Biological Immunity and Software Resilience: Two Faces of the Same Coin?

Marco Autili, Amleto Di Salle, Francesco Gallo(✉), Alexander Perucci, and Massimo Tivoli

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università dell'Aquila, L'Aquila, Italy
{marco.autili,amleto.disalle,francesco.gallo,massimo.tivoli}@univaq.it, alexander.perucci@graduate.univaq.it

Abstract. Biological systems modeling and simulation is an important stream of research for both biologists and computer scientists. On the one hand, biologists ask for systemic approaches to model biological systems to the purpose of simulating them on a computer and predicting their behavior, which is resilient by nature. This would limit as much as possible the number of experiments in laboratory, which are known to be expensive, often impracticable, hardly reproducible, and slow. On the other hand, beyond facing the development challenges related to the achievement of the resilience to be offered by biological system simulators, computer scientists ask for a well-established engineering methodology to systematically deal with the peculiarities of software resilient systems, in their more general sense. In line with this, in this paper we report on our preliminary study of immune systems (a particular kind of biological systems) aimed at devising software abstractions that enable the systematic modeling of resilient systems and their automated treatment. We propose a bio-inspired concept architecture for structuring resilient systems based on the Akka implementation of the widely-known Actor Model, which supports scalable and resilient concurrent computation. To the best of our knowledge, this work represents a first preliminary step towards devising a bio-inspired paradigm for engineering the development of resilient software systems.

1 Introduction

Biological systems modeling and simulation is an important stream of research for both biologists and computer scientists. As stated by Hofmeyr [16], the importance of adopting a systemic approach to biology is not new. Its relevance in the modern biological context comes from the need of robust mathematical models and computer simulations that faithfully predict the behaviour of entire biological systems, which are resilient by nature. In this direction, the construction of predictive models of bio-molecular networks is of paramount importance.

Based on differential equations, cellular networks of moderate size have been modeled successfully. However, when large-scale networks are concerned, the

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 1–15, 2015.
DOI: 10.1007/978-3-319-23129-7_1

construction of predictive quantitative models is not easy, if not impossible, due to limited knowledge of mechanistic details and kinetic parameters. Indeed, the knowledge about these systems is typically available in the form of a textual description plus some informal diagrams, which often lead to ambiguities.

Furthermore, as noted by Sackmann et al. [27], the amount of biological knowledge is increasing, and experiments in laboratory tend to be expensive, slow, and often impracticable. As a result, the assistance of computers is becoming indispensable. Computer-assisted experiments could be less expensive, faster and more easily reproducible: "*executable software models of biological systems can be used for predictions, preparation and elimination of unnecessary, dangerous or unethical laboratory experiments*" [21]. As discussed in Sect. 6, one of the major questions systems biology is currently trying to answer is how to represent biological knowledge in a machine-processable way.

On the other side of the coin, beyond facing the development challenges related to the achievement of the resilience to be offered by biological system simulators, computer scientists ask for a well-established engineering methodology to systematically deal with the peculiarities of software resilient systems, in their more general sense [18]. By observing a strong analogy with biological systems, software engineering approaches in the literature (see Sect. 6) achieve resilience by means of mechanisms, e.g., replication, containment, isolation and delegation, which ensure that parts of the system can fail and recover without compromising the system as a whole. For example, recovery of components can be delegated to another (external) component and high-availability is ensured by replication where necessary. Thus, just like biological systems, software resilient systems are more flexible, loosely-coupled, scalable, and more amenable to change. They are significantly more tolerant of failure with respect to non-resilient systems. Still confirming the conceptual relation between biological systems and software resilient systems, biological immunity is related to the ability of an organism to resist a particular infection or toxin by the action of specific antibodies or sensitized white blood cells. This ability recalls the concept of software resilience, i.e., *the ability of a system to persistently deliver its services in a dependable way even when facing changes, unforeseen failures and intrusions*.

The work proposed in this paper starts from the observation that biological immunity and software resilience may be considered as two faces of the same coin. As a particular kind of biological systems, immune systems are resilient systems par excellence. Thus, while architecting resilient software systems, it does make sense to be inspired by the fundamental elements, relations, and behaviors of immune systems.

In line with the above, in this paper we report our preliminary study of immune systems aimed at devising software abstractions that enable systematic modeling and automated treatment of resilient software systems. We propose a bio-inspired concept architecture for structuring resilient systems based on the Akka[1] implementation of the widely-known Actor Model [15], which supports scalable and resilient concurrent computation. To achieve this, we have devised

[1] http://akka.io/.

an abstraction of immune system elements that are then mapped to concrete concepts of the Akka Actor Model. To the best of our knowledge, this work represents a first preliminary step towards devising a bio-inspired paradigm for engineering the development of resilient software systems.

The paper is organized as follows. In Sect. 2, we set the context of our work by summarizing the fundamentals of immune systems and introducing the devised abstractions. Leveraging these abstractions, in Sect. 3 we briefly describe three immune system scenarios that are representative with respect to the resilience concept. Basing on the Akka Actor Model introduced in Sect. 4, in Sect. 5, we propose a bio-inspired concept architecture for resilient software systems and we apply it to the described scenarios. Section 6 discusses related work, and Sect. 7 concludes and outlines future research directions that we will undertake to extend and put in practice the proposed concept architecture, and to precisely define the bio-inspired paradigm on top of it.

2 Immune Systems

An immune system [16] is a particular kind of biological system that is self-protecting against diseases. It is made of biological structures and processes within an organism. The minimum biological structure within an immune system is the cell, which in turn is made of molecules.

A key feature of an immune system is the ability to distinguish between (i) non-infectious structures, which must be preserved since they do not represent a disease, and (ii) infectious structures, i.e., *pathogens*, which must be removed since they result in injuries to the organism the immune system belongs to.

The discrimination between non-infectious and infectious structure takes place at the molecular level and is mediated by specific cell structures that enable the presentation and recognition of harmful components referred to as *antigens* (i.e., small fragments of a pathogen). In particular, these cell structures are able to detect anomalous/undesired situations through antigens recognition, and to place the immune system in a state of alarm. From this state, other cell structures are in charge of reacting with a defensive response, hence removing infectious structures.

By referring to [19], the main elements of an immune system can be summarized as follows:

- **Lymphocytes.** They are the cells of an immune system. For the purposes of this paper, it is enough to distinguish between *T Cells* and *B Cells*:
  - **T Cells.** They can be of two kinds, *T Helper* and *T Killer*. The former are cells responsible for preventing infections by managing and strengthening the immune responses enabled by the recognition of antigens. The latter are cells able to destroy certain tumor cells, viral-infected cells, and parasites. Furthermore, they are responsible for down-regulating immune responses, when needed.
  - **B Cells.** They are responsible for producing *antibodies* in response to foreign proteins of bacteria, viruses, and tumor cells.

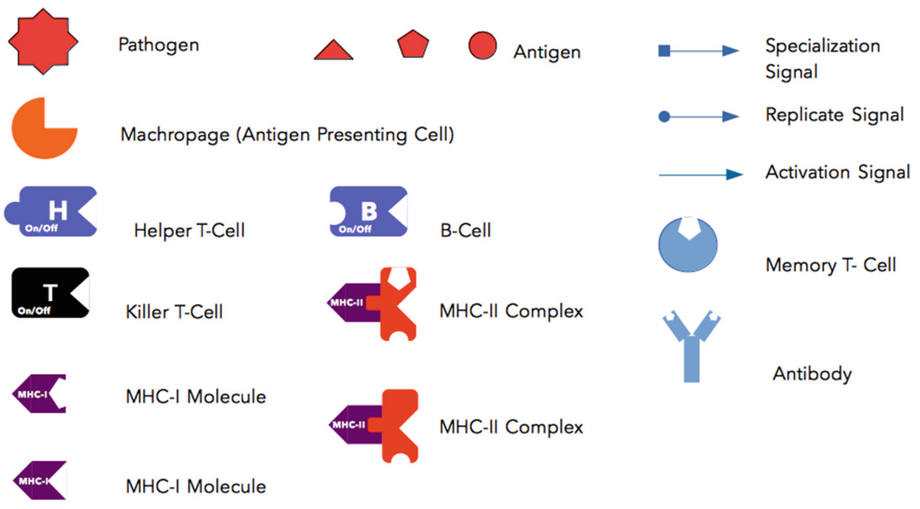

Fig. 1. Immune system components

Both B Cells and T Cells carry receptor molecules that make them able to recognize specific pathogens. In particular, T Killers recognize pathogens only after antigens have been processed and presented in combination with a *Major Histocompatibility Complex* (MHC) molecule. In contrast, B Cells recognize pathogens without any need for antigen processing.

– **Macrophages.** They are important in the regulation of immune responses. They are often referred to as *Antigen-Presenting Cells* (APC) because they pick up and ingest foreign materials and present these antigens to other cells of the immune system such as T Cells and B Cells. This is one of the important first steps in the initiation of an immune response.
– **Memory Cells.** When B Cell and T Cell are activated and replicated, some cells belonging to their progeny become long-lived *Memory Cells*. The role of Memory Cells is to build an immunological memory that makes the immune system stronger in being self-protecting to future infections/attacks. In particular, a Memory Cell remembers already recognized antigens and lead to a stronger immune response when these antigens are recognized again.
– **Immune response.** An immune response to foreign antigens requires the presence of APC in combination with B Cells or T Cells. When an APC presents an antigen to a B Cell, the B Cell produces antibodies that specifically bind to that antigen in order to kill/destroy it. If the APC presents an antigen to a T Cell, the T Cell becomes active. Active T Cells essentially proliferate and kill target cells that specifically express the antigen presented by the APC. The production of antibodies and the activity of T Killers are highly regulated by T Helpers. They send *signals* to T Killers in order to regulate their activation, proliferation (replication) and efficiency (specialization).

Following the description above, Fig. 1 shows the main elements of an immune system as constructs of a simple graphical notation that we use in Sect. 3 for immune system modeling purposes. The figure shows also the messages (signals) that the system elements can exchange.

3 Immune System Scenarios

By leveraging the graphical modeling notation introduced above, in this section, we briefly describe two scenarios in the immune systems domain, which are representative for a broad class of resilient systems. The scenarios provide the reader with a high-level description of the interactions that happen among the elements of an immune system during an immune response.

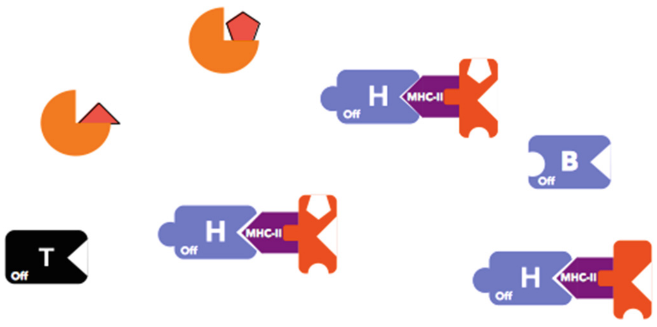

Fig. 2. Immune system - Scenario 0

Scenario 0. Figure 2 shows a scenario where the elements of an immune system are in an *inactive* state, marked with the *off* label. In particular, two APC engulf a virus or bacteria: each APC decomposes the pathogen (virus or bacteria) and exposes on its surface a piece of the pathogen, i.e., an antigen (see the triangle and pentagon in the figure). Metaphorically, we can think of this as a setup phase of a computer system, where each system's component is in an idle state and the antigen is a "perturbation" that comes from the outside or even by the system itself. In our context, we can see this perturbation as either a new or an anomalous, undesired, system behavior.

Scenario 1a and 1b. Continuing Scenario 0, the presence of an antigen causes the activation of one or more cells of the immune system (left-hand side of Fig. 3). In this case, the antigen is caught only by those cells that are able to treat it; so they switch from the *off* state to the *on* state. By referring to the right-hand side of Fig. 3, T Helpers recognize specific antigens and replicate themselves (replicate signals); T Helpers make T Killers and APC active (by sending to them an activation signal); T Killers replicate themselves. Some T Helpers specialize into Memory Cells (specialization signal). This scenario highlights how an immune response is a distributed and decentralized process.

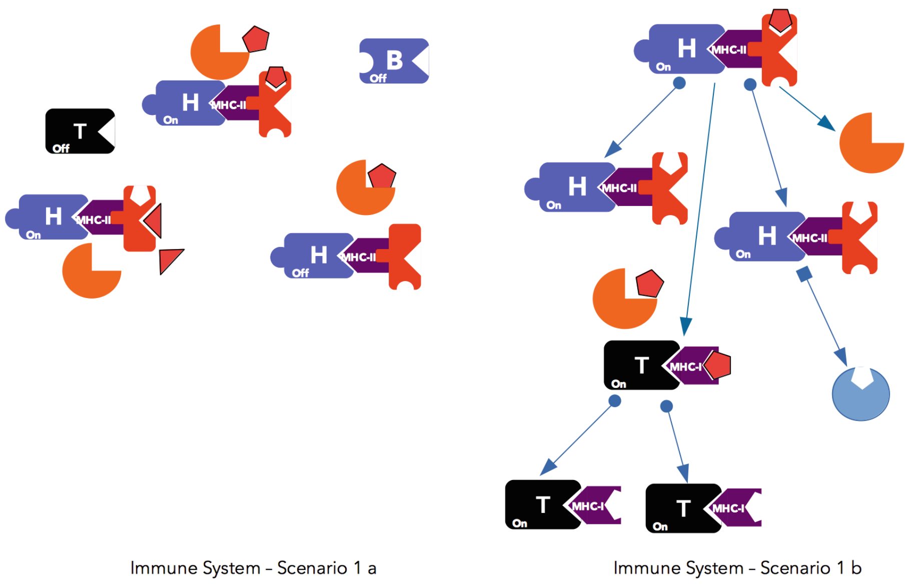

Fig. 3. Immune system - Activation scenarios

Scenario 1c. In this scenario, a B Cell becomes active after that it caught a known antigen and the T Helper close to it became active. This scenario points out that a B Cell has less constraints to satisfy with respect to T Cells, i.e., a B Cell does not need to interface with MHC molecules (Fig. 4).

These simple, yet representative scenarios, lead us to observe that, as a particular kind of resilient system in a specific domain:

- an immune system is composed of loosely-coupled elements, hence promoting modularity;
- the elements of an immune system can interact with each other by exchanging asynchronous messages, hence enhancing reliability;
- an immune system can react to unforeseen events, e.g., intrusions, being therefore fault tolerance;
- an immune system can foresee, e.g., dangerous situations through preventive recognition, hence achieving robustness;
- the elements of an immune system have the ability to adapt to context changes and recover from undesired situations, therefore showing flexibility and evolvability.

4 Actor Model

The Actor Model is a formal mathematical model of concurrent computation that was first proposed by C. Hewitt et al. in 1973 [15]. Over the years, this model has seen several programming languages employing the notion of actor,

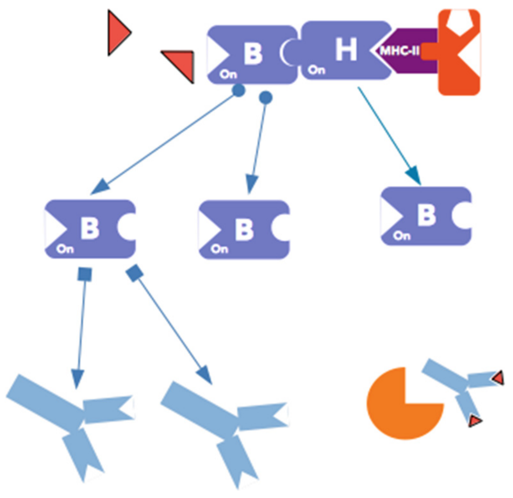

Fig. 4. Immune system - Scenario 1c

most notably Erlang [2] (a concurrent programming language designed for programming fault-tolerant distributed systems) and Scala [13] (which offers a concurrent programming model based on the Scala standard library dedicated to Actors).

For the purpose of this paper, we focus on the Akka toolkit[2], which is a framework that natively permits a programming style based on the Actor Model. Akka is written in Java and Scala, and offers an unified runtime and programming model that allows for building highly concurrent, distributed, and resilient message-driven applications.

In this section, after introducing the Actor Model, we present the Akka framework (Sect. 4.1), whose elements are then used in Sect. 5 to present our Akka-based Concept Architecture.

The Actor Model is characterized by (i) inherent concurrency of computation within and among Actors, (ii) dynamic creation/replication of Actors, (iii) inclusion of Actor addresses in messages, and (iv) interaction only through direct asynchronous message passing.

The Akka Actor Model achieve resilience by offering software abstractions that support the systematic development of flexible, loosely-coupled and scalable applications, which are significantly more tolerant of failure and stay responsive in the face of failure. Resilience is achieved through replication, containment, isolation and delegation mechanisms. Specifically, Akka components are isolated from each other, thereby ensuring failures isolation within the affected component only; the failing parts of the system have the possibility to recover without compromising the system as a whole; recovery actions are delegated to other

[2] http://akka.io.

(supervisor) components and, when needed, components replication is used to achieve high-availability.

4.1 Akka Actor Model

Actors are objects that encapsulate state and behavior, and communicate through message passing. Akka allows for modeling applications in terms of interacting actors that can be dynamically assigned sub-tasks, and arranging related functions into an organizational/hierarchical structure while thinking about how to tolerate/escalate failures.

An actor has its own state and can be seen as a container for Behavior, a Mailbox, Children and a Supervisor Strategy. Actors have a hierarchical structure and each actor has a supervisor, with the root supervisor being the `SystemActor`. In more details:

Actor Reference: actor references are used to represent actors to the outside. References enables transparency in the sense that an actor can be, e.g., restarted without needing to update references elsewhere, or seamlessly moved on remote hosts.

Behavior: messages are matched against the current behavior of the actor, i.e., to functions which defines the actions to be taken in reaction to the message. Importantly in our setting, the behavior of an actor can be changed dynamically, e.g., to allow the actor to come back to work after an "out-of-service" state is reached.

Mailbox: the mailbox connects the sender and receiver actor, and allows for enqueuing the exchanged messages in order to support asynchronous communication.

Children: an actor is potentially a supervisor and can create children for delegating sub-tasks. The creation and termination actions are not blocking (i.e., they happen behind the scenes in an asynchronous way).

Supervisor Strategy: a supervisor actor has strategy for handling faults of its children. For our purpose, the important aspect here is that fault handling is done transparently by the Akka framework, by exploiting monitors and by applying the available supervision strategies. Moreover, another crucial aspect towards achieving resilience is that strategies can be updated/added dynamically.

5 Akka-Based Concept Architecture

Figure 5 shows the bio-inspired concept architecture we are working on. It represents the starting point to support the bio-inspired paradigm that we have in mind for engineering the development of resilient software systems. Indeed, by referring to the scenarios 0, 1a, and 1b described in Sect. 3, we show only those elements that are strictly needed to provide the reader with intuitions on how to put together the elements of the concept architecture into a logically coherent argumentation.

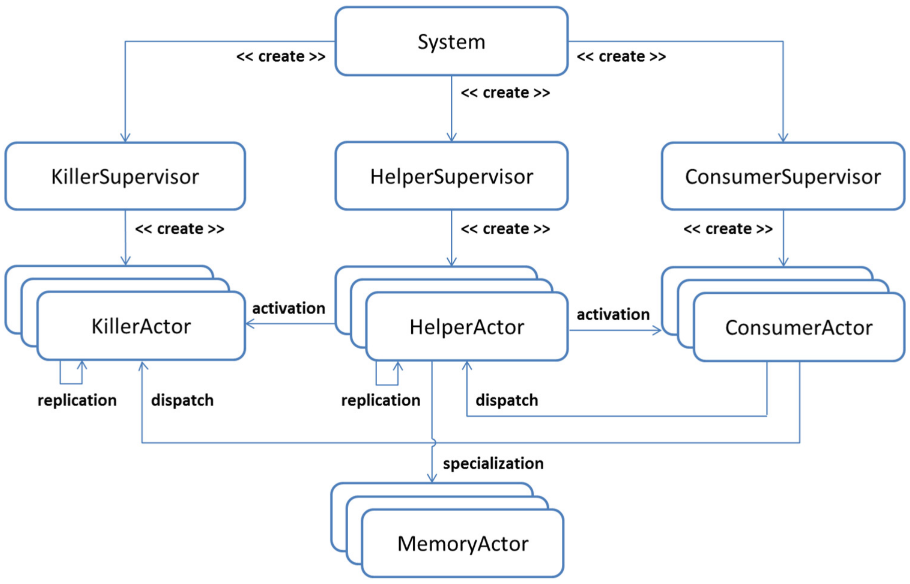

Fig. 5. Bio-inspired concept architecture for resilient systems

According to the Akka framework, the `System` actor creates the `KillerSupervisor`, `HelperSupervisor` and `ConsumerSupervisor` *supervisor* actors. By taking inspiration from immune systems (Sect. 2) and related scenarios (Sect. 3), the supervisor actors are in charge of detecting changes, intrusions, failures, and undesired behaviors. For instance, referring to systems where security is a crucial dependability requirement, the addition of a new component that behaves as a trojan, or the presence of a component that misbehaves. Another example, when referring to systems where performance is crucial, would be the presence of a component that does not perform as expected anymore.

Furthermore, supervisor actors are in charge of creating respective sub-actors, namely `KillerActor`, `HelperActor` and `ConsumerActor`, which are responsible for, e.g., self-protecting/self-reconfiguring the system from the detected undesired behavior/change by putting in place a resolutive response. This is done through recognition of the kind of problem and production/replication of those set of actors that can realize a suitable solution. E.g., removing, isolating, disabling the introduced trojan or the misbehaving component; replacing the component that badly perform with a new version of it that performs as expected.

In particular, by mimicking what happens in scenarios 0, 1a and 1b (Fig. 3) when the presence of an antigen causes the activation of one or more cells of the immune system and the antigen is caught only by those cells that are able to treat it, in our concept architecture, the occurrence of the problem/change is signalled by means of *dispatch messages*. Each message activates specific instances of `HelperActor` and `KillerActor`. The former play the role of resolutive response managers. After activation, a `HelperActor` instance aims at (i) inducing replication of `HelperActor` instances by sending *replication* messages; (ii) activating `KillerActor` and `ConsumerActor` instances by sending *activation* messages; and (iii) specializing some actor instances in `MemoryActor` instances by sending *specialization* messages. `KillerActor`s are actuators that consume messages and actually put in place the problem resolution or reconfiguration strategy that realizes the triggered response.

Considering resilience attributes of immune systems [3] and the support to resilience offered by the Akka Actor Model (Sect. 4), our concept architecture applied to the scenarios 0, 1a, and 1b is able to meet the following software resilience attributes:

– **agility**

  −*immune systems*: they have multiple barriers or layers of defense to prevent a pathogen from causing harm.
  −*software systems*: system undesired behaviors or changes represent the software counterpart of pathogens and our Actor Model implementation compartmentalizes the, e.g., undesired behavior, and allows its resolution locally without compromising the operation of the system as a whole.

– **redundancy**

  −*immune systems*: having the ability to replicate antibodies increases the probability that a pathogen that matches these antibodies will be stopped. Furthermore, generated antibodies keep memory of the already matched pathogens, hence strengthening the ability to stop it.
  −*software systems*: Akka toolkit offers persistence that enables actors to persist their internal state so that it can be recovered when a produced or replicated actor is started, restarted after a JVM crash or by a supervisor, or migrated in a cluster.

– **dynamic learning**

  −*immune systems*: as a redundancy enabler, based on previous knowledge about already recognized antigens, the immune system is able to learn new disturbances and related resolutive responses.
  −*software systems*: if some change or undesired behavior occurs at run time, and the affected actor is not able to manage it, the actor initially treats it as an unknown message. As a such, the message is passed to its supervisor. Based on historical knowledge, the supervisor is able to learn if there are other actors able to deal with the change or undesired behavior. If it is the case, the related resolutive logic is dynamically injected into the affected actor through the `PartialFunction` mechanism offered by Akka. Clearly, different policies can be applied when the supervisor has not been able to learn possible solutions, e.g., killing the affected actor, or requiring human intervention.

– **flexibility**

  −*immune systems*: antibodies are produced and added depending on the need, without any redesign.
  −*software systems*: the flexibility concept is native in Akka since actors can be dynamically produced or replicated, without blocking the system.

– **robustness**

  −*immune systems*: leveraging Apoptosis [23] or "programmed cell death" as a mechanism for deletion of "unwanted" cells, an immune system has the ability to keep working even when multiple cells are killed since not properly working anymore.
  −*software systems*: in the Actor Model, robustness is a form of fault tolerance. It uses the "let it crash" policy to manage the programmed death of faulty components that can be dynamically killed or stopped for preserving the system functioning, if possible, or at least for preventing dangerous system's misbehaviours.

6 Related Work

In this section, we discuss related work in the areas of (i) biological systems modeling and simulation and (ii) software resilient systems.

Biological systems modeling and simulation. Current approaches to biological systems modeling and simulation can be organized into three classes, namely *quantitative*, *qualitative*, and *rule-based* approaches.

Quantitative approaches [12,29]. They make use of differential equations and stochastic simulation to model biological processes. They essentially suffer two main issues. On the one hand they do not scale in the heterogeneity of constituent elements of the biological system, hence preventing their applicability to biological processes with many species and variables. On the other hand, as complex mathematical models, they are hard to be exploited and difficult to understand by end-users, not only biologists but also software engineers.

Qualitative approaches [21,27]. They are primarily based on the biological system's network structure and, differently from quantitative approaches, do not require knowledge about internal parameters of the system, e.g., kinetic parameters. Rather, models can be produced to abstract different views of a biological process hence allowing to reason at different biological organization levels, e.g., sub-cellular, cellular, tissue, organ, organism and ecosystem. The focus, here, is to identify how different components are connected together, how they are controlled and how they behave when functioning as a system. However, for these approaches to be effective and profitable, two main aspects must be ensured: (i) the model representation language must be rich enough to represent the various heterogeneous system's elements and to capture all the system behaviours at the different organization levels; and (ii) the system identification technique must be powerful enough to identify substantially complex models, which can enable realistic simulation. The class of qualitative approaches comprises various formalisms that span from Boolean Network models [30] to constraint-based models [26], to Petri Nets [4,14], to logical models [20,24]. In addition to static analysis of the structural properties of a biological system's network topology, Petri Nets and logical models enable to reason on the systems behaviour by means of discrete dynamic modeling [1,8,24,30].

Rule-based approaches. They promote the production of rule-based models by using specialized languages such as BioNetGen language [9] or Kappa [6]. Being more reusable than equations, rule-based models allow for enhancing modularity, hence making tractable the modeling and analysis [11] of complex biological systems involving several different species [7]. Rules can also be used to generate simulations, both deterministic and agent-based [28].

Software resilient systems. The work in [17] proposes an approach to automatically detect faults of a redundant duplex system by using a model checker. The proposed method analyzes vulnerabilities of different variants of an algorithm by applying fault injection modelling. Relying on the Event-B model, in [22], the authors present a formal approach to model and assessing reconfigurable systems that guarantees resilience of data processing. The proposed system architecture is able to dynamically scale and reconfigure.

Natural and Biological systems, such as Ant colonies and Immune systems, have several features that can be exploited in designing and developing resilient systems. More precisely, these super organisms often use self-organizing behaviors and feedback loops [5] that allow the system to achieve reliable and robust solutions using information gathered from entities [25], without centralized control. The biological immune system can be seen as a massively distributed architecture: the multitude of independent cells work together resulting in the emergent behavior of the immune system. The immune system evolves to adapt and improve the overall system performance (e.g., organizational memory) [10,31]. These systems can be seen as complex collective systems in which the behaviour emerges from the product of interactions between individual entities. These entities follow a simple set of rules (i.e., not via top-down mechanism) and react only to their local environment. Features and principles as bottom-up mechanisms, feedback loops could be used for designing a scalable, adaptive and efficient framework.

7 Conclusions and Future Work

In this paper we proposed a bio-inspired concept architecture for resilient software systems based on the Akka Actor Model. Our proposal originates from the observation that immune systems natively enjoy resilience properties that have a direct counterpart in software resilient systems. Our long term goal is to reach a mature enough knowledge about the key concepts that are common to both immune systems and resilient systems. By transposing (i) the elements of immune systems, (ii) their self-protecting behavior, and (iii) their relationships/interaction with other elements into their respective software design principles, this knowledge would allow us to elevate our preliminary concept architecture to a rigorously defined bio-inspired architectural style for resilient systems. This architectural style would constitute a common ground on top of which building the novel biological development paradigm we have in mind for software resilient systems.

To reach this goal, several challenges have to be faced. They are related to the definition and realization of general-purpose mechanisms that, being amenable of domain-specific customizations, support features such as:

- *automatic recognition* of software failures/changes through, e.g., run-time monitoring, feedback loops;
- *dynamic learning* of the solutions required to correctly react to the recognized failures/changes based on, e.g., dynamically acquired historical knowledge;
- *modular actuation* of the (learned) solution, without compromising the overall system function;
- *opportunistic selection* of those available solutions that better fit some predefined policy, e.g., to guarantee specified non-functional requirements;
- *self-stabilization* of the self-* actions, e.g., self-adaptation, self-reconfiguration, to guarantee the system equilibrium with respect to some specified invariant properties, despite continuous applications of self-* actions;
- *multilayer management* of failures/changes (and related strategies) in a modular, yet cohesive, way depending on the affected layer(s), e.g., application, middleware, operating system, network layer.

References

1. Bio-pepa: A framework for the modelling and analysis of biological systems. Theoretical Computer Science, 410(33–34), 3065–3084 (2009)
2. Armstrong, J.: Erlang. Commun. ACM **53**(9), 68–75 (2010)
3. Chandra, A.: Synergy between biology and systems resilience, master's thesis, missouri university of science and technology (2010)
4. Chaouiya, C.: Petri net modelling of biological networks. Briefings Bioinform. **8**(4), 210–219 (2007)
5. Cheng, B.H.C., et al.: Software engineering for self-adaptive systems: a research roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009)
6. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-Based modelling, symmetries, refinements. In: Fisher, J. (ed.) FMSB 2008. LNCS (LNBI), vol. 5054, pp. 103–122. Springer, Heidelberg (2008)
7. Deeds, E.J., Krivine, J., Feret, J., Danos, V., Fontana, W.: Combinatorial complexity and compositional drift in protein interaction networks. PLoS ONE **7**(3), 03 (2012)

8. Dematté, L., Priami, C., Romanel, A.: The blenx language: a tutorial. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 313–365. Springer, Heidelberg (2008)
9. Faeder, J., Blinov, M., Hlavacek, W.: Rule-based modeling of biochemical systems with bionetgen. In: Maly, I.V. (ed.) Systems Biology, volume 500 of Methods in Molecular Biology, pp. 113–167. Humana Press (2009)
10. Farmer, J., Packard, N.H., Perelson, A.S.: The immune system, adaptation, and machine learning. Physica D **22**(13), 187–204 (1986)
11. Feret, J., Danos, V., Krivine, J., Harmer, R., Fontana, W.: Internal coarse-graining of molecular systems. Proc. Nat. Acad. Sci. **106**(16), 6453–6458 (2009)
12. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2361 (1977)
13. Haller, P., Odersky, M.: Scala actors: unifying thread-based and event-based programming. Theoret. Comput. Sci. **410**(2–3), 202–220 (2009)
14. Hardy, S., Robillard, P.N.: Pn: Modeling and simulation of molecular biology systems using petri nets: modeling goals of various approaches. J. Bioinform. Comput. Biol. **2–4**, 2004 (2004)
15. Hewitt, C., Bishop, P., Steiger, R.: A universal modular ACTOR formalism for artificial intelligence. In: Proceedings of the 3rd International Joint Conference on Artificial Intelligence. pp. 235–245. Standford, CA, August 1973
16. Hofmeyr, S.A.: An interpretative introduction to the immune system. In: Design Principles for the Immune System and Other Distributed Autonomous Systems, pp. 3–26. Oxford University Press (2000)
17. Höller, A., Kajtazovic, N., Preschern, C., Kreiner, C.: Formal fault tolerance analysis of algorithms for redundant systems in early design stages. In: Majzik, I., Vieira, M. (eds.) SERENE 2014. LNCS, vol. 8785, pp. 71–85. Springer, Heidelberg (2014)
18. Majzik, I., Vieira, M. (eds.): SERENE 2014. LNCS, vol. 8785. Springer, Heidelberg (2014)
19. Janeway Jr., C., Travers, P., Walport, M., et al.: Immunobiology: The Immune System in Health and Disease, 5th edn. Garland Science, USA (2013)
20. Klamt, S., Saez-Rodriguez, J., Lindquist, J.A., Simeoni, L., Gilles, E.D.: A methodology for the structural and functional analysis of signaling and regulatory networks. BMC Bioinform. **7**, 56 (2006)
21. Krepska, E., Bonzanni, N., Feenstra, A., Fokkink, W.J., Kielmann, T., Bal, H.E., Heringa, J.: Design issues for qualitative modelling of biological cells with petri nets. In: Fisher, J. (ed.) FMSB 2008. LNCS (LNBI), vol. 5054, pp. 48–62. Springer, Heidelberg (2008)
22. Laibinis, L., Klionskiy, D., Troubitsyna, E., Dorokhov, A., Lilius, J., Kupriyanov, M.: Modelling resilience of data processing capabilities of CPS. In: Majzik, I., Vieira, M. (eds.) SERENE 2014. LNCS, vol. 8785, pp. 55–70. Springer, Heidelberg (2014)
23. Lawen, A.: Apoptosisan introduction. BioEssays **25**(9), 888–896 (2003)
24. Morris, M.K., Saez-Rodriguez, J., Sorger, P.K., Lauffenburger, D.A.: Logic-based models for the analysis of cell signaling networks. Biochem. **49**(15), 3216–3224 (2010)
25. Nieh, J.C.: A negative feedback signal that is triggered by peril curbs honey bee recruitment. Curr. Biol. **20**(4), 310–315 (2010)
26. Papin, J.A., Palsson, B.O.: Topological analysis of mass-balanced signaling networks: a framework to obtain network properties including crosstalk. J. Theoret. Biol. **227**(2), 283–297 (2004)

27. Sackmann, A., Heiner, M., Koch, I.: Application of petri net based analysis techniques to signal transduction pathways, BMC Bioinform. **7–482** (2006)
28. Sneddon, M.W., Faeder, J.R., Emonet, T.: Efficient modeling, simulation and coarse-graining of biological complexity with NFsim. Nat. Meth. **8**(2), 177–183 (2011)
29. Srivastavawz, R., Youw, L., Summersy, J., Yin, J.: on stochastic vs. deterministic modeling of intracellular viral kinetics (2002)
30. Wang, R.-S., Saadatpour, A., Albert, R.: Boolean modeling in systems biology: an overview of methodology and applications. Phys. Biol. 9(5) (2012)
31. Watanabe, Y., Ishiguro, A., Shirai, Y., Uchikawa, Y.: Emergent construction of behavior arbitration mechanism based on the immune system. In: The 1998 IEEE International Conference on Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence, pp. 481–486 (1998)