Expert Systems with Applications 40 (2013) 7267–7279

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Review

Self-adaptive systems: A survey of current approaches, research challenges and applications

Frank D. Macías-Escrivá [a], Rodolfo Haber [a,b,*], Raul del Toro [b], Vicente Hernandez [c]

[a] Departamento de Informática, Escuela Politécnica Superior, Universidad Autónoma de Madrid, C/Francisco Tomás y Valiente, s/n, 28049 Madrid, Spain
[b] Centro de Automática y Robótica (CSIC-UPM), Ctra. De Campo Real km 0,200, 28500 Madrid, Spain
[c] Departamento de Ingeniería y Arquitecturas Telemáticas, EUIT de Telecomunicación, Universidad Politécnica de Madrid, Ctra. Valencia Km. 7, 28031 Madrid, Spain

ARTICLE INFO

Keywords:
Self-adaptive software
Decision-making
Feedback loops
Software reflection
Model-driven development
Goal-based model

ABSTRACT

Self-adaptive software is capable of evaluating and changing its own behavior, whenever the evaluation shows that the software is not accomplishing what it was intended to do, or when better functionality or performance may be possible. The topic of system adaptivity has been widely studied since the mid-60s and, over the past decade, several application areas and technologies relating to self-adaptivity have assumed greater importance. In all these initiatives, software has become the common element that introduces self-adaptability. Thus, the investigation of systematic software engineering approaches is necessary, in order to develop self-adaptive systems that may ideally be applied across multiple domains. The main goal of this study is to review recent progress on self-adaptivity from the standpoint of computer sciences and cybernetics, based on the analysis of state-of-the-art approaches reported in the literature. This review provides an over-arching, integrated view of computer science and software engineering foundations. Moreover, various methods and techniques currently applied in the design of self-adaptive systems are analyzed, as well as some European research initiatives and projects. Finally, the main bottlenecks for the effective application of self-adaptive technology, as well as a set of key research issues on this topic, are precisely identified, in order to overcome current constraints on the effective application of self-adaptivity in its emerging areas of application.

1. Introduction

Since the mid-60s, the topic of system adaptivity has been widely studied and significant efforts have been made by the scientific community to find new approaches to elucidate the basic principles of self-adaptivity theory and practice. Thereafter, the scientific literature on self-adaptivity has been extensive, mainly over the past decade. Different interpretations and concepts have nevertheless been produced by the research community, as the subject is still the focus of intense research and development. Although a lack of consistency is normal in any evolving field of scientific research, it does little to support uniform global understanding.

Over the past decade, the importance of self-adaptivity has been increasingly acknowledged in its various application areas and related technologies. Software engineering and related topics are the common elements in the successful introduction of self-adaptivity. Hence, the imperative need to investigate systematic software engineering approaches for the development of self-adaptive systems, which would ideally be applicable across multiple domains (Brun et al., 2009).

Modern embedded software systems have to execute multiple tasks in diverse scenarios. These systems are increasingly expected to function in a dependent way in changing environments and to react to changes within the system. Self-adaptive embedded systems have to make decisions on adaptivity at runtime with respect to changing requirements (Weiss, Becker, Kamphausen, Radermacher, & Gérard, 2011). Self-adaptive software is capable of evaluating and changing its own behavior, whenever the evaluation shows that the software is not accomplishing what it was intended to do, or when better functionality or performance may be possible (Salehie & Tahvildari, 2011). Artificial intelligence represents an effective way of emulating adaptivity, making organized and efficient systems easier to reconfigure and more highly adaptive (Leitao, 2009).

The central goal of this paper is to review recent progress on self-adaptivity from the viewpoint of computer science and cybernetics. Its analysis and compilation of information from the literature identifies key scientific and technical challenges and assesses cutting-edge theories, methods and techniques for the design and development of self-adaptive systems. The main bottlenecks that

* Corresponding author. Tel.: +34 918711900.
E-mail addresses: frank.macias@estudiante.uam.es (F.D. Macías-Escrivá), rodolfo.haber@car.upm-csic.es, rodolfo.haber@uam.es (R. Haber), raul.deltoro@car.upm-csic.es (R. del Toro), vicente.hernandez@upm.es (V. Hernandez).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.07.033

prevent the immediate inclusion of self-adaptivity software into worldwide technology are also discussed.

The rest of this paper is organized as follow. Some definitions of self-adaptativity are discussed in Section 2. Section 3 focuses on the self-adaptive capabilities of complex systems, the main tools, and their methods, which range from traditional to recent approaches. Section 4 explores the main applications or application domains, as well as challenges and research opportunities in self-adaptivity that may take us beyond state-of-the-art applications in the near future. Finally, a few concluding remarks discuss current and future directions for further research.

2. Self-adaptivity: some definitions

Biology and nature offer plenty of powerful mechanisms, refined by evolutionary processes over millions of years, to handle emergent and evolving environments (Leitao, 2009). These natural self-adaptive systems have attracted the attention of software designers, because of the ease with which they can change their own behavior in response to changes in their environment (Nakagawa, Ohsuga, & Honiden, 2008).

This review of the literature on self-adaptivity has identified numerous definitions, a detailed discussion of which is beyond the scope of this paper. Here, we enumerate only some of the most commonly accepted definitions.

Definition 1. Self-adaptivity is the capability of the system to adjust its behavior in response to the environment. The "self" prefix indicates that the systems autonomously decide (i.e., with minimal or no interference) how to adapt or to organize themselves so that they can accommodate changes in their contexts and environments. While some self-adaptive systems may be able to function without any human intervention, guidance in the form of higher-level objectives (e.g., through policies) is useful and realizable in many systems (Brun et al., 2009).

Definition 2. In the context of multi-model systems, adaptation is a procedure or method for switching between models. Adaptivity can therefore be defined as the capability of a system to achieve its goals in a changing environment, by selectively executing and switching between models. This capability contrasts with the conventional use of the term adaptation used in mono-model systems, where design parameters or relations are tuned to fit observed behavior (Ravindranathan & Leitch, 1998).

Definition 3. A self-adaptive system consists of a closed-loop system (i.e., modify in runtime itself using feedback due to continuous changes of the system), its requirements and existing tendencies in developing and deploying complex system, thereby reducing human efforts in the computer interaction. The conception of self-adaptive system depends on user's requirements, system properties and environmental characteristics. Self-adaptive software requires high dependability, robustness, adaptivity, and availability (Naqvi, 2012).

Definition 4. A self-adaptive system evaluates its own behavior and changes its own performance when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible (Salehie & Tahvildari, 2011).

Although these definitions address run-time changes regarding functional and non-functional requirements, many researchers in the software engineering community have mainly focused their efforts on non-functional aspects.

Salehie and Tahvildari (2011) decompose the adaptation mechanism into several processes: *monitoring* software entities (self-awareness) and the environment (context-awareness), *analyzing* significant changes, *planning* how to react, and *executing*, so that the decisions to take effect. In most of the existing solutions, the adaptation processes are assigned to an external adaptation manager that is separate from the application logic. An *adaptation manager* realizes the four above-mentioned processes, to control the behavior of adaptable software. 1. The application logic is programmed in adaptable software that receives signals from the sensors and effectors required for self-adaptivity.

Self-adaptive topics have been widely addressed in several software engineering research areas: requirements engineering (Brown, Cheng, Goldsby, & Zhang, 2006; Sawyer, Bencomo, Whittle, Letie, & Finkelstein, 2010), software architecture (Den Hamer & Skramstad, 2011; Garlan, Cheng, Huang, Schmerl, & Steenkiste, 2004; Garlan, Cheng, & Schmerl, 2003; Oreizy et al., 1999; Richter et al., 2006), middleware (Geihs et al., 2009; Liu & Parashar, 2006; Schmitt, Roth, Kiefhaber, Kluge, & Ungerer, 2011), and component-based development (Bencomo, Grace, Flores, Hughes, & Blair, 2008; Peper & Schneider, 2008); even so, these are invariably isolated initiatives. Self-adaptivity and feedback have also been studied from the perspectives of Systems Theory, Artificial Intelligence and Computer Science providing theoretical foundations and application fields such as: control engineering, mobile and autonomous robots, multi-agent systems, fault-tolerant computing, dependable computing, distributed systems, autonomous computing, self-managing systems, autonomous communications, adaptable user interfaces, machine learning, economic and financial systems, business and military strategic planning, sensor networks, pervasive and ubiquitous computing, etc. Software is the key and common element that enables these systems to be self-adaptive.

3. Self-adaptivity: past and present in tools and methods

The complexity of current software systems and uncertainty in computational environments have motivated us to explore new ways for the design and management of systems and services (Abelson et al., 2000; Brun & Medvidovic, 2007a, 2007b; Di Marzo-Serugendo, Gleizes, & Karageorgos, 2005; Diao et al., 2005) in fields such as Artificial Intelligence, Control Theory and Biology. In this endeavor, the capability of the system to adjust performance prior to changes in the environment by self-adaptivity is one of the most active research directions.

3.1. Main approaches

Self-adaptive systems may be characterized by their operating conditions and by multiple dimensional properties such as centralization and decentralization, top-down and bottom-up approaches, feedback latency, and environmental uncertainty (low vs. high). The main approaches to deal with self-adaptivity are presented as follows, highlighting the key roles of computer science and cybernetics.

A top-down self-adaptive system is often centralized and operates with the guidance of a central controller or policy, assesses its own behavior in the current surroundings, and adapts itself accordingly, whenever its monitoring and analysis systems indicate that it should do so. This type of system often operates with an explicit internal representation of itself and its overall aims. An analysis of a top-down self-adaptive system and its components provides a clear understanding of the overall system and its behavior. On the contrary, decentralization is often a feature of cooperative self-adaptive or self-organizing systems, which function without a central authority. They are usually bottom-up,

have a large number of components and interact locally according to simple rules. The global behavior of the system emerges from these local interactions. Specific properties of a global system may not, therefore, be easily deduced from an analysis of only the local properties of its parts.

Most engineered and nature-inspired self-adaptive systems often reflect biological or sociological phenomena. In practice, the line between these two approaches is not clear and compromises will often lead to an engineering approach incorporating techniques from both alternatives (Brun et al., 2009).

3.1.1. External control mechanisms

Garlan et al. (2004) used external control mechanisms for self-adaptivity, based on the belief that they provide a more effective engineering solution than internal mechanisms, because they localize the concerns of problem detection and resolution in separable modules that can be analyzed, modified, extended, and reused across different systems. The proposed framework adopts an architecture-based approach, to provide a reusable infrastructure and mechanisms for specific systems.

Schmeck, Müller-Schloer, Cakar, Mnif, and Richter (2010) describe a system with either internal or external control mechanisms, which provide a way of controlling the behavior of the system by setting certain attributes of the system and its environment to specific values (contrary to the common belief that it is not possible to control environmental parameters). The control mechanism is considered a central entity, but it might be distributed or have a multi-level structure. The authors also define a degree of autonomy to be able to quantify how autonomously a system is working. The degree of autonomy measures external control that is exerted directly by the user (no autonomy) and distinguishes it from internal or external control of a system that might be fully controlled by an observer/controller architecture that is part of the system (full autonomy).

3.1.2. Component based software engineering (CBSE)

Bencomo et al. (2008) give an overall description of the approach implemented by Genie, a tool that supports the modeling, generation and operation of highly reconfigurable component-based systems. Likewise, Mishra and Misra (2009) utilized CBSE to design a model for a self-adaptive system that automatically performs the component integration process at runtime, by accessing the equivalent component from an *a priori* available repository. The model integrates the caching technique to reduce the time needed to search for the best-fitted component to replace the one that is required, when a system fails to respond due to component failure. Component assessment, in order to get the best alternative component, is done by numerical metadata, a measure that assigns the number of match functionalities for each component present in the repository. The computation of numerical metadata is based on the concept of an abstract syntactic tree and different systems will therefore be obtained before each environment at run time.

Yeom and Park (2012) described a scalable framework for developing adaptive, autonomous, highly distributed, and mobile agent-based network applications. They defined a set of key features for adaptive and autonomous component-based agents in their architecture. Furthermore, they also presented functional requirements for building an application composed of a federation of agents.

3.1.3. Model-driven

Supported by an innovative model-driven development methodology that is based on abstract adaptation models and corresponding model-to-code transformations, Geihs et al. (2009) designed a piece of middleware for the development and operation of context-aware, self-adaptive applications. They designed an abstract, platform-independent model that performed automatic code generation, providing high flexibility for the development of adaptive applications.

Vogel, Neumann, Hildebrandt, Giese, and Becker (2009) proposed a model-driven approach to developing self-adaptive systems with self-monitoring architectures. This approach can lead to incremental synchronization between the run-time system and models for different self-management activities.

In heterogeneous intelligent control (i.e., artificial intelligence-based control using heterogeneous knowledge), models are selected from different layers, involving different sources of knowledge. The knowledge used to determine the model switching strategy is crucial to its efficient operation. A default strategy can be encoded as a set of deterministic model-switching procedures. More generally, switching knowledge, or more formally meta knowledge, can also be encoded as either a set of rules or, at least conceptually, as a set of principles (Ravindranathan & Leitch, 1998).

3.1.4. Nature-inspired engineering

Nature-inspired engineering is a relatively young research area (Brun et al., 2009). Nevertheless, the application of nature-inspired strategies for designing self-adaptive software is receiving more attention from public and private sectors. One research line is centered on software and hardware design solutions (Abelson et al., 2000; Brun & Medvidovic, 2007a, 2007b; Di Marzo-Serugendo et al., 2007) inspired by natural and biological systems. Another research strategy is focused on how to build self-adapting software systems on the basis of the properties and the behavior of natural and biological systems (Clement & Nagpal, 2003; Shen et al., 2006; Yu, Ramaswamy, & Bush, 2008). Similarly, Leitão (2008) proposed a bio-inspired solution, presenting evolving mechanisms based on self-organization, supervision and learning concepts, and ant-based communication, supported by the use of multi-agent principles.

3.1.5. Multiagent systems

The key features of multiagent systems in the engineering of self-adaptive systems are, specifically, loose coupling, context sensitivity, robustness in response to failure and unexpected events. Goal-based, loose coupling of agents provides the flexibility needed for self-adaptivity and reuse (Weyns & Georgeff, 2010). Agents are independent objects that can accomplish their tasks and are flexible components that can be combined with and segregated from each other in a framework (Yeom & Park, 2012). Nowadays, new cooperative strategies for multi-agent systems and the combination of high-level compressed-state representation and a hybrid reward function produce the best results, in terms of both task completion rates and learning efficiency. An interesting application of self-adaptive agent-based fuzzy-neural system to enhance the performance of scheduling jobs in a wafer fabrication factory in proposed in (Chen, 2011).

Each component can be implemented as a mobile agent that can travel through computers using their own migration schemes, regardless of whether it is a homogeneous or heterogeneous agent. It means that each component can autonomously migrate to another computer or duplicate itself and send copies to others. Another important issue is to facilitate the dynamic association of one or more components in distributed systems. The framework can support low-level operating and networking details for agent migration and communication.

3.1.6. Feedback systems

Control engineering emphasizes feedback loops, elevating them to first-class entities (Franklin, Powell, & Emami-Naeini, 2006;

Hellerstein, Parekh, Diao, & Tilbury, 2004; Tanner, 1963), claiming that almost any system that is considered automatic has some element of feedback control. One reason for the widespread use of feedback control is its guarantee that the measured output will track a desired behavior or reference even in the presence of disturbance. The concept of feedback and feedback loops is therefore essential for enabling self-adaptive systems.

Over the past few decades, software engineering has to some degree neglected the relevance of dynamic aspects and has mainly considered static architecture for systems. On the contrary, control engineering and control methods have developed over decades on the basis feedback loops and dynamic models. A seminal paper by Magee and Kramer on dynamic structure in software architecture (Magee & Kramer, 1996) addressed relevant aspects to set the scientific foundations for many subsequent research projects (Cheng, Garlan, & Schmerl, 2005; Garlan et al., 2003; Kramer & Magee, 2007; Oreizy et al., 1999). However, while these research projects implement feedback systems, the actual feedback loops are kept hidden or are abstracted. Certainly, the feedback loop has an important and decisive role in software process management and its improvement and is essential to software evolution. Well-known examples are the feedback loops at every stage in Royce's waterfall model (Royce, 1970) or the risk feedback loop in Boehm's spiral model (Boehm, 1988).

Common to most of the existing approaches is their identification of four processes for adaptivity: *monitoring*, *analyzing*, *planning*, and *executing* (MAPE). In this section, we will explore tools and methods for addressing key topics in self-adaptive software systems, closely related to the achievement of previously mentioned processes, as illustrated in Fig. 1.

Broadly divided into two categories, these tools and methods may be either global or specific. Global tools and methods enable or assist designers and developers to create a self-adaptive system from the standpoint of a whole system. Specific methods are those for studying, designing, implementing, and supporting any of the aforementioned specific processes for self-adaptivity such as MAPE.

3.2. Global tools and methods

3.2.1. Models

Bencomo (2009) studied what is called an "eternal software system", a kind of system that is required to survive variations in its execution environment with or without human intervention. The research argues that run-time self-representations (runtime models) are the key to the production of an eternal software system. The author describes the methods to sustain the point of view that self-representation, reflection and architectural models are basic principles to support the use of runtime models.

Goal-based models have proven to be effective for the specification of self-adaptive systems, monitoring and switching between

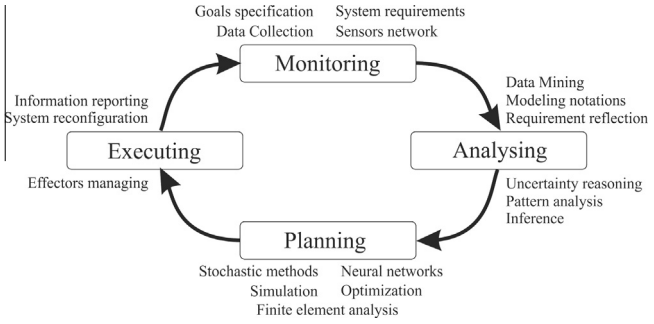

Fig. 1. Adaptivity processes and associated topics.

adaptive behaviors. The ability to reason about partial goal satisfaction is a particular strength of goal-based modeling. A notable body of work in software engineering has applied goal-based modeling notations, such as i∗ (E. Yu, 1995), to discover and to specify the requirements of self-adaptive systems. Indeed, i∗ represents goals as a structure of connected nodes without explicitly representing a logical relationship between the parent and offspring. Another tool called Soft-goal Interdependency Graph (SIG) uses an AND-OR graph which decomposes and relates goals using AND and OR relationships. For example, the NFR framework benefits from SIG (Chung, Nixon, Yu, & Mylopoulos, 2000). Design decisions stimulate the model from operationalization nodes, and the reasoning propagates the impacts though goals. Designers evaluate their decisions, by reviewing the degree of satisfaction with goals that are achieved.

The majority of existing architectures for the adaptation manager in self-adaptive software take advantage of the Sensor-Plan-Act (SPA) model, used extensively in building traditional robotic systems (Salehie & Tahvildari, 2011). In these systems, events are collected, analyzed, and fused to update the domain model (i.e., global model). The system then plans its strategy in the new situation. However, the idea of behavior-based robotics is to use distributed specialized task-achieving modules, called behaviors, and to apply command fusion instead of sensor fusion (Arkin, 1998). So, there is no need to develop, to maintain, and to extend a coherent monolithic model of the adaptable software and its context. However, while goal-driven models are used in requirements engineering, they have not previously been systematically used for run-time adaptation.

More than two decades ago, Rao and Georgeff (1995) introduced the basics of the Belief-Desire-Intention (BDI) agent architecture model. The BDI model is an agent building style based on an agent architecture that provides a reasoning mechanism with the concepts of belief, desire (or goal) and plan. This pioneering model and its extensions in multi-agent systems are well-established and built on a strong body of knowledge, which could be useful in designing self-adaptive software systems; some of the frameworks discussed later in this article are supported by this model.

Today's distributed embedded systems comprise various fields of application in a complex 'system of systems' scenario, which hold great advantages in terms of flexibility, resource utilization, energy efficiency and robustness. They should be able to adapt to changes in the environments and the internal system, requiring enhanced development methods to incorporate adaptation into their design. Weiss et al. (2011) present the self-∗ profile, a modeling extension for describing the adaptation, and the respective design flow with built-in transformations.

3.2.2. Simulation

Some essential theories, techniques and computational procedures for modeling and simulation are identified by Couture (Couture & Valcartier, 2007), and summarized as follows.

- Game Theory. This is a theory of interactions based on decisions and relative advantage. This quantifies decision fitness at an individual pair level, but is harder to apply to more diffuse systems. The important aspect here is how to distinguish positive from negative evolutionary paths – goal directed behaviors.
- Spin Glasses. This technique, borrowed from physics, uses a lattice of interacting points and is chiefly found in complexity work on cellular automata, which can be used to model many physical phenomena. The technique, whilst excellent for simulation, proves mathematically difficult, but is important in relation to the demonstration of emergent, higher level structures.

- Time Series Analysis. Based on communications theory, time series analysis seeks to identify regularities in the behavior of a system over time, trying to quantify cyclic or chaotic (strange) attractors. It is often applied to financial systems. The main drawback is that the system must have a large amount of data for analysis, though its advantage is that limits can be placed on system behavior.
- Fuzzy Logic. In the analysis of nonlinear systems, fuzzy logic gives us a way of quantifying many interacting variables, generating a result that maps all possible interactions of the inputs. This technique has yet to be widely applied to complex ideas, but has importance insofar as it has the potential to treat multiple conflicting variables in decision systems.
- Multiobjective Optimization. This idea, drawn from operational research, attempts to account for the interdependency of multiple values in the real world and in combination with evolutionary computation, helps us to study the dynamics of epistatic systems and multiple global optima (Pareto fronts) common to such systems. It involves many techniques, some of which involve synergic considerations.
- System Dynamics. This computer modeling technique attempts to quantify how the dynamics of systems, based on assumptions of how the parts/variables are interconnected (their dependency structure), differs from our preconceptions about such dynamics. It highlights the difficulties of predicting actual complex system behavior when our views are constrained by the results of over-simplified reductionist experiments.
- Evolutionary Dynamics. By statistically measuring the diversity, cumulative activity and innovations of evolving systems it becomes possible to classify these in terms of their open-ended evolutionary potential. The technique allows the emergent behavior of artificial and natural systems to be determined. However, few, if any artificial systems currently show any unbounded emergent potential.
- Multi-Agent Systems. This technique, based upon artificial life ideas, studies the dynamics of collections of interacting autonomous agents. The self-organization that follows from different initial assumptions and sets of agent values helps to quantify how different features of real systems can interact and evaluates their stability to perturbations caused by changes in their internal structure and their goals. Leitão (2008) has studied the design of reconfigurable manufacturing systems supported by the use of multi-agent principles; multi-agent systems, derived from distributed artificial intelligence, suggest the definition of distributed control based on autonomous agents that can perform efficient, flexible and robust manufacturing control.

Discrete Event System Specification (DEVS) (Zeigler, Kim, & Praehofer, 2000) is a formalism, which provides a means of specifying the components of a system (of systems) in a discrete event simulation. In DEVS formalism, one must specify basic models and how they connect up with each other. These basic models are called atomic models and larger models, which are obtained by connecting these atomic blocks in meaningful fashion, coupled models. Each of these atomic models has in-ports (to receive external events), out-ports (to send events), a set of state variables, internal transition, external transition, and time advance functions.

3.2.3. Architecture

New architectural concepts and services embodying layers of middleware are now possible as hardware and software artifacts become faster, cheaper, and better at a relatively predictable rate coupled with the growing acceptance of a network-centric paradigm (where distributed applications with a range of QoS needs are constructed by integrating separate components connected by various forms of communication services).

Schantz and Schmidt (2002) stated that the growing importance of middleware stems from recognition of the need for more advanced support – beyond simple connectivity – to construct effective distributed systems. For over twenty years, the industry has developed various architectural solutions based on middleware technologies to alleviate many complexities associated with developing software for distributed applications. Some of the most successful middleware technologies have centered on distributed object computing (DOC). DOC is an advanced, mature, and field-tested middleware connectivity paradigm that also supports flexible and adaptive behavior. At the heart of distribution middleware are request brokers, such as Common Object Request Broker Architecture (CORBA) from OMG, Java Remote Method Invocation (RMI) from Sun, Distributed Component Object Model (DCOM), and Simple Object Access Protocol (SOAP) from Microsoft.

New results have emerged from the paradigm of Service Oriented Architecture (SOA). Dustdar, Goeschka, Truong, and Zdun (2009), addressed the challenges of how to adapt services, processes, and teams to changing situations by means of the SOA-based approach, comprising model-driven compliance support, runtime interaction mining, run-time management of requirements, and an explicit control-loop architecture.

The AESOP (ArchitecturE for Service-Oriented Process-Monitoring and -Control) (Karnousko, Colombo, Jammes, Delsing, & Bangemann, 2010) initiative envisages a SOA approach for monitoring and control (batch and continuous processes). The SOA-based approach proposed by AESOP can, on the one hand, simplify the integration of monitoring and control systems as an application layer. On the other hand, these networking technologies can simplify the inclusion or migration from existing solutions and integration of the next generation SCADA and distributed control systems at a network layer.

Mittal, Zeigler, Risco Martín, Sahin, and Jamshidi (2009) describe how various elements such as automated DEVS model generation, automated test-model generation, and net-centric simulation over SOA are put together in the DEVS Unified Process (DUNIP) (Mittal, 2007). The infrastructure provides for a platform-free specification language DEVSML (Mittal, Risco-Martin, & Zeigler, 2007b) and its net-centric execution using a service-oriented architecture called DEVS/SOA (Mittal, Risco-Martin, & Zeigler, 2007a). Both the DEVSML and DEVS/SOA provide novel approaches to integrate, collaborate, and remotely execute models on SOA.

Nakagawa et al. (2008) describe an approach to developing self-adaptive systems utilizing a requirements model to build the system architecture. They illustrate the generation of system architectures by using descriptions of goal-oriented requirements analysis. Later, Nakagawa et al. (Nakagawa, Ohsuga, & Honiden, 2010), stated that in their framework, developers build an architecture model representing components and the connections between them first, and then implement these components by inheriting our extended behavior class in order to construct an agent corresponding to the self-$*$ system.

Finally, the IBM Autonomic Computing Architecture (IBM-Corporation) is one of the pioneering architectures for self-adaptive systems that explicitly exposes the feedback control loop mechanism and the adaptation steps as shown in Fig. 2. Functional components and interfaces are identified for decomposing and managing the feedback loop. This architecture provides a framework for building self-managing systems as long as it constitutes a blue print for developing feedback control loops.

The competing and complementary strategies and approaches to middleware based solutions simultaneously represent a healthy and robust technical area of continuing innovation, as well as a

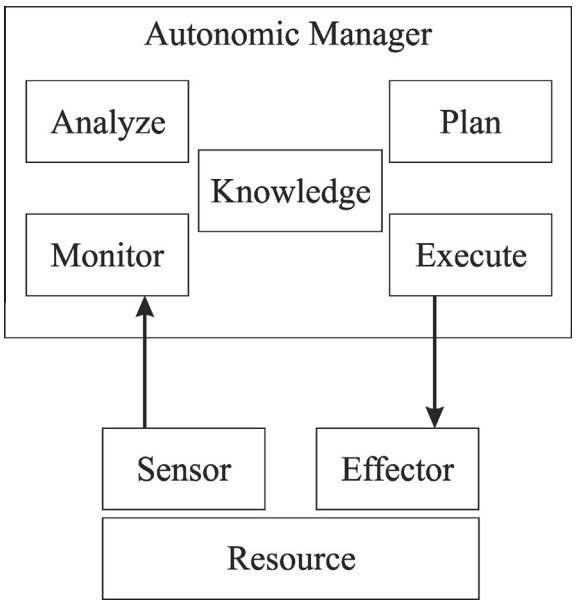

Fig. 2. IBM autonomic computing architecture.

source of confusion due to the multiple forms of similar capabilities, patterns, and architectures (Schantz & Schmidt, 2002).

3.2.4. Frameworks

R&D into middleware patterns, frameworks, and standards for distributed systems have played a leading role in establishing the technical viability of collections of systems that can dynamically adapt, within real-time constraints, their collective behavior to varying operating conditions, delivering the appropriate application level response under these different conditions.

Two of the most representative approaches in the domain of middleware frameworks, are Java 2 Enterprise Edition (J2EE) (Oracle) and .NET (Microsoft). Both approaches have introduced advanced software engineering capabilities to the mainstream IT community and incorporate various levels of middleware as part of the overall development process, although only partial support is available for adaptive performance of critical and embedded solutions. In the case of J2EE, promising extensions have been developed, some of which are discussed in this paper.

Adaptive Communication Environment (ACE) (DOC-Group) over other host infrastructure middleware alternatives (.NET/CLR and J2EE) is another interesting solution. ACE is a freely available, highly portable toolkit that shields applications from differences between native OS programming capabilities, such as file handling, connection establishment, event de-multiplexing, inter-process communication, (de)marshaling, concurrency, and synchronization. These characteristics motivate some world-wide applications (Schantz & Schmidt, 2002).

The Open Services Gateway initiative (OSGi) framework (OSGi-Alliance, 2012) is a modular system and services platform for Java programming language that implements a complete and dynamic component model, which is not sufficiently well addressed in standalone Java/VM environments. Applications or components (coming in the form of bundles for deployment) can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot; management of Java packages/classes is specified in great detail. The application life-cycle management (start, stop, install, etc.) is done via APIs that allow for remote downloading of management policies. The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly. Some authors (Choonhwa & Nordstedt, 2003; Gu, Pung, & Zhang, 2004; Helal et al., 2005; Jonghwa, Dongkyoo, & Dongil, 2005) have used the OSGi framework for the development of adaptive software systems. Some widely used current framework implementations, all under open source licenses, are, for instances, Knopflerfish, Apache Felix, Concierge OSGi and Equinox.

JADE (Java Agent DEvelopment Framework) (Telecom-Italia), a software framework fully implemented in Java, provides a multi-threaded programming style for constructing agents. It simplifies the implementation of multi-agent systems through a middleware that complies with Foundation for Intelligent Physical Agents (FIPA) specifications and through a set of graphical tools that supports the debugging and deployment phases. The agent platform can adopt a cross-machine distribution (not even needing to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another, as and when required. JADE allows developers to describe concurrent behaviors on it, which can be a foundation for constructing multi-processes on self-$*$ systems.

JADE provides a multi-threaded programming style for constructing agents, although by itself it provides no support to the agent building style of the BDI model. The Jadex (Pokahr, Braubach, & Lamersdorf, 2003) framework implements a BDI-infrastructure for JADE agents, in order to provide a reasoning mechanism with the belief, desire (or goal) and plan concepts. This framework is also used as a platform for self-$*$ systems. Since developers can describe plans and these activating conditions in Jadex, it satisfies the *autonomous activation* requirement; however, it requires developers to describe the relationship between plans and goals, and this makes it difficult to understand the concurrent processes.

Some effort has also gone into DEVS formalism, mainly aiming at support for multiplatform simulation capability as provided by DEVS/SOA framework. It consists of distributed simulation between different DEVS platforms or simulator engines such as DEVSJAVA and DEVS-C++, on Windows or Linux platforms.

3.3. Specific tools and methods

Specific tools and methods are those that support one or more specific tasks for self-adaptivity: *monitoring* of the managed system or the environment to collect data for adaptation, *analysis* of the monitored data to take adaptivity decisions, *planning* to determine the steps to achieve adaptivity, and the *execution* of the steps of the plan to achieve adaptivity.

3.3.1. Feedback control loops

Feedback control loops are considered a key issue in pursuing self-adaptivity for any system, because feedback control supports the four above-mentioned processes as inherent to adaptation: MAPE. The classic structure of a feedback control loop is depicted in Fig. 3.

Core to the design of a feedback control loop is the choice of controller. One of the most used so far is the PID (Proportional-Integral-Derivative) controller (Franklin et al., 2006) with regard to precision and stability (Peng, Chen, Yu, & Zhao, 2010). PID

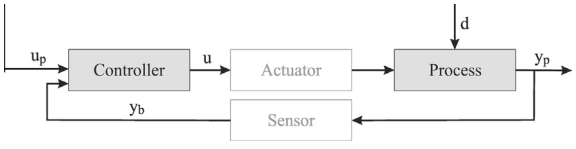

Fig. 3. Block diagram of a feedback control loop.

controllers are usually the best choice, where only rough knowledge of the underlying process is available (Bennett, 1993).

Brun et al. (2009) consider that adaptive control in control theory involves modifying the model or the control law of the controller to be able to cope with slowly occurring changes in the controlled process. Therefore, a second control loop is installed on top of the main controller. This second control loop adjusts the controller's model and operates much slower than the underlying feedback control loop. For example, the main feedback loop, which controls a web server farm, reacts rapidly to bursts of Internet load to manage QoS. A second slow-reacting feedback loop may adjust the control law in the controller to accommodate or to take advantage of anomalies emerging over time. New approaches based on adaptive observers and artificial neural networks have also been applied to mobile robots with limited information (Bong Seok, Jin-Bae, & Yoon-Ho, 2011).

3.3.2. Decision-making

Some researchers have constructed software to carry out decision-making process by using rule-based approaches or control theory. In some cases, decision-making systems are inspired by biological processes, such as the human nervous system and emergent behavior in insect species that form colonies (McKinley, 2004).

According to MAPE methodology, when the alternative actions have been analyzed, decision-making is applied to decide (to plan) which actions will be performed, if any. Context information may also be used during adaptation planning and decision-making (Gjorven, Eliassen, & Aagedal, 2006). Therefore, *Adaptation Planning*, *Decision-making* and *Context Sensing and Reasoning* services should be provided as part of any self-adaptive service platform.

Among the different forms of decision-making noted by Roy (1996)-selecting, sorting, ranking, and description- the decision-making process in self-adaptive software is closer to the selecting format. The decision-making process basically addresses an action selection problem, in order to select a proper action from a finite set of alternatives (Salehie & Tahvildari, 2011). In a goal-driven approach the problem of decision-making can be defined as:

Given an adaptation goal set G, an adaptation action set AC, and an attribute set AT from a software system, the problem is how to build a goal-action-attribute model and to select the appropriate action ac_i at run-time, to satisfy goals under different conditions.

This problem has some similarity to decision-making problems in robotics and agent-based systems. Most existing solutions for self-adaptive software benefit from the Sensor-Plan-Act (SPA) model, used extensively in building traditional robotic systems. In a goal-driven model, decision-making mechanisms can perform in two general *competitive* and *cooperative* forms. Goals compete with each other when selecting the next action in competitive decision-making procedures, whereas in the cooperative form, the preferences of goals are combined and fused to determine what to do next. In the *cooperative* category, Arkin (1998) suggested that super-positioning (vector addition) is the most straightforward method, if it is feasible. In the *competitive* methods, arbitration is a way to select one goal (winner-takes-all), e.g., based on predefined priorities. Maes (1994) argued that autonomous agents with no explicit goals and goal-handling capabilities will lead to significant limitations in their operation, and went on to propose a less autocratic method: an activation network for actions, in order to facilitate dynamic action selection based on stimulated goals or actions.

A notable point about a goal-driven decision process is that it may not be possible or even efficient to use automated planning, in this case, reactive planning. Planning can be an appropriate option for self-healing, but is generally not effective for self-optimizing (Salehie & Tahvildari, 2011). Therefore, having a goal-based decision-making mechanism does not necessarily mean that a planning-based approach is employed.

There are more "democratic" ways of decision-making, based on adopting a voting mechanism. In general, voting-based mechanisms can arguably be placed in the *competitive* category. Their social choice methods and voting games are well known in cooperative game theory, in order to combine decisions made by agents.

Decision-making problems have been extensively studied in many fields. The selection of an action from a set of alternatives becomes harder when the decision-making process involves several criteria rather than a single criterion. These types of problems are known as Multi-Criteria Decision-Making (MCDM) problems. Sridhar, Madni, and Jamshidi (2009) proposed an innovative mechanism for MCDM problems, with decision-making at the coordinator level, based on several competing and/or contradicting criteria that exist in systems within an SoS.

They generalize the problem of MCDM in the following way (Sridhar et al., 2009):

Let $\Omega = \{s_1, s_2, \ldots, s_m\}$ and $X = \{x_1, x_2, \ldots, x_m\}$ be a set of alternatives and a set of criteria, respectively. The decision-making process proceeds by formulating a matrix A with set of criteria and set of alternatives, given by:

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$$

Each entry (a_{ij}) denotes the degree to which the criterion (x_j) is satisfied by the alternative (s_i. The idea is now to reduce the multicriteria problem into a single global criterion problem, by aggregating all the elements of matrix A, given by $a = H(a_1, a_2, \ldots, a_{mj})$, with H as the aggregation operator. The most common aggregation operator is the weighted arithmetic mean. In a later work, Sridhar et al. (2009) investigated the pitfalls of common aggregation operators (such as weighted mean) and provided a countermeasure for aggregating criteria without using common aggregators.

Sawyer et al. (2010) proposed a mathematical framework that supports decision-making over requirement alternatives; the parameters for the decision model should be measurable so that they can be related to the data collected during system monitoring; and the computational complexity of the decision model should be such that it can be evaluated efficiently at run-time. This framework is built on existing outranking and interactive approaches to multi-criteria decision-making (Roy, 1996), as well as on research evaluating alternatives and dealing with conflicts in goal models.

In technical terms, cognitive decision-making is generally understood as mimicking a human-like complex mental decision process. Implementations often rely on incremental and recursive reasoning and on inference processes closely associated with machine-learning strategies to refine rule sets and to resolve conflicts (Bochow & Emmelmann, 2011). The human being, when making decisions, tends to work with vague or imprecise concepts, which can often be expressed linguistically. One way of modelling decision-making processes is based on the theory of Approximate Reasoning (AR), which enables certain classes of linguistic statements to be treated mathematically. The foundation of AR is Zadeh's theory of Fuzzy sets, which sets out to preserve the approximate nature of human reasoning rather than trying to avoid it. His viewpoint is that this element of imprecision is an important factor contributing to human intelligence (Zadeh, 1965). Fuzzy systems, neuro-fuzzy systems, and theoretical development related to Fuzzy Logic and applications are very active research topics, a thorough review of which lies outside the scope of this paper. A complete survey on hybrid expert systems is provide in Sahin, Tolun, and

Hassanpour (2012). There is a clear trend in neuro-fuzzy and rough neural expert systems' areas with the aim of enabling self-adaptivity. A novel design of an adaptive neuro fuzzy inference system (ANFIS) for estimation contact forces of a robotic manipulator in proposed in Petković, Pavlović, Ćojbašić, and Pavlović (2013). Self-adaptivity is indeed fostering the dynamic adjustment of the searching space according to the performance indices (Hung & Lin, 2012; Wu, 2011).

Formal language and specification plays an important role in supporting predictable dynamic reconfiguration, metadata, component descriptions, and policies, making them all available at run-time. It is likely that several different languages for the various elements mentioned above would be required. Each could be an extension of an existing language, or could be brand new, but each should be as "formal" as possible, in order to allow run-automated reasoning at the semantic level, for example in determining substitutability of services, acceptable degraded performance characteristics etc. The essential forms of specification are: description of patterns; self-description of components; specification of metadata; and specification of policies. Each of these forms of specification may be used in either design-time or run-time decision-making processes (Di Marzo-Serugendo et al., 2007).

3.3.3. Requirements engineering

Adaptive software systems are increasingly being used in various domains, such as medical community, software industry, manufacturing and services of all kinds. Therefore, understanding the requirements of an adaptive software system is critical to successful development and deployment, as a means of taking advantage of adaptation semantics that describe how systems behave during adaptation.

Sawyer et al. (2010) argued that requirements for self-adaptive systems should be run-time entities that can be reasoned, in order to deal with the uncertainty of unanticipated contexts that prompt new requirements, to understand the extent to which these requirements are satisfied at any one time, and to support adaptation decisions that can take advantage of the systems' self-adaptive machinery. They take inspiration from the fact that explicit, abstract representations of software architectures used to be considered design-time-only entities, but *computational reflection* showed that architectural concerns could be represented at run-time too, helping systems to reconfigure themselves dynamically as the context changed.

Architectural reflection provides another viewpoint on how requirements may become run-time artifacts. Architectural reflection involves introspection of the underlying component-based structures. It is arguable that the same principles may be applied that permit introspection and reasoning based on (meta-) models of requirements at run-time (Sawyer et al., 2010). Requirement reflection enables self-adaptive systems to revise and to re-evaluate design-time decisions at run-time, when more information can be acquired on these by observing their own behavior. Current work on computational reflection offers a potential way to structure the run-time relationship between the requirements model and the architecture.

The KAOS methodology (Dardenne, van Lamsweerde, & Fickas, 1993) provides a graphical way to present the adaptation semantics. Brown et al. (2006) identified high-level objectives for each adaptation semantics and represented them as the corresponding KAOS goal entity. Different states in the adaptive semantics have also been identified and represented on the basis of KAOS requirements. Nakagawa et al. (2008) also use KAOS as a method for analyzing and modeling goal-oriented requirements for constructing a self-adaptive system.

In principle, a controller at runtime can monitor the change impact on quality requirements of the system, update the expectations and priorities from the environment, and take reasonable actions to improve the overall performance. In practice, however, existing controllers are mostly designed for tuning low-level performance indicators rather than high-level requirements. Peng et al. (2010) proposed a theoretical self-tuning method, by linking overall satisfaction to a business value indicator as feedback, that can dynamically adjust the tradeoff decisions as a result of different quality requirements.

In goal-based modeling approaches, it is common practice to build a structure to relate goals, attributes, and actions (Salehie & Tahvildari, 2011). In software engineering, goals are mainly defined and utilized in requirements specification, such as i∗ (Yu, 1995) and SIG (Chung et al., 2000). Used in this way, goal-based models have proven to be effective for the specification of requirements of a self-adaptive system and for the specification of monitoring and change between adaptive behaviors; a consequence of their ability to reason about partial goal satisfaction, an accepted particular strength of goal-based modeling.

Close collaboration between requirement engineers and software architects for self-adaptive systems can also provide very promising benefits (Bencomo, Grace, & Sawyer, 2009). Decisions made about the computational architecture can give early feedback and enable analysis during the requirements specification. A self-adaptive system can be modeled as a collection of target systems, each of which correspond to, and operate within, a state of the environment.

Adaptivity is usually described through a set of properties called self-∗ properties, such as self-configuring, self-healing, self-optimizing, and self-protecting properties, among numerous self-∗ properties reported in the state-of-the-art, as being aligned with non-functional requirements (NFR) and software quality factors. Therefore, it is quite important to analyze goals at run-time, given that they provide requirements traceability, or 'requirements reflection' (Bencomo, Whittle, Sawyer, Finkelstein, & Letier, 2010).

3.4. Limitations to go beyond the state of the art

Along with other researchers (Brun et al., 2009; Dustdar et al., 2009; Müller, Pezzè, & Shaw, 2008) in the field of software engineering, we strongly support hidden, abstracted, dispersed, and internalized feedback loops when presenting and documenting the architecture of an adaptive system. Commonly used software modeling notations (e.g., UML) provide no means of describing and analyzing control and procedure to deal with uncertainty. Furthermore, the lack of a notation implies the absence of an explicit task to document control, which would in turn imply a high risk of failure, when explicitly designing, analyzing, and validating the feedback loops (Brun et al., 2009).

The most successful attempt to overcome the previously mentioned lack of notation for control loops was the XML-based markup language, SCXML (State Chart XML: State Machine Notation for Control Abstraction) (W3C, February 2007). It provides a generic state-machine based execution environment founded on Harel state charts. SCXML is able to describe complex state-machines. For example, it is possible to describe notations such as sub-states, parallel states, synchronization, or concurrency, in SCXML. This standard has evolved from a specific domain standard (CCXML, designed to support call control features in voice applications), to become a general-purpose event-based state machine language.

The incorporation of closed-loop mechanisms into software systems is imperative, so that they can adapt themselves to changing conditions. Regarding the scale of existing software applications, their dynamic environments, and variable system requirements, the software operation management is often costly, time-consuming, and likely to be error-prone. This problem can be attributed to the open-loop structure of many existing software systems, and

the need for continuous human supervision. One of the shortcomings of software engineering for self-adaptive applications is the lack of actual case studies. Very limited scenarios are nowadays available such as toy applications. Moreover, some scenarios of self-adaptive software allow no adaptation at an application level. The control actions are mainly focused on the middleware, server, network, and even the operating system.

Current approaches to decision-making in adaptive software have been effective in certain domains, but environmental dynamics and software complexity have limited their general application (McKinley, 2004). Another reported issue is the inefficiency of the decision-making mechanism, which should be considered a current limitation concerning system performance.

Goal-based approaches to adaptivity are not widely addressed in many research efforts in the self-adaptive software area (Salehie & Tahvildari, 2011). The well-established goal models in requirements engineering (RE) are basically designed for development time. Their purpose is to analyze different design decisions for software engineers instead of focusing on run-time evaluation of goals for automated decision-making.

Leitao (2009) stated that adaptation in SoS poses important future challenges. For example, how to adapt their emergent behavior using learning algorithms is still a long way off being answered.

4. Application domains: challenges and opportunities

This section enumerates some important applications and application domains of self-adaptive systems. Although not an exhaustive list, we have tried not to restrict ourselves to the standard range of finance, healthcare, aerospace and military applications for system of systems (SoS) and self-adaptive software systems.

Subsequently, current challenges and research opportunities will be discussed, based on weaknesses in the way adaptation concepts are applied at present and on the valuable work done over the last decade in this field.

4.1. Applications and application domains

It should be noted that normally there is a mix of domains in every application, although one or more can be predominant, for example, sensors, wireless devices, embedded systems, and of course, software systems, are part of the majority of other domain applications. Hence, we should examine the following from a standpoint of integration with multidisciplinary concepts very much in mind.

4.1.1. SoS and intelligent infrastructure systems

Infrastructure Systems, providing services such as energy, transport, communications, and clean and safe water are vital to the functioning of modern society (Jamshidi, 2008).

Thissen and Herder (Jamshidi, 2009) illustrated the application of systems of systems concepts with reference to a possible transition of the energy system toward sustainability, where they took a model-based analytic approach to systems. In a second case study, they illustrated the application of different system of systems modeling techniques to the possible design and implementation of flexible multi-fuel energy provision in the Netherlands.

As part of a European initiative, the MULTIFORM project (Hüfner, Fischer, Sonntag, & Engell, 2012) will enhance tool support of an integrated model-based design process of the physical system, the controllers, and the communication and software infrastructure for the integrated control design of large and complex networked systems.

4.1.2. Sensor networks and embedded systems: DEMANES approach

The main purpose of sensor networks is to utilize the distributed sensing capability provided by tiny, low-powered, and low-cost devices. Multiple sensing devices can be used cooperatively and collaboratively to capture events or to monitor space more effectively than a single sensing device (Jamshidi, 2009). The realm of sensor networks ranges from the environmental to the military, including manufacturing, commercial and health systems, to name but a few. The heterogeneity of devices and ever-changing scenarios and requirements, provides a tremendous opportunity for case studies in self-adaptive software systems.

Genie models have been used to develop GridStix, a grid-enabled wireless sensor network for flood management that has been deployed (in prototype form) on the flood plain of the River Ribble in North Yorkshire, England (Bencomo et al., 2008). Its Gridkit middleware runs on every sensor and creates a self-managing sensor network that is capable of reacting to changing conditions and node failures caused by the flooding to provide a continuous flow of environmental information.

Another application is the self-adaptivity of a sensor network deployed for a wide area surveillance task. To that end, Rogers et al. considered a wide area surveillance problem based upon a simulation of an urban settings (using the Robocup Rescue Simulation Environment -see http://www.robocuprescue.org/) (Rogers, Jennings, & Farinelli, 2009).

The CHOSeN project (*www.CHOSeN.eu*) aims to develop application-specifically adaptable communication technologies enabling the real deployment of smart wireless sensor networks in large-scale, performance-critical application fields such as the automobile and aeronautic sectors.

The GINSENG project (*www.ict-ginseng.eu*) planned a significant advance beyond the state-of-the-art by developing a novel performance controlled WSN that is designed for use in a range of industrial environments. GINSENG, which ended in February 2012, deals with QoS at the communication level. Its results indicate that devices can be given the ability to determine communication QoS. This can then be offered as a service to a global device, while subsystem and functionality integration is addressed by AESOP (Karnousko et al., 2010).

Another European project is WIDE (Decentralized and Wireless Control of Large-Scale Systems – see http://ist-wide.dii.unisi.it), an advanced control and real-time optimization of large-scale and spatially distributed processes based on the integrated use of distributed model predictive control and wireless sensor feedback.

A further research area for providing self-adaptive functionalities consist of middleware-based approaches. In the RUNES project (Batori, Theisz, & Asztalos, 2012), a component based middleware for reconfigurable networked embedded systems and wireless sensor networks was investigated. Heterogeneity is supported by installing the middleware on top of different supported operating systems for different types of hardware platforms.

The DEMANES project (www.demanes.eu) is a new on-going research initiative in the field of design, monitoring and operation of adaptive networked embedded systems. The goal of DEMANES is to provide a framework and component-based methods and tools for the development of run-time adaptive systems, making them capable of reacting to changes in themselves, in their environment (battery state, availability and throughput of the network connection, availability of external services, etc.), in user needs (requirements), and in changing contexts.

DEMANES combines recent advances from systems and control engineering, in order to move beyond the state of the art. The concept, methodology and tools developed in DEMANES will be validated and demonstrated in three case studies: smart safe and secure urban transport and environment, smart airport

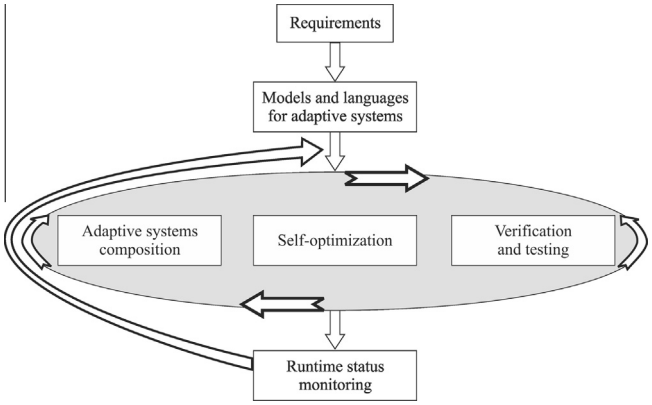

Fig. 4. The concept of the DEMANES project.

management and cooperating sensors at home. Run-time adaptivity makes the systems capable of reacting to changes in themselves, in their environment (battery state, availability and throughput of the network connection, availability of external services, etc.) and in user needs (requirements). The concept of DEMANES is depicted in Fig. 4.

DEMANES follows a strict model-based approach spanning all phases of system development, testing, deployment and operation. The component and composition models used in formalizing the design alternatives are refined during the development process and assist with the quantitative evaluation of alternatives (i.e., guiding the designer to make the proper design choices). Relevant subsets/aspects of the models are brought over to the executable system to facilitate self-organization and self-optimization. The DEMANES project aims to develop a smart integrated tool chain, reusable components and a framework for the design, implementation, testing, validation and operation of adaptive networked embedded systems. In addition to the tool chain, DEMANES will further deliver a model-driven design methodology, reference designs for characteristic dependable, real-time distributed systems and a pilot implementation of a runtime platform for applications designed according to the methodology developed. The platform will provide for system's self-awareness by means of performance monitoring, runtime functional contract checking, monitoring of real-time properties and reconfiguration.

4.1.3. Social services

Social services and applications are reflecting how new self-adaptive functionalities can tackle relevant aspect. A clear sample, is how a simple method for categorizing resources on social tagging systems can be self-adaptive, scalable and implementable in any real social tagging system (Córdoba, Astrain, Villadangos, & Echarte, 2013).

Another area is the health-related services. The backbone of network-centric health care operations is a set of interconnected web-enabled information networks with intelligence capabilities that support the seamless transfer of all necessary data and information to the point of health care delivery so that the doctor or decision-maker's choices are always based on the best possible data, information, and knowledge (Jamshidi, 2009).

Decision-making has also become a key process in the novel approach based on system of systems (SoS) applied to dispute management systems. Increasing attention has focused on more effective dispute avoidance and resolution due to the significance of the costs associated with disputes. Many researchers have attempted to develop systems that aim to manage disputes by providing dispute evaluation (Ilter, 2011). SoS approach can be useful in this domain because, while maintaining the same amount of management and resources as before, more precise results can be obtained from each system in the decision-making processes of the users.

Aggressive and conservative bidding strategies reinforce adaptation in the variations of resource availability in *ad hoc* grids. Li and Li (2012) proposed an *ad hoc* grid resource management system, the producers and consumers of the *ad hoc* grid resource are modeled on the self-interested decision-makers described in microeconomic theory.

4.1.4. Manufacturing industry

The AESOP (Karnousko et al., 2010) initiative comprises a SOA approach for monitoring and control of Process Control applications (batch and continuous process). Large process industry systems are a complex (potentially very large) set of (frequently) multi-disciplinary, connected, heterogeneous systems that function as a complex system of which the components are themselves systems. The proposed approach can simplify the integration of monitoring and control systems on application layer.

4.1.5. Traffic and transportation

R&D is currently directed at the domain of Air Traffic Control (ATC), to improve the way in which operators perceive and understand complex situations. The solutions that have been found contribute to building an understanding of complex situations easily and quickly in high-stress situations. Militarily complex operations using complex systems present similar problems and needs (Couture & Valcartier, 2007).

National transportation systems are collections of networks composed of heterogeneous systems (sector). Research on each sector is generally conducted independently, occasionally missing important interactions between sectors, or even within a sector (Jamshidi, 2009). A systematic method for modeling these interactions is essential to the formation of a more complete model and understanding of any transportation system, enabling them to apply the body of knowledge accumulated over recent years in the fields of SoS and adaptive systems.

An application for an Automatic Guided Vehicle (AGV) transportation system illustrates the self-adaptive value of multi-agent systems. One remarkable implementation of this idea is the Emc^2 (Egemin Modular Controls Concept-see http://emc2.egemin.com) approach. This is an R&D project which has applied agent technology to the development of a self-adaptive control system for an automated transportation system.

4.1.6. Software industry

Some antivirus systems (e.g., IBM's AntiVirus) apply knowledge of the behavior of biological systems such as cells and social insects as their inspiration for implementing emergent computing solutions.

Denneberg and Fromm attempted to develop open software for autonomous mobile robots. The open software concept for autonomous robots (OSCAR) is based on a layered model with four software levels: command layer, execution layer, image layer, and hardware layer (Jamshidi, 2009).

Controlling many cooperative robots is no easy task. The software needed is complex and must allow multitasking. Many developers have begun using real-time operating systems (RTOSs) to mitigate the complexity. In addition to their precise synchronization of multiple events, RTOSs provide the application programmer with predefined system services and varying degrees of hardware abstraction, both of which aim to make software development easier and more organized (Jamshidi, 2009).

4.2. Challenges and opportunities

A well-known problem with self-adaptive systems is that users may not understand or trust them. Such a lack of intelligibility can mean that users may cease to use a self-adaptive system. One well-studied approach to addressing this problem is to provide human-readable explanations of adaptive behavior. The intuition is that if users can query the system's decisions, they are is less likely to abandon it and may, indeed, accept the system's choices over their own, which may not be based on a full understanding of the system and the context (Sawyer et al., 2010).

Yu, Threm, and Ramaswamy (2011) suggested that, as control theory has been successfully applied in robotics, self-adaption, self-management, and additional areas, it is time to consider combining control theory with complex systems theory to produce real-time intelligence in software systems. They also point out the importance of studying how a software agent or system could adjust its structure and response behavior to fit the natural environment better, by exhibiting: adaptation, cooperation, or, self-organization, evolution, and emergence.

Researchers in the self-adaptive software domain subscribe to the view that deciding is a vital process (among others pointed out by Oreizy et al. (1999)), and that decision-making remains a challenge, as noted by Salehie and Tahvildari (2005) and McKinley (2004). Existing solutions for self-adaptive software, and more broadly for autonomic systems, have still not addressed all the requirements of a desired decision-making process. Maes enumerates several specific requirements in Maes (1994)): finding good enough actions, minimizing back and forth switching between actions that contribute to distinct goals, and never getting stuck in a loop or deadlock situation to satisfy an unattainable goal, are some of these requirements (Salehie & Tahvildari, 2011).

However, no single set of criteria and metrics exists for verifying that a solution complies with these requirements. Gjorven et al. (2006) discussed the applicability of some quality factors for adaptation, but there are notable difficulties for evaluating most of the quality aspects of adaptation (Salehie & Tahvildari, 2011).

Related to the area of producing real-time intelligence in software systems Yu et al. (2011) have devised a challenge for understanding how global features and structures emerge from simple local interactions and how new levels of components are formed within complex natural processes and the development of social systems.

Building self-adaptive software systems in a cost-effective and predictable manner is a major engineering challenge. New theories are needed to accommodate engineering procedures, in systematic, traditional top-down approaches and bottom-up approaches. A promising starting point to meet these challenges is to bring together suitable control methods and theoretical approaches for the creative design of self-adaptive software systems.

4.3. How to go beyond the current state of the art

One of the trends driving researchers and practitioners is multi-layered architectures (i.e., applications, middleware, network and operating system infrastructure), focused on application composition from reusable components. This middleware-centric, multi-layered architecture descends directly from the adoption of a network-centric viewpoint brought about by the emergence of the Internet and the componentization and commoditization of hardware and software (Schantz & Schmidt, 2002).

The new generation of architectures should enable desired future development, by rapid prototyping, development and deployment of on demand services, with the purpose of enhancing flexibility, communication performance, robustness, and scalability (Letaifa, Haji, Jebalia, & Tabbane, 2010).

While self-adaptive software systems are being adopted in more domains, it is imperative to re-formulate, or evolve current methods and tools related to feedback loops. A clear target is therefore to hybridize concepts and method from control engineering, artificial intelligence, computer science and cybernetics, to be jointly adapted and applied to software-intensive self-adaptive systems.

We would emphasize that feedback control loops, although implicitly contained in many existing software systems, have to be made more explicit, before they become a first-class citizen of architecture, design, and infrastructure support (Müller et al., 2008).

On the topic of requirements engineering, we propose to explore the use of non-uniform methods of requirements engineering, in order to deal with non-homogeneous contexts. In consequence, we outlined a diagram of context types versus methods, as shown in Fig. 5.

Operational (real-time) decision-making is supported by two sets of technologies (i.e., information and decision technologies) and underpinned by three disciplines: data fusion/analysis, decision modeling, and systems engineering (Tien, 2009). Data fusion/analysis methods include data mining, visualization, data management, probability, statistics, quality, reliability, fuzzy logic, multivariable testing, and pattern analysis; however, real-time data fusion/analysis is more complex and requires additional research. Decision-modeling methods include discrete simulation, finite element analysis, stochastic methods, neural networks, genetic algorithms, optimization, and so on; however, real-time decision modeling, like real-time data fusion/analysis, also requires additional research, especially since all steady-state models become irrelevant in a real-time environment. Systems engineering includes cybernetics or feedback and control; it integrates people, processes and products from a holistic perspective, especially human-centered systems that are computationally-intensive and intelligence-oriented. Similarly, undertaking systems engineering within a real-time environment requires additional thought and research.

More extensive research in decision-making for adaptive software is needed. In particular, further study of decision-making optimization would help it evolve; although there is a solid theoretical basis (classical, fuzzy logic, etc.) there is a lack of integration of this knowledge for self-adaptive software systems. For instance, an algorithm based on self-adaptive particle swarm optimization (MSSE-SPSO), which combines a particle swarm optimization with a self-adaptive evolution strategy is suggested in Jiang, Li, and

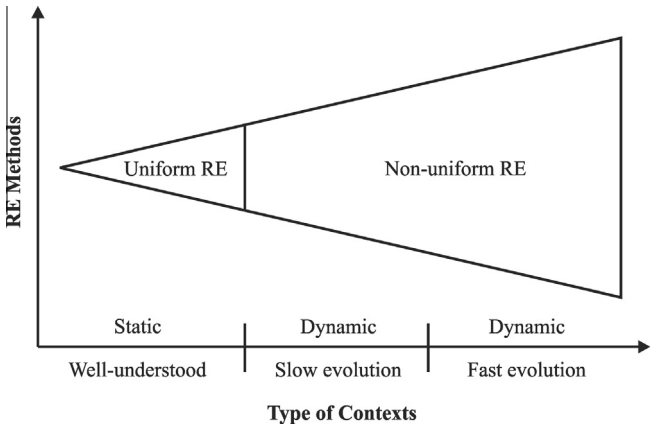

Fig. 5. Context types continuum vs. requirement engineering methods.

Huang (2013). However, the stability of self-adaptive and self-organizing systems is a bottleneck and new Lyapunov-based approaches are nowadays applied to guarantee stability (Lian, 2012).

Future systems must accommodate high-dimensional sensory data, must continue to learn from new experiences and take advantage of new adaptations as they become available (McKinley, 2004). Moreover, the use of simulated models could be devised as one possible solution to face the lack of case studies of self-adaptive applications.

5. Concluding remarks

This review has presented, from a computer science point of view, concepts, methods and challenges in the new field of self-adaptive software systems, in a simple and systematic way. An understanding of this new and challenging topic will certainly help to promote their practical application and promotion by the software engineering and computer science communities.

Through this literature review of self-adaptivity, we have looked at some theoretical and practical approaches for the achievement of a new generation of software systems and have identified some key issues for us to move beyond the state of the art. Feedback control and artificial intelligence techniques are identified as two of the enabling disciplines that will favor the development of a new generation of self-adaptive systems. This paper has sought to lay the foundations for self-adaptative software engineering as a mature field, which can harness existing systems and will not solely rely on technological improvements for its progress.

Acknowledgement

This work was supported by the European Commission in the ARTEMIS JTI 295372 "Design, Monitoring and Operation of Adaptive Networked Embedded Systems (DEMANES)". We are also grateful to Antony Price for his assistance with the editing of this paper. This work was also supported by the Ministerio de Economia y Competitividad [Ministry of Economy and Competitiveness] through its DPI2012-35504 CONMICRO research project.

References

Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Thomas, F., et al. (2000). Amorphous computing. *Communications of the ACM, 43*, 74–82.

Arkin, R. C. (1998). *Behavior-based robotics*. Cambridge, MA: MIT Press.

Batori, G., Theisz, Z., & Asztalos, D. (2012). Metamodel based methodology for dynamic component systems. In (Vol. 7349 LNCS, pp. 275–286).

Bencomo, N. (2009). On the use of software models during software execution. In *Proceedings of the 2009 ICSE workshop on modeling in software engineering* (pp. 62–67): IEEE Computer Society.

Bencomo, N., Grace, P., Flores, C., Hughes, D., & Blair, G. (2008). Genie: supporting the model driven development of reflective, component-based adaptive systems. In *Proceedings of the 30th international conference on Software engineering* (pp. 811–814). Leipzig, Germany: ACM.

Bencomo, N., Grace, P., & Sawyer, P. (2009). Revisiting the Relationship between Software Architecture and Requirements: The case of Dynamically Adaptive Systems. In D. Weyns, S. Malek, R. d. Lemos & J. Andersson (Eds.), *WICSA/ECSA workshop on self-organizing architectures. SOAR 2009*. Cambridge, UK.

Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., & Letier, E. (2010). Requirements reflection: requirements as runtime entities. In *32nd ACM/IEEE international conference on software engineering* (Vol. 2, pp. 199–202). Cape Town, South Africa.

Bennett, S. (1993). *A history of control engineering, 1930–1955*. Peter Peregrinus Ltd. (Vol. 47).

Bochow, B., & Emmelmann, M. (2011). *Purpose-driven, self-growing networks – A framework for enabling cognition in systems of systems*. Budapest.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*, 61–72.

Bong Seok, P., Jin-Bae, P., & Yoon-Ho, C. (2011). Adaptive formation control of electrically driven nonholonomic mobile robots with limited information. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 41*, 1061–1075.

Brown, G., Cheng, B. H. C., Goldsby, H., & Zhang, J. (2006). Goal-oriented specification of adaptation requirements engineering in adaptive systems. In *Proceedings of the 2006 international workshop on self-adaptation and self-managing systems* (pp. 23–29). Shanghai, China: ACM.

Brun, Y., Di Marzo Serugendo, G., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezzè, M., & Shaw, M. (2009). Engineering self-adaptive systems through feedback loops. In B. Cheng, R. de Lemos, H. Giese, P. Inverardi & J. Magee (Eds.), *Software engineering for self-adaptive systems* (Vol. 5525, pp. 48–70). Springer Berlin/Heidelberg.

Brun, Y., & Medvidovic, N. (2007a). An architectural style for solving computationally intensive problems on large networks. In *Software engineering for adaptive and self-managing systems, 2007*. International Workshop on ICSE Workshops SEAMS'07 (pp. 2–2): IEEE.

Brun, Y., & Medvidovic, N. (2007b). Fault and adversary tolerance as an emergent property of distributed systems' software architectures. In *Proceedings of the 2007 workshop on engineering fault tolerant systems* (pp. 7). Dubrovnik, Croatia: ACM.

Chen, T. (2011). A self-adaptive agent-based fuzzy-neural scheduling system for a wafer fabrication factory. *Expert Systems with Applications, 38*, 7158–7168.

Cheng, S.-W., Garlan, D., & Schmerl, B. (2005). Making self-adaptation an engineering reality. In B. Ozalp, J. Rk, M. Alberto, F. Christof, & L. Stefano (Eds.), *Self-star properties in complex information systems* (pp. 158–173). Springer-Verlag.

Choonhwa, L., Nordstedt, D., & Heal, S. (2003). *Enabling smart spaces with OSGi*. University of Florida.

Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2000). *Non-functional requirements in software engineering*. Berlin: Springer.

Clement, L., & Nagpal, R. (2003). Self-assembly and self-repairing topologies. In *Workshop on adaptability in multi-agent systems*, RoboCup Australian Open.

Córdoba, A., Astrain, J. J., Villadangos, J., & Echarte, F. (2013). A self-adapted method for the categorization of social resources. *Expert Systems with Applications, 40*, 3696–3714.

Couture, M., & Valcartier, D. (2007). *Complexity and chaos-State-of-the-art: Overview of theoretical concepts*. Canada: Minister of National Defence.

Dardenne, A., van Lamsweerde, A., & Fickas, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming, 20*, 3–50.

Den Hamer, P., & Skramstad, T. (2011). Autonomic service-oriented architecture for resilient complex systems. In (pp. 62–66). Madrid.

Di Marzo-Serugendo, G., Fitzgerald, J., Romanovsky, A., et al. (2007). A generic framework for the engineering of self-adaptive and self-organising systems. *Organic Computing-Controlled Self-Organization*, 165–189.

Di Marzo-Serugendo, G., Gleizes, M. P., & Karageorgos, A. (2005). Self-organisation and emergence in multi-agent systems. *The Knowledge Engineering Review, 20*, 165–189.

Diao, Y., Hellerstein, J. L., Parekh, S., Griffith, R., Kaiser, G. E., & Phung, D. (2005). A control theory foundation for self-managing computing systems. *IEEE Journal on Selected Areas in Communications, 23*, 2213–2222.

DOC-Group, The ADAPTIVE Communication Environment (ACE), http://www.cse.wustl.edu/~schmidt/ACE.html.

Dustdar, S., Goeschka, K. M., Truong, H.-L., & Zdun, U. (2009). Self-adaptation techniques for complex service-oriented systems. In *Next Generation Web Services Practices, 2009. NWESP '09. Fifth International Conference on* (pp. 37–43).

Franklin, G. F., Powell, J. D., & Emami-Naeini, A. (2006). *Feedback control of dynamic systems*. Englewood Cliffs, NJ: Prentice Hall.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., & Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer, 37*, 46–54.

Garlan, D., Cheng, S.-W., & Schmerl, B. (2003). Increasing system dependability through architecture-based self-repair. In L. Rog, G. rio De, & Alexander Cristina R. (Eds.), *Architecting dependable systems* (pp. 61–89). Springer-Verlag.

Geihs, K., Barone, P., Eliassen, F., Floch, J., Fricke, R., Gjorven, E., et al. (2009). A comprehensive solution for application-level adaptation. *Software: Practice and Experience, 39*, 385–422.

Gjorven, E., Eliassen, F., & Aagedal, J. O. (2006). Quality of adaptation. In *Autonomic and Autonomous Systems, 2006. ICAS '06. 2006 International Conference on* (pp. 9–14).

Gu, T., Pung, H. K., & Zhang, D. Q. (2004). Toward an OSGi-based infrastructure for context-aware applications. *IEEE Pervasive Computing, 47*, 54–87.

Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., & Jansen, E. (2005). The Gator Tech Smart House: a programmable pervasive space. *IEEE Computer, 45*.

Hellerstein, J., Parekh, S., Diao, Y., & Tilbury, D. M. (2004). *Feedback control of computing systems*. Wiley-IEEE Press.

Hüfner, M., Fischer, S., Sonntag, C., & Engell, S. (2012). Integrated Model-Based Support for the Design of Complex Controlled Systems. In (Vol. 31, pp. 1672–1676).

Hung, P.-C., & Lin, S.-F. (2012). The partial solutions consideration based self-adaptive evolutionary algorithm: A learning structure of neuro-fuzzy networks. *Expert Systems with Applications, 39*, 10749–10763.

IBM-Corporation. (2006). An architectural blueprint for autonomic computing, http://www-03.ibm.com/autonomic/pdfs/ACwp-Final.pdf. In.

Ilter, D. A. (2011). A system of systems (SoS) approach to dispute management systems. In *Proceedings of the CIB W78-W102 2011: International Conference*. Sophia Antipolis, France.

Jamshidi, M. (2008). System of systems engineering – new challenges for the 21st century. In *Aerospace and Electronic Systems Magazine, IEEE* (Vol. 23, pp. 4–19).

Jamshidi, M. (2009). *System of systems engineering: innovations for the 21st century* (Vol. 58): John Wiley & Sons Inc.

Jiang, Y., Li, X., & Huang, C. (2013). Automatic calibration a hydrological model using a master–slave swarms shuffling evolution algorithm based on self-adaptive particle swarm optimization. *Expert Systems with Applications, 40*, 752–757.

Jonghwa, C., Dongkyoo, S., & Dongil, S. (2005). Research and implementation of the context-aware middleware for controlling home appliances. *IEEE Transactions on Consumer Electronics*.

Karnousko, S., Colombo, A. W., Jammes, F., Delsing, J., & Bangemann, T. (2010). *Towards an Architecture for Service-Oriented Process Monitoring and Control*.

Kramer, J., & Magee, J. (2007). *Self-managed systems: An architectural challenge*. In (pp. 259–268).

Leitao, P. (2009). Holonic rationale and bio-inspiration on design of complex emergent and evolvable systems. *Transactions on Large-Scale Data- and Knowledge-Cent. Systems*, 243–266.

Leitão, P. (2008). A bio-inspired solution for manufacturing control systems. *Innovation in manufacturing networks*, 303–314.

Letaifa, A. B., Haji, A., Jebalia, M., & Tabbane, S. (2010). State of the Art and Research Challenges of new services architecture technologies: Virtualization, SOA and Cloud Computing. *International Journal of Grid and Distributed Computing, 3*, 545–654.

Li, C., & Li, L. (2012). Design and implementation of economics-based resource management system in ad hoc grid. *Advances in Engineering Software, 45*, 281–291.

Lian, R.-J. (2012). Design of an enhanced adaptive self-organizing fuzzy sliding-mode controller for robotic systems. *Expert Systems with Applications, 39*, 1545–1554.

Liu, H., & Parashar, M. (2006). Accord: a programming framework for autonomic applications. *Trans. Sys. Man Cyber Part C, 36*, 341–352.

Maes, P. (1994). Modeling Adaptive Autonomous Agents. *Artificial Life, 1*, 135–162.

Magee, J., & Kramer, J. (1996). Dynamic structure in software architectures. In *Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering* (pp. 3–14). San Francisco, California, United States: ACM.

McKinley, P. K. (2004). Composing adaptive software. In S. Seyed Masoud, P. K. Eric & H. C. C. Betty (Eds.), (Vol. 37, pp. 56–64).

Microsoft, Microsoft .NET Framework, http://www.microsoft.com/NET.

Mishra, A., & Misra, A. K. (2009). Component assessment and proactive model for support of dynamic integration in self adaptive system. *SIGSOFT Software Engineering Notes, 34*, 1–9.

Mittal, S. (2007). *DEVS unified process for integrated development and testing of service oriented architectures*. University of Arizona.

Mittal, S., Risco-Martin, J. L., & Zeigler, B. P. (2007a). DEVS-BasedWeb Services for Net-centric T&E. In *Summer computer simulation conference*.

Mittal, S., Risco-Martin, J. L., & Zeigler, B. P. (2007b). DEVSML: Automating DEVS execution over SOA using transparent simulators. In *SpringSim* (Vol. 2, pp. 287–295). Norfolk, Virginia, USA.

Mittal, S., Zeigler, B. P., Risco Martín, J. L., Sahin, F., & Jamshidi, M. (2009). Modeling and Simulation for Systems of Systems Engineering. In M. Jamshidi (Ed.), *System of Systems Engineering: Innovations for the 21st Century* (pp. 101–149). Hoboken, New Jersey: John Wiley & Sons.

Müller, H., Pezzè, M., & Shaw, M., 2008. Visibility of control in adaptive systems. In *Second international workshop on ultra-large-scale software-intensive systems (ULSSIS 2008), ICSE 2008 Workshop*.

Nakagawa, H., Ohsuga, A., & Honiden, S., 2008. Constructing Self-Adaptive Systems Using a KAOS Model. In *Proceedings of the 2008 second IEEE international conference on self-adaptive and self-organizing systems workshops* (pp. 132–137): IEEE Computer Society.

Nakagawa, H., Ohsuga, A., & Honiden, S. (2010). Cooperative behaviors description for self-$*$ systems implementation advances in practical applications of agents and multiagent systems. In Y. Demazeau, F. Dignum, J. Corchado & J. Pérez (Eds.), (Vol. 70, pp. 69–74): Springer Berlin/Heidelberg.

Naqvi, M., (2012). Claims and supporting evidence for self-adaptive systems – A literature review. In (pp. 47).

Oracle, Java 2 Enterprise Editon, http://java.sun.com/j2ee.

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimhigner, D., Johnson, G., & Medvidovic, N. (1999). An architecture-based approach to self-adaptive software. *Intelligent Systems and their Applications, IEEE, 14*, 54–62.

OSGi-Alliance, (2012). OSGi Service Platform, Core Specification. Release 5, http://www.osgi.org. In. USA.

Peng, X., Chen, B., Yu, Y., & Zhao, W. (2010). Self-tuning of software systems through goal-based feedback loop control. In *Proceedings of the 2010 18th IEEE international requirements engineering conference* (pp. 104–107): IEEE Computer Society.

Peper, C., & Schneider, D. (2008). Component engineering for adaptive ad-hoc systems. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems* (pp. 49–56). Leipzig, Germany: ACM.

Petković, D., Pavlović, N. D., Ćojbašić, Ž., & Pavlović, N. T. (2013). Adaptive neuro fuzzy estimation of underactuated robotic gripper contact forces. *Expert Systems with Applications, 40*, 281–286.

Pokahr, A., Braubach, L., & Lamersdorf, W. (2003). Jadex: Implementing a BDI-infrastructure for JADE agents. *EXP – in search of innovation (Special Issue on JADE), 3*, 76–85.

Rao, A., & Georgeff, M. (1995). BDI agents: From theory to practice. In C. The MIT Press (Ed.), *Proceedings of the international conference on multi-agent systems* (pp. 312–319). San Francisco, CA, USA.

Ravindranathan, M., & Leitch, R. (1998). Heterogeneous intelligent control systems. In *IEE Proceedings – Control Theory and Applications* (Vol. 14S, pp. 551–558).

Richter, U. M. M., Branke, J., Müller-Schloer, C., & Schmeck, H. (2006). Towards a generic observer/controller architecture for Organic Computing. In C. Hochberger & R. Liskowsky (Eds.), *INFORMATIK 2006: Informatik für Menschen. GI-Edition – Lecture Notes in Informatics* (Vol. 93, pp. 112–119): GI.

Rogers, A., Jennings, N. R., & Farinelli, A. (2009). Self-organising sensors for wide area surveillance using the max-sum algorithm. In D. Weyns, S. Malek, R. d. Lemos & J. Andersson (Eds.), *WICSA/ECSA workshop on self-organizing architectures. SOAR 2009*. Cambridge, UK.

Roy, B. (1996). *Multicriteria Methodology for Decision Aiding: Kluwer Academic Publishers*. Dordrecht.

Royce, W. W. (1970). Managing the development of large software systems. In *Proceedings of the 9th International Conference on Software Engineering* (pp. 328–338). Monterey, California, United States: IEEE Computer Society Press.

Sahin, S., Tolun, M. R., & Hassanpour, R. (2012). Hybrid expert systems: A survey of current approaches and applications. *Expert Systems with Applications, 39*, 4609–4617.

Salehie, M., & Tahvildari, L. (2005). Autonomic computing: emerging trends and open problems. In *Proceedings of the 2005 workshop on Design and evolution of autonomic application software* (pp. 1–7). St. Louis, Missouri: ACM.

Salehie, M., & Tahvildari, L. (2011). *Towards a goal-driven approach to action selection in self-adaptive software*. Software - Practice and Experience.

Sawyer, P., Bencomo, N., Whittle, J., Letie, E., & Finkelstein, A. (2010). Requirements-aware systems: A research agenda for RE for self-adaptive systems. In (pp. 95–103). Sydney, NSW.

Schantz, R. E., & Schmidt, D. C. (2002). Research Advances in Middleware for Distributed Systems. In *Proceedings of the IFIP 17th world computer congress-TC6 stream on communication systems: The state of the art* (pp. 1–36): Kluwer, BV.

Schmeck, H., Müller-Schloer, C., Cakar, E., Mnif, M., & Richter, U. (2010). Adaptivity and Self-Organization in Organic Computing Systems. *ACM Transactions on Autonomous and Adaptive Systems, 5*, 10:11–10:32.

Schmitt, J., Roth, M., Kiefhaber, R., Kluge, F., & Ungerer, T. (2011). Realizing self-x properties by an automated planner. In (pp. 185–186). Karlsruhe.

Shen, W. M., Krivokon, M., Chiu, H., Everist, J., Rubenstein, M., & Venkatesh, J. (2006). Multimode locomotion via SuperBot reconfigurable robots. *Autonomous Robots, 20*, 165–177.

Sridhar, P., Madni, A. M., & Jamshidi, M. (2009). Advances in Wireless Sensor Networks: A Case Study in System of Systems Perspective. In M. Jamshidi (Ed.), *System of Systems Engineering: Innovations for the 21st Century* (pp. 275–292). Hoboken, New Jersey: John Wiley & Sons.

Tanner, J. (1963). Feedback control in living prototypes: A new vista in control engineering. *Medical and Biological Engineering and Computing, 1*, 333–351.

Telecom-Italia, JADE, http://jade.tilab.com/.

Tien, J. M. (2009). A System of Systems View of Services. In M. Jamshidi (Ed.), *System of Systems Engineering: Innovations for the 21st Century* (pp. 293–316). Hoboken, New Jersey: John Wiley & Sons.

Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., & Becker, B. (2009). Model-driven architectural monitoring and adaptation for autonomic systems. In *Proceedings of the 6th international conference on Autonomic computing* (pp. 67–68). Barcelona, Spain: ACM.

W3C. (February 2007). State Chart XML (SCXML): State Machine Notation for Control Abstraction, http://www.w3.org/TR/scxml/.

Weiss, G., Becker, K., Kamphausen, B., Radermacher, A., & Gérard, S. (2011). Model-driven development of self-describing components for self-adaptive distributed embedded systems. In (pp. 477–484). Oulu.

Weyns, D., & Georgeff, M. (2010). Self-adaptation using multiagent systems. *IEEE Software, 27*, 86–91.

Wu, Q. (2011). A self-adaptive embedded chaotic particle swarm optimization for parameters selection of Wv-SVM. *Expert Systems with Applications, 38*, 184–192.

Yeom, K., & Park, J. H. (2012). Morphological approach for autonomous and adaptive systems based on self-reconfigurable modular agents. *Future Generation Computer Systems, 28*, 533–543.

Yu, E. (1995). *Modelling strategic relationships for process reengineering*. University of Toronto.

Yu, L., Ramaswamy, S., & Bush, J. (2008). Symbiosis and software evolvability. *IT Professional, 10*, 56–62.

Yu, L., Threm, D., & Ramaswamy, S. (2011). Toward evolving self-organizing software systems: A complex system point of view. In (Vol. 6704 LNAI, pp. 336–346). Syracuse, NY.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control, 8*, 338–353.

Zeigler, B. P., Kim, T. G., & Praehofer, H. (2000). *Theory of Modeling and Simulation*. New York: NY.