Building iRIS: A Robotic Immune System

Dietmar Schreiner

Vienna University of Technology
Institute of Computer Languages,Compilers and Languages Group
Vienna, Austria
schreiner@complang.tuwien.ac.at

Abstract. Progress in robotics has led to complex autonomous and even collaborating robotic systems, fulfilling mission critical tasks in safety critical environments. An increase in capabilities and thus complexity consequently led to a dramatic increase in possible faults that might manifest in errors. Even worse, by applying robots with emerging behavior in non-deterministic real-world environments, faults may be introduced from external sources. Consequently, fault testing has become increasingly difficult. Both, software and hardware may fail or even break, and hence may cause a mission failure, heavy damage, or even severe injuries and loss of life. The ability of a robotic system to function in presence of such faults, so to become fault tolerant, is a continuously growing area of research. Our work meets this challenge by developing a mechanism for robotic systems that is capable of detecting defects, selecting feasible counter measures, and hence keeping robots in a sane and and consequently safe state. Inspired by biology, we conceptually aim at an immune system for a robot (RIS), which is able to detect anomalies, and which is able to autonomously counter them by appropriate means. This position paper outlines the requirements and research scopes that have been identified as relevant for the development of a robotic immune system.

1 Introduction

Robustness and dependability are key factors for today's as much as upcoming embedded computer systems not only in safety and mission critical domains but also in everyday life. Driven by progress and market the number of devices that operate within the real world and hence may harm people and environment, like autonomous vehicles, computer guided aircrafts, or reconnaissance and rescue robots, is constantly growing. At the same time, the complexity of those devices and applications–devices typically consist of a large number of networked processors, sensors and actuators–has reached a noticeable extent. Complicated tasks have to be fulfilled within a nondeterministic environment under strict consideration of safety guarantees, but also under harsh cost and energy constraints. Features like long-term autonomy and system adaptability have become an additional burden on robotic systems' software. Traditional techniques in software development that typically handle a precisely enumerated set of faults do not

scale well with the aforementioned increasing system complexity and the real world's nondeterminism.

We think that a nature inspired approach similar to the concept of a biological immune system could be able to provide a sound and robust solution for this robotic issue. Assuming that there exists no complete enumeration of faults that should be detected and handled over the lifespan of a system within an indeterministic environment, classical error handling techniques quickly reach their limits in terms of development cost and technical capability. An immune systems can overcome the problem of unknown and unexpected faults by detecting anything that is not a sane system state, and by utilizing adaptable rules on how to (re)establish a sane system. This concept was developed by evolution over millions of years, and is a promising approach for robust autonomous dependable systems within nondeterministic environments.

2 Scopes of Research for a RIS

To achieve the goal of nature inspired fault-tolerant self-healing software research has to be conducted at the intersection of five major scopes like depicted in Figure 1. Affordable devices that fulfill all necessary dependability constraints like safety or robustness [1] can only be built by joining results from all denoted domains: (i) Model Driven Development as a basis for code generation as much as for system verification, (ii) Static Analysis as a methodology to derive system properties from program codes as much as to guide compilation and code generation tasks, (iii) Programming Languages to provide proper means of abstraction and semantically enriched specifications, (iv) Reflective Computing as a methodology for run-time monitoring and supervision, and (v) Nature Inspired Computing as promising way to handle the systems' increasing complexity.

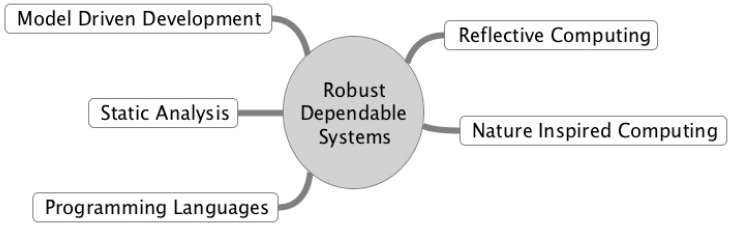

Fig. 1. Research Scopes

2.1 Model Driven Development

The complex nature of robotic applications—aspects like real time constraints as much as issues of concurrency or resource consumption have to be considered at the same time—introduces the need for a sound development methodology that not only provides separation of concerns but also allows the incorporation of domain specific knowledge typically held by the robots' user rather than by programmers.

In model driven development views on the system that is built, the models, are specified at various levels of abstraction. These models are subject to so called model transformations, processes that automatically generate new artifacts like system configurations, new models, or even executable program code. Considering the requirements on robotic software engineering model driven development is a well suited methodology: specific aspects of the system under development are described in distinct views at proper levels of abstraction; domain specific know-how can easily be formalized by non-programmers as yet another view that can be transformed into technical views via automatic model transformations; system as much as application properties can be extracted from models via model transformations.

Our ongoing research within the domain of model driven development deals with automatic generation of synchronization artifacts for concurrent embedded systems applications [2]. Expected outcome is a sound methodology for automatic code injection and code generation for software sensors and reflective components in order to get run-time monitoring support for complex embedded systems as the backbone of an innate immune system.

2.2 Static Analysis

By examining executable code or even the executables' source code, many vital properties of a software system can be calculated before execution. These information can be used to automatically refactor and optimize code, detect various defects, or simply determine sane bounds for a program's execution.

The work relevant for a robotic immune system mainly aims at analyses that identify run-time properties before execution, like worst case loop-bounds, feasible paths, and WCET bounds, as a basis for an immune system's innate knowledge base, as much as at the identification of mount points for software sensors (which will be used later on in model driven injection (see Section 2.1)).

2.3 Programming Languages

Programming languages are the computer systems' interface for software developers. However, most programming languages used for embedded systems today are well suited for process centric or even hardware related software development, but do not meet upcoming requirements like distributed computing, concurrency, or dependability constraints. Additionally, those languages require a high expertise in programming and firm knowledge of the underlying hardware,

which typically excludes experts from the devices' application domain from application development.

Visual programming languages, but also visual modeling techniques–we do consider a program's source code to be a model of the program–are one way to provide abstraction where necessary, and to lower the barrier for application domain experts to create or at least modify their own application. Additionally, complex system constraints can be expressed in a comprehensible way, and can furthermore be used by transformation and compilation techniques to generate subsystems of an embedded immune system. The idea of graphical modeling is well accepted, its application to the automotive domain is for example outlined in [3] where UML 2.0 profiles were used to graphically specify a system's communication requirements.

2.4 Reflective Computing

Reflective computing denotes techniques that provide self-inspection of running systems. Although this idea is not new to computer science, it is part of the research agenda for a RIS. Introspection is an inevitable feature for any immune system, biological as much as artificial. Aiming at the automatic generation of an artificial immune system for a robotic device, software sensors have to be automatically deployed at compile time in order to disencumber application developers.

Research in that domain will hence deal with questions on identifying the proper locations for software sensors (this question goes hand in hand with static analysis) as much as a robust and resource constrained embodiment of those sensors.

2.5 Nature Inspired Computing

Nature has developed remarkable mechanisms for extremely complex tasks by evolution over a very long time. Consequently, those concepts can provide the solution to many open scientific but also engineering issues. However, the concepts have to be understood and simplified, abstracted in terms of a computer scientist, and adapted to specific needs.

The vision outlined in this paper is that of an artificial immune system for autonomous robust embedded devices. This system is inspired by biological immune systems, at least as far as they are understood today, and is custom tailored to the upcoming technical domain of collaborating robotic systems.

3 iRIS: An Overview

In terms of computer science, a biological immune system is a robust, multilayered, distributed system that is able to identify numerous pathogens. Its main responsibility is to counter act harmful effects to keep the organism in a sane state. An artificial immune system is meant to do exactly the same for computers

and in our case robotic systems. Harmful effects can be detected and can be overcome autonomously.

In [4] one bio-inspired algorithm, the negative selection algorithm, is described, which is based on the detection of self from non-self. Dedicated immune cell types like lymphocytes have receptors that allow them to bind to specific proteins. During maturation these cells are 'trained' on proteins that are naturally present in the organism, the self-antigenes. Lymphocytes which spuriously bind to self-antigenes are destroyed immediately (apoptosis). After reaching maturity, the trained cells are spread over the organism. If they bind to a protein now, this is a clear indication of a non-self protein, a pathogen.

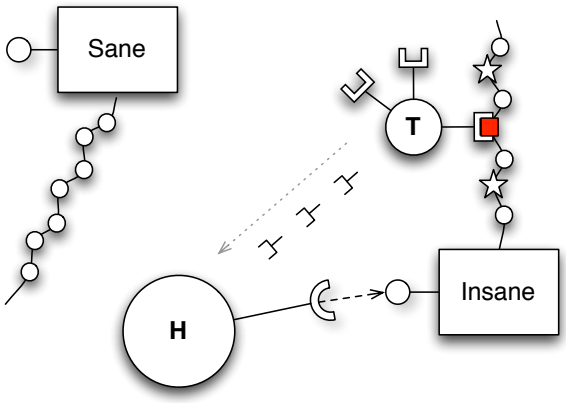

Fig. 2. Immune reaction to faulty component

iRIS (innate Robotic Immune System) is structured in accordance to a biological immune system: Within the organism (the robot), distributed autonomous light-weight processes, so called T-processes, constantly monitor the system's sanity via their receptors, e.g., software-sensors, execution monitors, or even hardware sensors. Antigenes are sequential representations of specific states within the robot, and may be defined and classified at development but also dynamically at run-time. Detection of non-self antigenes is achieved in two ways: (i) T-processes undergo a process of maturation, which means they are trained to detect self-antigenes on a running system. (ii) T-processes utilize 'genetic knowledge', represented by predefined rules and parameters. On detection of a non-self antigene, T-processes activate H-processes, which are in charge of eliminating anomalies (e.g., by restarting malfunctioning components of a robot, reinitializing affected structures, or recalibrating sensors).

Figure 2 depicts the conceptual idea behind an iRIS immune reaction: It shows two components, *Sane* and *Insane*, both exposing their associated antigene. As *Insane* is malfunctioning, the T-process T detects this non-self behavior and notifies specialized H-processes. One H-process finally docs at *Insane's*

maintenance interface to counter-act the fault. In addition, our observations show that typical faults within robotic systems require complex repair actions at distributed subsystems. For that reason, H-processes may emit messenger antigenes to trigger additional repair activities at all affected subsystems.

Compared to existing work, iRIS covers aspects of autonomic computing like summarized in [5]. It does reflective computation [6, 7] at run-time in a bio-inspired way, using knowledge extracted from static analyses and system models at development and compile time. A model driven development methodology as much as static analyses come to use to extract the 'genetic knowledge' of iRIS. In addition, iRIS incorporates dynamic means of machine learning, also inspired by natural immune systems, to cope with unforeseen faults.

4 Conclusion

This paper has outlined the basic idea for an innate robotic immune system (iRIS), which aims at robust dependable robots. In order to build such a system research has to be conducted at least in five domains within computer science: (i) Model Driven Development, (ii) Static Analysis, (iii) Programming Languages, (iv) Reflective Computing, and (v) Nature Inspired Computing. Merging results from these domains will enable the development of an innate immune system capable of self/non-self discrimination, which denotes the baseline for further adaptable systems.

References

1. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. Dependable and Secure Computing, IEEE Transactions on **1**(1) (jan.-march 2004) 11 – 33
2. Schreiner, D., Puntigam, F.: Robots, Software, Mayhem? Towards a Design Methodology for Robotic Software Systems. In: Supplemental Volume of the Eight European Dependable Computing Conference (EDCC 2010). (April 2010) 31–32 ISBN: 978-84-692-9571-7.
3. Schreiner, D., Göschka, K.M.: Modeling Component Based Embedded Systems Applications with Explicit Connectors in UML 2.0. In: Proceedings of the 2007 ACM symposium on Applied computing (SAC '07), New York, NY, USA, ACM Press (March 2007) 1494–1495
4. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. In: Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy. (1994)
5. Dobson, S., Sterritt, R., Nixon, P., Hinchey, M.: Fulfilling the vision of autonomic computing. Computer **43**(1) (2010) 35 –41
6. Maes, P.: Concepts and experiments in computational reflection. In: OOPSLA. (1987) 147–155
7. Rodríguez, M., Fabre, J.C., Arlat, J.: Wrapping real-time systems from temporal logic specifications. In 0002, F.G., Thévenod-Fosse, P., eds.: EDCC. Volume 2485 of Lecture Notes in Computer Science., Springer (2002) 253–270