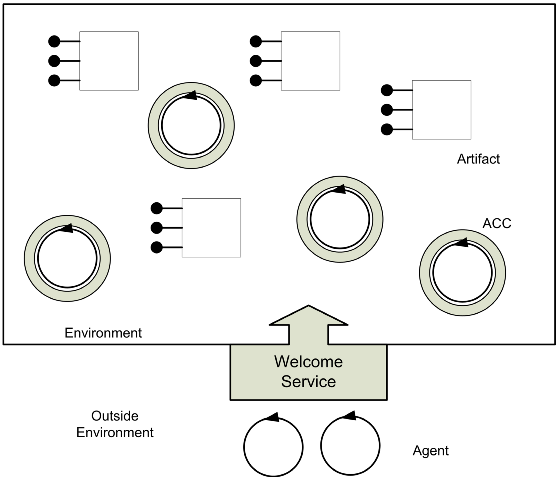

Fig. 1. A general architecture for a multi-agent system

ACC enables and rules the interaction between the agent and the environment [19], and it is then able to model security and organisation aspects in MASs [15]. In particular, the ACC is the right place to put authorisation policies, typically specified using a Role Based Access Control model (RBAC). 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, as described in previous section, we instead promote the idea that a dynamic system is better protected by additional dynamic mechanisms. So we introduce other techniques inspired by the immune system and previous work on IDSs [16, 20].

As the human immune system has barriers, we consider authentication and authorisation procedures to be information systems barriers. We need an extra layer to cope with dynamic issues concerning harmful sequences of actions which cannot be statically identified as such. So we add a layer inspired by the acquired immune system of humans. Forrest et al. suggested a list of organising principles that should guide the design of a computer security system: distributability, multi-layering, no (totally) secure layer, diversity, disposability, autonomy, adaptability, and identity via behaviour—see [16] for more details on them.

We introduce a class of (distributed) agents, which we call agentLy, to model lymphocyte task: they should observe the actions performed by agents (autonomy) as reified by the ACC in charge of control them. When they detect a suspicious behaviour (identity via behaviour) they should dispatch an alert message to the authority that can trigger the proper response. In our case, this authority could invalidate the agent ACC, i.e. denying any further access to the system resources. Each agentLy is able to detect only a subset of the possible bad behaviours (diversity). If after a time interval it does not detect any attack then it can be replaced by another one (disposability). The security layer dynamically covers the space of possible bad behaviours. The system should be able to learn and synthesise new signatures of abnormal behaviour (adaptability). As already stated we do not deal with synthesising these signature, but we encapsulate these skills in our agentLys and then abstract away from their details.

Now that we have sketched the system architecture and pointed out which properties could be useful, we are interested in predicting the behaviour of the system in large-scale scenarios, and the performance factors we should expect. We try to answer these questions in the following sections.

3 Simulations in π-Calculus

In this section we briefly introduce π-calculus [6] and its stochastic extension [9]. Then, we present an example of a simple program using the Stochastic Pi Machine (SpiM) [10].

3.1 The π-Calculus

The π-calculus is a formal model developed to reason about concurrency [6]: it is a language for describing and analysing systems consisting of agents (or processes) which interact with each other, and whose configuration of neighborhood is continually changing [21]. 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 version of π-calculus that allows to send/receive a single name only to/from a channel is called monadic. The polyadic version of the π-calculus also allows more names in a single communication to be sent/received [21]. The semantics of π-calculus can be described by a transition system, where the transition relation $P \longrightarrow P'$—process P moving to P' by the occurrence of an inner interaction—is defined by operational rules [6].

3.2 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 [22]. There, each transition $P \xrightarrow{r} P'$ is labelled with a *rate* r, a nonnegative real value that scales 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 particularly relevant, as it allows to express agents (and artifacts) with different roles separately, and then simply reuse such definitions to express the whole system model by composition (parallel composition, summation and replication).

3.3 Stochastic π-Calculus and π-Machine

Priami introduced a stochastic extension to π-calculus [9]. Each channel name is associated with an activity rate r: the delay of an interaction through that channel (representing the use of a resource [9]) 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 [23]. 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:

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

We consider the SpiM implementation for the stochastic π-calculus interpreter [10]. As an example, we want to simulate a simple scenario where agents can enter and leave a system after being authenticated and authorised. The system parameters are *(i)* the number of agents within the system at $t = 0$, *(ii)* the "concentration" of malicious vs. legitimate agents, *(iii)* the rates at which legitimate agents enter and leave the system, and *(iv)* the rates at which malicious agents enter and leave the system.

We are simulating the system for 1000 time units. We want to keep the average number of agents constant so that the entering rate is equal to the leaving one. Code and simulation results are reported in Figure 2 and 3.

The initial part of the specification introduces channel names—we set e.g. the small rate 0.1 for agents entering and leaving the system (`delayEnter` and `delayLeave`), and a 70% ratio of legitimate (good) agents vs. malicious (bad) ones (`isGoodEA` and `isBadEA`). In the behavioural part, we instantiate 1000

```
1000.0 (* Simulation duration *)

(* Counters *)
new CountBad:1000000.0:<>               new CountGood:1000000.0:<>
new isBadEA:300000.0:<>                 new isGoodEA:700000.0:<>
new delayLeaveG:0.1:<>                  new delayLeaveB:0.1:<>
new delayEnterG:0.1:<>                  new delayEnterB:0.1:<>
new EA:<>        (* Agent*)
new BEA:<>       (* Malicious Agent*)      new GEA:<>    (* Legitimate Agent *)
new ActionLeaveG:<>                     new ActionLeaveB:<>
new ActionEnterG:<>                     new ActionEnterB:<>
new LB:<>        (* Signal: malicious agent must leave *)
new LG:<>        (* Signal: legitimate agent must leave *)
new InitEA:<int> (* Initialize agents *)
new Timer:<<>,<>>

(* Behaviour *)
( !GEA(); LG()
| !BEA(); LB()
    (* Manage the flow of agents (in/out the system) *)
    (* 4 timers:
        - legitimate agents entering and leaving
        - malicious agents entering and leaving  *)
| Timer<ActionEnterG, delayEnterG>
| !ActionEnterG(); (CountGood<> | GEA<> | Timer<ActionEnterG, delayEnterG>)
| Timer<ActionEnterB, delayEnterB>
| !ActionEnterB(); (CountBad<> | BEA<> | Timer<ActionEnterB, delayEnterB>)
| Timer<ActionLeaveG, delayLeaveG>
| !ActionLeaveG(); (CountGood() | LG<> | Timer<ActionLeaveG, delayLeaveG>)
| Timer<ActionLeaveB, delayLeaveB>
| !ActionLeaveB(); (CountBad() | LB<> | Timer<ActionLeaveB, delayLeaveB>)
    (* Initialize Lymphocytes, Good and Bad agents (t=0) *)
| !EA(); (isBadEA<>; (CountBad<> | BEA<> ) + isGoodEA<>; (CountGood<> | GEA<>))
| !isBadEA()
| !isGoodEA()
| !InitEA(n); if n>0 then ( EA<> | InitEA<n-1> )
| InitEA<1000>

(* Library *)
| !Timer(c,r); (r<>|r();c<> )
)
```

Fig. 2. Source code of the model of a simple system where malicious and legitimate agents can enter and leave

agents (`InitEA<1000>`) and then counted the number the evolution of legitimate and malicious agents (`CoundGood` and `CountBad`), which are those shown in the plot[2]. This example is used as basis for developing interesting dynamics in more complex cases, as developed in the following.

4 Simulating Self-organising Systems

In this section we discuss three scenarios, exemplifying an incremental design process for our IDS. We refer to the general architecture described in section 2.3 (figure 1). For space reasons the complete simulation code for these examples is not reported[3].

[2] The whole description of this code is avoided for brevity.

[3] The interested reader can download them from the site: `http://www.alice.unibo.it/download/spim/esoa.zip`

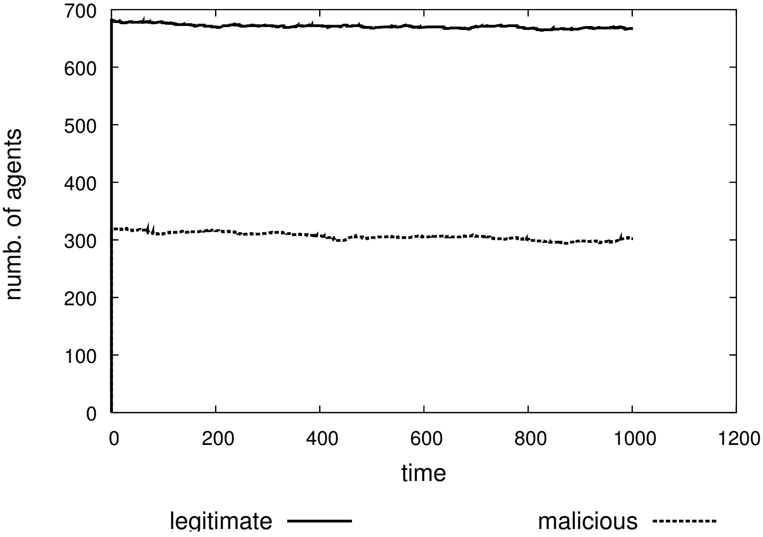

Fig. 3. Simulation of the system in Figure 2

4.1 Scenario 1

Starting from the previous example we add to the system lymphocyte agents (agentLy), which observe the behaviour of external agents. Its role, already described in Section 2.3, is to monitor ACC states to detect wrong behaviours. Each agentLy performs independently from the others, but they all share the same parameters. We add to the previous list the following parameters

- the number of agentLys (`10`)
- the rate at which an agentLy performs inspections (`delayInspection:0.5`)
- the probability that an agentLy, during inspection, detects an agent as malicious (10%, due to `matchP:100000.0` and `matchN:900000.0`).

The first and second parameter should be dynamically adjusted: for instance if the system is under attack it can raise its defenses. We consider them as a constant for the duration of the simulation. The third parameter instead should be regarded as a contract between the agentLy and the system. The system can replace agentLys that do not comply with the contract. This can be e.g. realised by making agentLys have their own ACC, and the infrastructure checking their detection rate.

The chart in Figure 4 shows the results of the simulation. With the chosen values for the parameters, we observe that the system is able to eliminate the activity of malicious agents within around 400 time units.

4.2 Scenario 2

In this scenario we introduce the idea that agentLys feature a limited lifetime. Furthermore we want to model two classes of agentLys which have different abilities to detect malicious activity. We call the ones which perform better

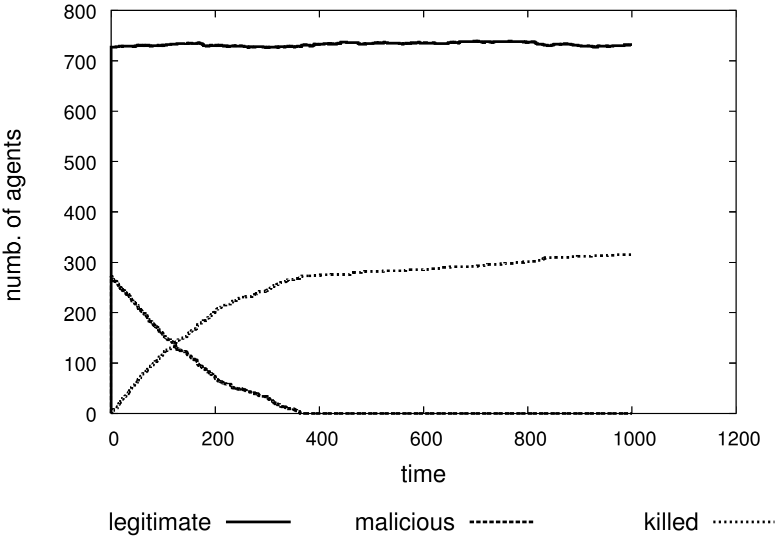

Fig. 4. A simulation of a simple system where malicious and legitimate agents can enter and leave. AgentLys limit the activity of malicious agents.

agentLyA, while the other ones agentLyB. Because an agentLyA performs better it will be rewarded with a longer lifetime, modelling the memory effect. So we add the following parameters:

- the probability that an agentLy belongs to class A rather than B (20% due to `isBadL:800000.0` and `isGoodL:200000.0`)
- the lifetime of the two classes of agentLys (`GL<200>` and `BL<100>`)
- the probability that an agentLyA, during inspection, detects an agent as malicious (30% due to `matchGP:300000.0` and `matchGN:700000.0`)
- the probability that an agentLyB, during inspection, detects an agent as malicious (10% due to `matchBP:100000.0` and `matchBN:900000.0`).

Figure 4 shows the results of the simulation. With the chosen values for the parameters the system is able to limit the activity of malicious agents. As expected, this system performs better than the previous, for now malicious agents are eliminated in less than 300 time units. Furthermore, by increasing the probability that an agentLy belongs to class A, we can expect the performance of detection to increase accordingly.

4.3 Scenario 3

In this scenario we introduce the hypothesis that the types of malicious behaviours are not uniformly distributed in the space of possible behaviours. So an agentLy that has a poor performance must learn from those who perform better. To account for this phenomenon we modelled the possibility for the system to clone an agentLyA to replace an agentLyB. This can be realised in our MAS by imposing in an agentLy's ACC contract that ineffective agentLys should accept

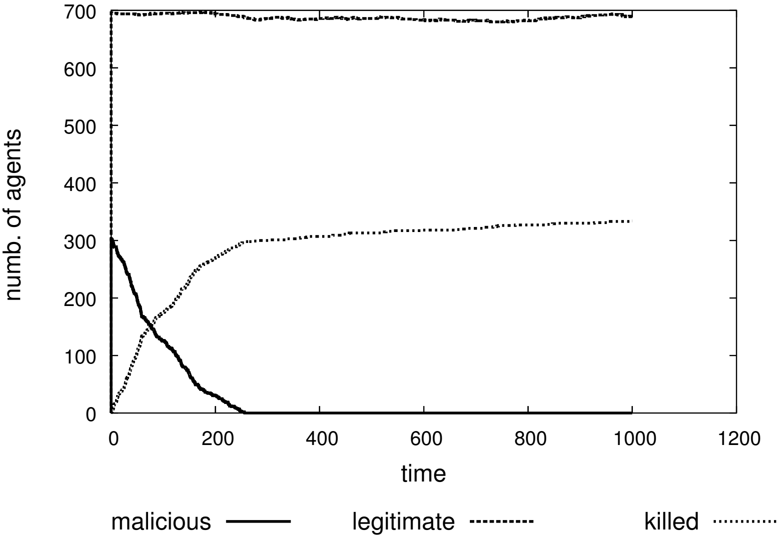

Fig. 5. A simulation of a simple system where malicious and legitimate agents can enter and leave. AgentLys limit the activity of malicious agents. An agentLyA perform better and has a longer lifetime than an agentLyB.

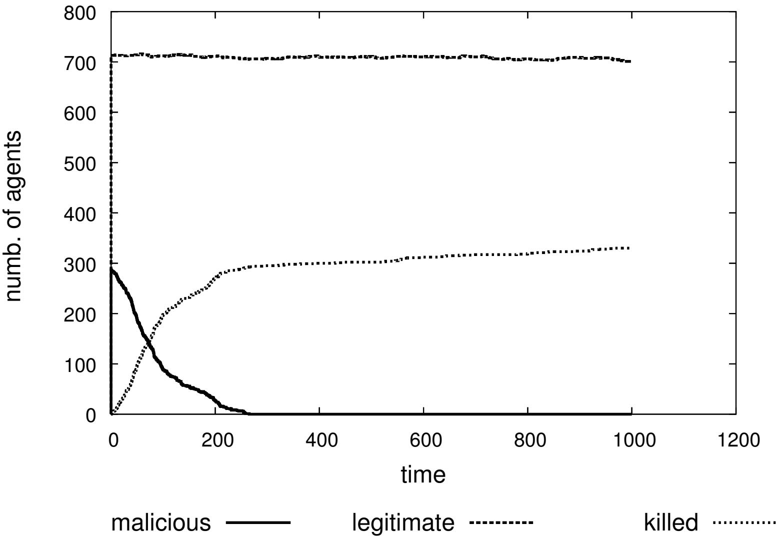

Fig. 6. A simulation of a simple system where malicious and legitimate agents can enter and leave. An agentLyA perform better and has a longer lifetime than an agentLyB. As agentLyA number increases it becomes more probable that a new agentLy will belong to class A.

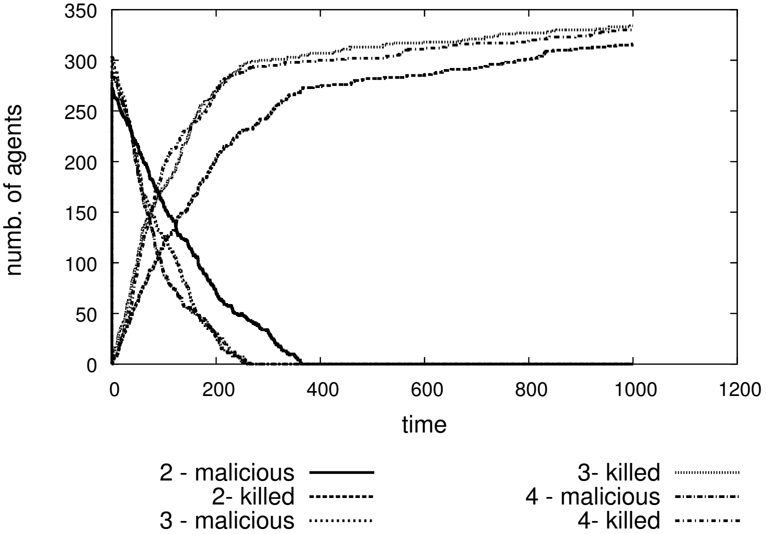

Fig. 7. A comparison in detection performance between the previous simulations

from the infrastructure upgrades on their behaviours—which the infrastructure takes from effective ones.

Figure 6 shows the results of the simulation. With the chosen values for the parameters the system is able to further limit the activity of malicious agents—within still remains similar to the previous case.

4.4 Results Comparison

In Figure 7, we merged the previous charts to evaluate the increase of detection performance. As we would expect there is an evident increase of performance between the first and second scenario, while a minor increase occurs between the second and the third. In general, while the third solution appears more promising, the actual results are strictly dependent from the value of parameters, hence more experiments are needed in order to evaluate whether the possible gains worth the considerable infrastructural support required to simulate lymphocyte learning. In fact, if the two classes of agents have a similar detection rate, the third solution might add only overhead to the system.

5 Conclusion

In this paper we have started putting together the elements of a framework for engineering self-organising applications: featuring MASs composed by agents and artifacts [14, 15], and simulations in a stochastic process algebra settings to tune system parameters at design-time. We used an intrusion detection system for TuCSoN as a mere explanatory case—as more comprehensive ones have already been explored (see e.g.[24, 12]).

The system depicted being 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. For the methodology, we relied to 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 the experiments we realised are still preliminary, we believe they generally emphasise the ability of the proposed approach to help the MAS developer to anticipate design decisions and strategies at the early stages of design—before actually developing prototypes and testing them.

Our plan for future works includes exploiting our approach to devise an actual implementation of intrusion detection systems on top of TuCSoN-based MASs. Other than evaluating the actual strategy to implement (as well as its distinctive parameters), we plan to explore the implications of extending such an approach to a network of nodes. In this paper we have only been concerned with self-organisation mechanisms: in future works we will explore the dynamics that causes new phenomena and behaviours 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 is an emergent property which could be very important for distributed system. Integrating agent cognition and stochastic simulation models is a longer-term research direction as well.

References

1. Horn, P.: Autonomic computing: IBM's perspective on the state of information technology (2001)
2. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (2003) 41–50
3. Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Thomas F. Knight, J., Nagpal, R., Rauch, E., Sussman, G.J., Weiss, R.: Amorphous computing. Communications of the ACM **43**(5) (2000) 74–82
4. Zambonelli, F., Gleizes, M.P., Mamei, M., Tolksdorf, R.: Spray computers: Frontiers of self-organization for pervasive computing. In: 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'04), Washington, DC, USA, IEEE Computer Society (2004) 403–408
5. Heylighen, F.: The science of self-organization and adaptivity. In: Knowledge Management, Organizational Intelligence and Learning, and Complexity. The Encyclopedia of Life Support Systems. EOLSS Publishers (2003)
6. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, part I/II. Information and Computation **100**(1) (1992)
7. Petri, C.A.: Kommunikation mit Automaten. PhD thesis, Institut für Instrumentelle Mathematik, University of Bonn, Bonn, Germany (1962)
8. Bryans, J., Bowman, H., Derrick, J.: Model checking stochastic automata. ACM Trans. Comput. Logic **4**(4) (2003) 452–492
9. Priami, C.: Stochastic pi-calculus. Computer Journal **38**(7) (1995) 578–589

10. Phillips, A.: The stochastic Pi machine (SPiM) (2005) http://www.doc.ic.ac.uk/~anp/spim/.
11. Phillips, A., Cardelli, L.: Simulating biological systems in the stochastic pi-calculus (2004)
12. Forrest, S., Hofmeyr, S.A., Somayaji, A.: Computer immunology. Communications of the ACM **40**(10) (1997) 88–96
13. Omicini, A., Zambonelli, F.: Coordination for internet application development. Autonomous Agents and Multi-Agent Systems **2**(3) (1999) 251–269
14. Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., Tummolini, L.: Coordination artifacts: Environment-based coordination for intelligent agents. In Jennings, N.R., Sierra, C., Sonenberg, L., Tambe, M., eds.: 3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004). Volume 1, New York, USA, ACM (2004) 286–293
15. Omicini, A., Ricci, A., Viroli, M.: RBAC for organisation and security in an agent coordination infrastructure. Electronic Notes in Theoretical Computer Science **128**(5) (2005) 65–85 2nd International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo'04), 30 August 2004. Proceedings.
16. Somayaji, A., Hofmeyr, S., Forrest, S.: Principles of a computer immune system. In: 1997 Workshop on New Security Paradigms (NSPW '97), New York, NY, USA, ACM Press (1997) 75–82
17. Debar, H., Marc, D., Andreas, W.: Towards a taxonomy of intrusion-detection systems. Computer Networks: The International Journal of Computer and Telecommunications Networking **31**(9) (1999) 805–822
18. Forrest, S., Hofmeyr, S.A., Anil, S., A., L.T.: A sense of self for Unix processes. In: 1996 IEEE Symposium on Security and Privacy, Los Alamitos, CA, USA, IEEE Computer Society (1996) 120–128
19. Omicini, A.: Towards a notion of agent coordination context. In Marinescu, D.C., Lee, C., eds.: Process Coordination and Ubiquitous Computing. CRC Press (2002) 187–200
20. Hofmeyr, S.A., Forrest, S.: Immunity by design: an artificial immune system. In: Genetic and Evolutionary Computation Conference (GECCO'99). (1999) 1289–1296
21. Milner, R.: The polyadic π-calculus: a tutorial. In Hamer, F., W., B., Schwichtenberg, H., eds.: International Summer School on Logic Algebra of Specification, Springer (1993)
22. Brinksma, E., Hermanns, H.: Process algebra and Markov chains. In: Lectures on formal methods and performance analysis: 1st EEF/Euro Summer School on Trends in Computer Science. Springer (2002) 183–231
23. Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry **81**(25) (1977)
24. Hassas, S., Foukia, N.: Towards self-organizing computer networks: A complex system perspective. In Di Marzo Serugendo, G., Karageorgos, A., Rana, O.F., Zambonelli, F., eds.: 1st International Workshop on Engineering Self-Organising Applications (ESOA 2003). (2003) 77–83