On the Challenges of Self-Adaptation in Systems of Systems

Danny Weyns and Jesper Andersson
Department of Computer Science
Linnaeus University, Sweden
{danny.weyns, jesper.andersson}@lnu.se

ABSTRACT

A system of systems (SoS) integrates independently useful systems into a larger system. Examples are integrated surveillance systems and networked smart homes. A SoS offers functions to users that cannot be provided by its individual parts, but emerge as a combination of these. However, providing these functions with a required level of quality is difficult due to inherent uncertainties, such as systems that attach and detach at will and faults that are difficult to predict. Self-adaptation is a well-studied approach that enables a system to reason about itself and adapt to achieve particular quality objectives in the face of uncertainties and change. However, the inherently decentralized nature of SoS raises fundamental challenges to self-adaptation. This paper presents three architectural styles to realize self-adaptation in SoS, discusses key challenges for each style, and outlines starting points that could help to tackle these challenges.

Categories and Subject Descriptors

D.2.11 [**Software engineering**]: Software architecture

General Terms

Design, management

Keywords

Systems of systems, self-adaptation, feedback loops, decentralized control

1. INTRODUCTION

A system of system (SoS) integrates independently useful systems into a larger system, delivering unique functions to users that emerge from the combination of the individual parts. Examples are integrated surveillance systems, intelligent traffic systems, and networked smart homes. Engineering SoS and guaranteeing runtime qualities (performance, reliability, etc.) that span beyond individual systems is complex due to a variety of uncertainties. Examples of such uncertainties are systems that attach and detach at will, dynamically changing availability of resources, and faults and intrusions that are difficult to predict. Self-adaptation is a well-studied approach that enables a software system to reason about itself and adapt autonomously to achieve particular quality objectives in the face of uncertainties and change. Central to the realization of self-adaptation are feedback loops that monitor and adapt managed parts of a system when needed. However, state of the art in the field of self-adaptation has primarily studied centralized and hierarchical control in self-adaptation, which is not applicable to systems that are inherently decentralized. Realizing self-adaptation in a SoS where no single entity has the knowledge and authority to supervise and adapt the constituent parts raises fundamental engineering challenges.

In this paper, we discuss a number of the key challenges to enable self-adaptation of SoS. Section 2 starts by clarifying our perspective on SoS and self-adaptation. Next, we present three architectural styles to realize self-adaptation in SoS in Section 3. For each style, we give an illustrative example, identify a number of challenges, and provide some starting points that could help tackling the challenges. Finally, we draw conclusions in Section 4.

2. SYSTEMS OF SYSTEMS AND SELF-ADAPTATION

Before we zoom in on challenges, we first clarify the key terms we use this paper: SoS and self-adaptation.

2.1 Systems-of-Systems

There is no commonly agreed definition for what constitutes a SoS. [23] refers to SoS or federations of systems (FOS) or federated systems of systems (F-SOS) as systems that possess characteristics of complex adaptive systems. [4] focuses on the nature of the composition to define the distinguishing characteristics of SoS, including autonomy, connectivity, diversity and emergence. [21] stresses scale and complexity as central properties of ultra-large scale systems, phrased by the slogan *scale changes everything.* In his influential paper on architecting principles of SoS, Maier [20] characterizes a SoS (he also uses the term collaborative system) as an assembly of components which individually may be regarded as systems and which possess two required properties:

- *Operational independence of the components*: If the SoS is disassembled into its component systems these systems must be able to usefully operate independently.
- *Managerial independence of the components*: The com-

ponent systems are separately acquired and integrated but maintain a continuing operational existence independent of the SoS.

Based on this characterization, Maier identifies a set of guiding design principles for SoS:

- *Stable intermediate forms*: individual systems or subsets of systems of a SoS should be capable of operating and fulfilling useful purposes, before full deployment and during operation.
- *Policy triage*: A SoS design team should carefully choose what to control; over-control will fail for lack of authority, under-control will eliminate the integrated nature of the SoS.
- *Leverage at the interfaces*: The architecture of a SoS is essentially defined by its interfaces, which are the primary points at which designers can exert control.
- *Ensuring collaboration*: Mechanisms should be exploited that create joint utility, which is known to be a basis for consistent behavior.

2.2 Self-Adaptation

Self-adaptation has been widely recognized as an effective approach to deal with the increasing complexity and dynamicity of modern software systems [22, 17, 6, 19]. A self-adaptive system comprises two parts: the managed system (also called system layer [10], managed resources [16], base-level subsystem [28], target system [13]) that deals with the domain functionality, and the managing system (or architecture layer [10], autonomic manager [16], reflective subsystem [28], controller [13]) that deals with the adaptations of the managed system to achieve particular quality goals.

One influential approach to structure the managed system is by means of four components that realize a feedback loop: Monitor-Analyze-Plan-Execute supported by a Knowledge repository [16] (MAPE-K). Figure 1 shows the elements of a MAPE-K system.

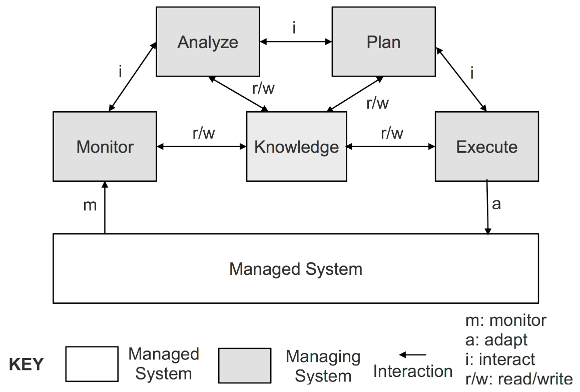

Figure 1: Elements of a MAPE-K managed system

A monitor component gathers information from the managed system and possibly the system's environment to update a set of models of the knowledge repository. An analyze component examines the gathered data and based on the adaptation goals draws conclusions on whether further actions should be undertaken. A plan component puts together a series of adaptation actions to resolve the problem identified by an analyze component. The actions to the managed system are then carried out by an execution component. Examples of MAPE-K based approaches are the Rainbow framework [10] that employs constraints defined over an architectural model of the managed system to realize self-adaptation, and K-Components [9] that reifies a system's component architecture as a configuration graph that can be rewritten by a configuration manager to adapt the system when needed.

Another well-studied approach to realize the managed system is by means of a controller. Figure 2 shows the elements of a closed loop control system. The target system is the

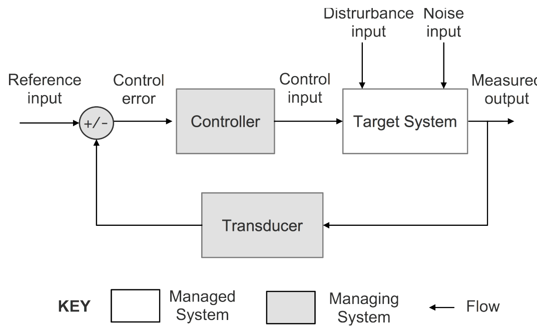

Figure 2: Elements of a closed loop control system

managed system. The measured output is the subject of control. The disturbance input is any change that affects the measured output and for which adaptation is required. Noise input affects the measured output produced by the target system. The reference input is the desired value of the measured output, and the control error is the difference between reference input and measured output. The transducer transforms the measured output so that it can be compared by the reference input. Based on the control error, the controller determines the setting of the control input that manipulates one or more parameters of the target system. An example of control-based self-adaptation for high-performance servers is described in [2]. Servers are modeled as difference equations and different types of controllers (e.g. Proportional, Integral) are applied to deal with performance requirements (e.g., server response time, convergence). [8] employs multi-input multi-output techniques for controlling a Web server. System models are derived from experiments and the controller optimizes CPU and memory usage based on a cost function.

3. ARCHITECTURAL STYLES OF SELF-ADAPTATION FOR SOS

As the constituent systems of a SoS are independently developed and operated, SoS are inherently decentralized systems. The SoS architect has to express the overall architecture through the specification of the communication elements between abstractions of the constituent systems of the SoS. In general, this requires well-defined communication protocols at different levels of the technology stack.

To deal with particular quality requirements, a managing layer can be added to a SoS, resulting in a self-adaptive SoS. The typical architecture of a self-adaptive SoS thus consists of a set of interacting managed systems that are controlled by local feedback loops. For SoS, in general no assumptions can be made about the presence of systems, availability of external resources, prediction of faults, etc. To deal with these uncertainties, a key challenge is to provide guarantees for properties that span multiple systems of the SoS. These properties refer to the adaption requirements and other behavioral aspects such as stability and transient behavior. Handling uncertainties is currently subject of active research in the field of self-adaptation. SoS add another dimension of complexity to the problems of uncertainty due to their inherent decentralized nature.

Figure 3 shows three basic architecture styles to structure the managing layer of a SoS. We derived the styles from classic control architectures, see e.g. [3], and generalizations over concrete architectural patterns for decentralized self-adaptation described in [29]. The three styles provide increasing levels of knowledge sharing and collaboration, allowing to mitigate uncertainty at different scales.

We now discuss the three styles. We give illustrative examples from our practice, identify a number of challenges, and provide some starting points that could help tackling the challenges. Although the different examples do not fully comply to Maier's properties of a SoS, they are decentralized systems where self-adaptation is realized with the different styles. As such they can serve as basic examples for the application of three architectural styles in SoS settings.

3.1 Local Adaptations

The first style, *local adaptations* represents a fully decentralized adaptation architecture. In this style, feedback loops do not coordinate directly, but, typically there will be indirect interactions. E.g., a local feedback loop may affect the response time of the local managed system, triggering other feedback loops to adapt. In a recently started project, called CareSmart, we study the application of smart home technology to provide innovative services for elderly care living in their own houses. The architecture of the system consists of smart home systems that collect and synthesize sensor data at the homes of the elderly people. Useful data is sent to mobile care assistant systems that welfare helpers can use to make decisions about visits and interact with the elderly or other persons when needed. This collaborative system provides different types of local adaptations. For example, each smart home system is provided with a context adaptor that detects changes in the context and dynamically adapts services based on the preferences of the elderly. E.g., the adaptor may activate a service that enables an elderly to alarm a relative via voice when he/she enters the bathroom without having the alarm with him/her. Mobile care assistants also have a context-adaptor that activates for example a service that provides specific information of an elderly once the welfare helper approaches his or her home. Initial results of the CareSmart project are reported in [18].

In the local adaptations style the design problem of self-adaption for a SoS boils down to the design of local feedback loops. However, this style provides limited support to each of the guiding design principles for SoS proposed by Maier. As feedback loops share no information with each other, there is a high-degree of uncertainty regarding other systems and the environment. Consequently, the approach is sensitive to side-effects of indirect interactions between individually well-optimized systems, as well as emergent behavior that results from interactions between the compositions of systems. In

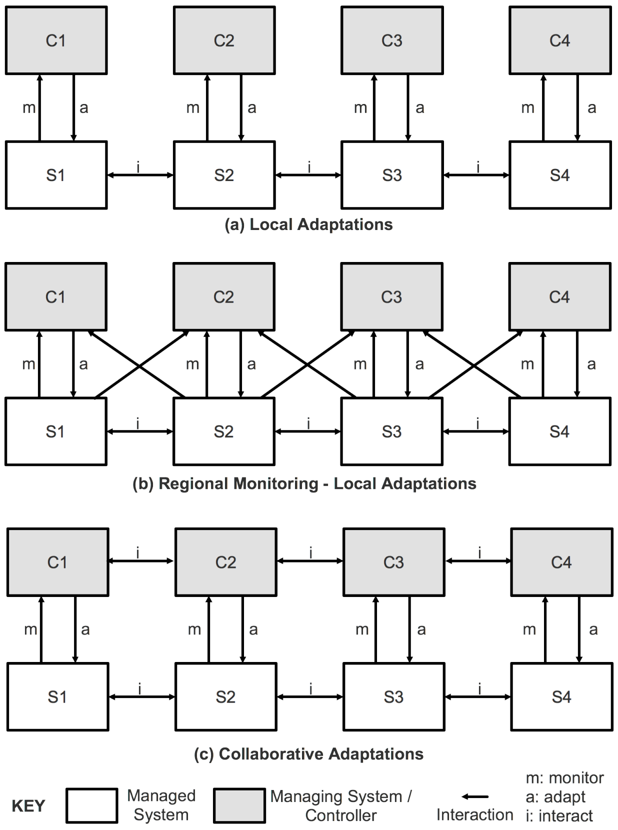
C1
C2
C3
C4
m
a
i
S1
S2
S3
S4
(a) Local Adaptations
(b) Regional Monitoring - Local Adaptations
(c) Collaborative Adaptations
KEY
Managed System
Managing System / Controller
Interaction
m: monitor
a: adapt
i: interact

Figure 3: Basic architectural styles of decentralized self-adaptation in SoS

the CareSmart project for example, the activation of context-dependent services by elderly people may trigger a series of reschedules of visit plans of mobile care assistants. To deal with the uncertainties of local adaptations, [1] points to the need for analysis tools that allow to understand and quantify the effects of indirect interactions, as well as runtime support for dynamic verification of the design assumptions coupled with appropriate actions when violations are detected. A number of interesting approaches have been proposed to analyze properties of a SoS (e.g., stability) based on the local adaptations style. Examples are analyses based on an integrated transfer function of a (partial) composition of systems [27] and grouping of local controllers [26]. Interesting fields that provide various techniques that can potentially be used for the analysis of properties of SoS designed with the local adaptations style are complex system theory and economics [14]. Examples are analysis based on the principles of entropy, Pareto efficiency, and the Nyquist stability.

3.2 Regional Monitoring–Local Adaptations

The second style, *regional monitoring–local adaptations* enables local feedback systems to gather runtime data from neighboring systems to support local decision making of adaptations. This data can be exploited to reduce the potential side effects of purely decentralized adaptation architecture. [24] discusses an example application where we have applied the second style. The application in an intelligent traffic monitoring system that provides information about traffic jams to clients, such as traffic light controllers, driver assistance systems, etc. The traffic monitoring system consists of intelligent cameras that are distributed along the road. If a traffic jam spans the viewing range of multiple cameras, they form an organization that provides information to clients, realizing scalability. The organizations dynamically adapt when the traffic conditions change. To deal with camera failures, each local camera system is extended with a MAPE-K loop that monitors the status of the cameras on which the local camera depends, i.e., the neighbors and the camera's of the organization to which the local camera belongs.

In the regional monitoring–local adaptations style the design of self-adaptation requires the design of local feedback loops and the definition of monitoring interfaces to collect data from other local managed systems. This style provides a better match with Maier's guiding design principles for SoS. Gathering data from neighboring systems helps reducing uncertainty and as such reduces the sensitivity to side-effects of indirect interactions between local systems. The tradeoff is an increased dependency between systems. For example, in the traffic monitoring system, the cameras within an organization share their status which supports efficient reorganizations when traffic conditions change or a camera fails. Nevertheless, support for stability, division of control, and collaboration remains limited. In the traffic monitoring system, we employed model checking techniques to verify robustness properties of both local systems and organizations. However, this approach is restricted to guarantee properties that are known at design time. A key challenge is to provide guarantees for properties based on information that can only be acquired at runtime. Promising approaches to deal with runtime uncertainties in SoS are the exploitation of testing and verification at runtime. Recent work in this direction, although not applied to SoS yet (according to Maier's properties), is presented in [5] that applies runtime verification of probabilistic models in a MAPE-K setting, and [7] that applies runtime testing to support inter-operability. Another, complementary approach to mitigate uncertainties is to exploit learning techniques. An inspiring initial work here is [12] that uses reinforcement learning in controller design. Another interesting approach is presented in [30] where Kalman filters are used for learning performance models in a self-adaptation setting.

3.3 Collaborative Adaptations

The third style, *collaborative adaptations* enables local feedback loops to collaborate with one another. From a control-theoretic perspective, state-of-the-art controller design does not provide solutions to guarantee stability in systems with arbitrary data exchange between controllers [25]. However, a number of researchers have studied different types of coordination between local feedback loops in self-adaptive systems [29]. These approaches typically restrict the interactions based on locality, the type of data that is exchanged, and the amount of data exchange. These restrictions allow to better control the adaptation and system behavior. As an example, in [11], we applied the collaborative adaptations style in a mobile learning application that supports traditional indoor lectures with outdoor activities [11]. Groups of three or four students use the application to measure and calculate properties of geometrical figures, such as circles and triangles. To support the tasks, students use GPS-enabled mobile devices that can interact with each other and with a server operated by the teacher. A group consists of one master device and multiple slave devices. The master device manages the group and interacts with the server. To guarantee the required level of GPS accuracy a self-adaptation layer is added to the system modeled with two types of MAPE-K loops deployed at each device. The first loop acts locally and is responsible for activating and deactivating the GPS service. The second loop deals with group management, where the master device collects data of the slaves and adapts the group when the GPS service of one of the devices fails.

The collaborative adaptations style provides a powerful approach to engineer self-adaption in SoS. The problem of designing self-adaptation requires the design of local feedback loops, their interactions and collaborations, and the definition of appropriate interfaces to support the collaborations. This style matches best with Maier's guiding design principles for SoS. If well-defined interaction protocols are available, the designer gets powerful means to support collaboration between systems and balance control. The tradeoff is a further increase of dependencies between systems of the SoS. Although, the collaborative adaptations style enables feedback loops to accommodate with one another reducing uncertainties, control of adaptation is decentralized. As such, a main issue remains providing guarantees for required properties that span multiple systems. Similar to the second style, promising approaches to deal with runtime uncertainties in the collaborative adaptations style are runtime testing and verification, and online learning. Multi-agent systems is another interesting field that offers interesting techniques to support the design of SoS with the collaborative style, such as autonomy and interaction protocols. Interesting recent work that deals with uncertainty issues is proposed in [15].

4. CONCLUSIONS

We have presented three basic architectural styles for self-adaptation in SoS and discussed challenges for each of them. The three styles provide increasing levels of knowledge sharing and collaboration, allowing to mitigate uncertainty at different scales. In the local adaptations style, feedback loops of different systems do not interact with each other. This truly decentralized style provides maximal autonomy to designers of individual systems. However, the cost is a tradeoff with respect to providing guarantees about uncertainties of cross-system properties. In the collaborations style, feedback loops interact directly with each other and collaboratively realize adaptations. This style creates dependencies between individual systems, but offers designers better support to guarantee cross-system properties. The regional monitoring–local adaptations style provides a middle ground between the two other styles. In this style feedback loops only share information. We belief that a selection between the styles will be determined by the particular domain properties of the SoS at hand.

The challenges discussed in this paper only focus on a set of architectural styles for self-adaptation of SoS, and in particular on coarse grained aspects of the design in relation to uncertainties. Beyond that, supporting self-adaptation in SoS poses numerous other challenges; to name a few: what are appropriate standards for the interactions between the adaptation components in SoS, how to deal with multiple concurrent adaptation concerns that spans multiple systems of a SoS, and how to involve the user in the adaptation process of SoS? In summary, enhancing SoS with support for self-adaptation offers an exiting area for future research.

5. REFERENCES

[1] T. Abdelzaher, Y. Diao, J. Hellerstein, C. Lu, and X. Zhu. Introduction to control theory and its application to computing systems. In *Performance Modeling and Engineering*. Springer, 2008.

[2] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *Control Systems, IEEE*, 23(3), 2003.

[3] L. Bakule. Decentralized control: An overview. *Annual Reviews in Control*, 32(1):87 – 98, 2008.

[4] J. Boardman and B. Sauser. System of systems-the meaning of of. In *International Conference on System of Systems Engineering*. IEEE, 2006.

[5] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic qos management and optimization in service-based systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, 2011.

[6] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, volume 5525. Springer, 2009.

[7] G. Denaro, M. Pezze, and D. Tosi. Ensuring interoperable service-oriented systems through engineered self-healing. In *ESEC/SIGSOFT FSE*, 2009.

[8] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury. Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server. In *Network Operations and Management Symposium*, 2002.

[9] J. Dowling and V. Cahill. *The k-component architecture meta-model for self-adaptive software. 3rd Metalevel Architectures and Separation of Crosscutting Concerns.* Springer, 2001.

[10] D. Garlan, S. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37:46–54, 2004.

[11] D. Gil de la Iglesia and D. Weyns. Guaranteeing robustness in a mobile learning application using formally verified mape loops. In *8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2013.

[12] R. Hafner and M. Riedmiller. Reinforcement learning in feedback control. *Machine Learning*, 84(1-2):137–169, 2011.

[13] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. Wiley, 2004.

[14] J. H. Holland. *Emergence: From Chaos to Order*. Redwood City, California: Addison-Wesley, 1998.

[15] W. Jiao and Y. Sun. Supporting adaptation of decentralized software based on application scenarios. *Journal of Systems and Software*, 86(7), 2013.

[16] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[17] J. Kramer and J. Magee. Self-managed systems: An architectural challenge. *Future of Software Engineering*, 2007.

[18] K. Kucher and D. Weyns. A self-adaptive software system to support elderly care. In *Modern Information Technology, MIT*, 2013.

[19] R. Lemos, H. Giese, H. Mueller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. Goaschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezze, C. Prehofer, W. Schaefer, R. Schlichting, D. Smith, J.-P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*. Springer, 2013.

[20] M. W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284, 1998.

[21] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau. Ultra-Large-Scale Systems - the software challenge of the future. Technical report, SEI, Carnegie Mellon, 2006.

[22] P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.

[23] A. P. Sage and C. D. Cuppan. On the systems engineering and management of systems of systems and federations of systems. *Inf. Knowl. Syst. Manag.*, 2(4):325–345, Dec. 2001.

[24] M. Usman Iftikhar and D. Weyns. A Case Study on Formal Verification of Self-Adaptive Behaviors in a Decentralized System. *ArXiv e-prints*, Aug. 2012.

[25] J. H. van Schuppen. Decentralized control with communication between controllers. In *Unsolved problems in mathematical systems and control theory.* Princeton University Press, Princeton, 2004.

[26] R. Wang and N. Kandasamy. On the design of decentralized control architectures for workload consolidation in large-scale server clusters. In *Int'l. Conference on Autonomic Computing*, 2012.

[27] X. Wang, D. Jia, C. Lu, and X. Koutsoukos. Deucon: Decentralized end-to-end utilization control for distributed real-time systems. *Parallel and Distributed Systems, IEEE Transactions on*, 18(7):996–1009, 2007.

[28] D. Weyns, S. Malek, and J. Andersson. Forms: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1), 2012.

[29] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. Goeschka. On patterns for decentralized control in self-adaptive systems. *Lecture Notes in Computer Science vol. 7475, Springer*, 2012.

[30] T. Zheng, M. Woodside, and M. Litoiu. Performance model estimation and tracking using optimal filters. *IEEE TSE*, 34(3):391–406, 2008.