Software Engineering Techniques for the Development of Systems of Systems*

Radu Calinescu and Marta Kwiatkowska
Computing Laboratory, University of Oxford
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

Abstract

This paper investigates how existing software engineering techniques can be employed, adapted and integrated for the development of large-scale systems of systems. Starting from existing system-of-systems (SoS) studies, we identify computing paradigms and techniques that have the potential to help address the challenges associated with SoS development, and propose an SoS development framework that combines these techniques in a novel way.

1 Introduction

The functionality and flexibility underpinning today's applications in areas ranging from transportation and healthcare to aerospace and defence can no longer be provided by a monolithic information system, however complex this might be. Instead, the required capabilities can only be achieved through employing collections of collaborative, heterogeneous and autonomously-operating systems—or *systems of systems*.

The crucial importance of many systems of systems and the high rates of late delivery, overspending and failure associated with their development have prompted the initiation of research programmes for the investigation of this new class of systems [13, 17, 27, 29] and its extensions [10, 24, 30]. The results of this research provide valuable insights into the distinguishing features of systems of systems [3, 5, 20, 25], and the challenges posed by their unprecedented size, diversity, variability, complexity, unforeseen interactions and emergent behaviour [10, 15, 19, 26].

The components of a system-of-systems (SoS) possess a level of operational autonomy that allows them to pursue their own, local objectives, independently and in addition to contributing to the global SoS objective(s) [5, 19]. SoS components are often developed, procured and managed independently [18, 25], and may belong to multiple open and evolving systems of systems that they could join and leave dynamically [5, 6, 13, 26]. Additionally, an SoS subclass that typifies key information systems of the future—i.e., the so-called *large-scale complex (IT) systems* [24] or *ultra-large-scale systems* [10, 30]—is characterised by incomplete and continually changing requirements and components, and by *normal failures*.

The challenges associated with the development of systems of systems are tremendous. They include the need to ensure the interoperability of a vastly diverse range of components [6, 25], to convey global objectives to SoS components in meaningful ways [10, 13], to achieve these objectives predictably and dependably in a dynamically changing environment [24, 30], and to attain high levels of SoS longevity [10, 26]. The decision to adopt an SoS solution, and hence to accept these challenges, often involves a degree of necessity. Thus, the planned application may have an inherently SoS nature, or may require the use of commercial, off-the-shelf systems and proprietary legacy systems as part of the overall solution. The primary justification, however, is the need to leverage the *emergence* of systems of systems, i.e., their ability to provide capabilities and levels of flexibility significantly beyond those provided by the sum of their components [3, 5, 18, 19, 26].

*This work was partly supported by UK EPSRC Grant EP/F001096/1.

These major advances in the understanding of systems of systems laid the foundation for essential work to identify high-level principles and practices governing their development [6, 10, 18, 19, 25]. Our paper takes this work further by investigating for the first time ways in which existing software engineering techniques can contribute to the development of systems of systems. After describing the results of this investigation in Section 2, the paper proposes a novel approach to integrating these techniques within a framework that extends the authors' previous work on probabilistic model checking [22, 23], model-driven development [8] and self-* computing [7, 9] to the realm of systems of systems (Section 3). We conclude with a brief summary in Section 4.

2 Software Engineering Techniques for SoS Development

This section examines existing software engineering paradigms and techniques that could help tackle some of the challenges associated with the development of systems of systems, and which are therefore likely to be part of the SoS development frameworks of the future. A summary of this analysis is presented in Table 1.

Service-oriented architectures (SOA) SoS development involves the integration and secure interoperation of vastly diverse technical systems [3, 5, 6, 10, 13, 18, 26]. Thanks to their platform independence, loose coupling and support for security, SOA solutions [33] represent strong candidates for implementing new computer systems or front-ends to legacy systems that need to be integrated into an SoS.

Policy-based autonomic computing Ecosystems, cities and economies are often pointed out as examples of effective systems of systems. A common characteristic of all these systems of systems is the way in which their global objectives are specified through high-level incentives, rewards and penalties rather than by setting concrete, precise targets [10, 24, 25]. Thus, the behaviour of ecosystems is governed by laws of nature. The development and everyday life of cities are subject to common or civil laws and regulations. The evolution of economies is guided by taxation policies. If these successful real-world examples are to be followed, techniques will be required that can convey the global objectives of systems of systems as *high-level policies* to their autonomous components. (Policy-based) autonomic computing addresses the development of systems that can manage themselves based on a set of high-level policies [21], and therefore represents an ideal paradigm for developing the computer-system components of an SoS.

Formal verification A major concern of systems of systems is their ability to achieve an overall objective in predictable and dependable ways, through the collaboration of component systems with different (and potentially conflicting) local goals [10, 24, 28]. Formal verification, and in particular model checking [11] and probabilistic model checking [23], comprise a range of techniques that could be used or adapted for use in the verification of SoS policies, and ultimately for SoS dependability management and assurance.

Model-driven development and code generation The open, evolving nature of systems of systems allows their components to join and leave dynamically [24, 28]. Having SoS components collaborate with peer systems whose characteristics are often unknown until runtime is a major challenge. A combination of model-driven development and runtime code generation in which a dynamically acquired model of a peer system is used to generate the necessary interfaces and logic for collaborating with this peer system [7] represents a promising approach to addressing this challenge.

Component-based development SoS engineering requires the integration of existing and future commercial, open-source and proprietary systems, and component-based development provides techniques that can help achieve this goal [1, 2, 12].

Table 1: Software engineering techniques that can help address SoS challenges

Techniques and paradigms	interoperability, security	dependability (assurance)	collaboration	global-objective specification	predictability	adaptibility	longevity	flexibility
	SoS challenges							
Service-oriented architectures	✓							
Policy-based autonomic computing				✓				✓
Formal verification		✓			✓			
Model-driven development/code generation						✓		
Component-based development			✓					
Dynamic reconfiguration						✓	✓	✓
Online machine learning						✓	✓	
Resource discovery			✓					✓

Dynamic reconfiguration Systems of systems are required to adapt continually to changes in their environment, structure and objectives [6, 18]. Recent advances in the study of dynamically reconfigurable software and hardware [14, 16] provide promising approaches for the development of the computer systems to be incorporated into the systems of systems of the future.

Online machine learning The levels of self-management that SoS components must achieve in impossible to anticipate circumstances are significantly beyond what can be pre-programmed into a computer system [15, 24, 30]. The online use of techniques specific to machine learning [4] is therefore likely to play a major role in the development of computer-based SoS components.

Resource discovery In the era of mobile computing, SoS components are expected to actively seek partner systems and establish collaborations with them, thus joining (and leaving) loosely-coupled federations of systems on a regular basis [5, 10, 24]. The rich spectrum of resource discovery techniques employed by today's distributed (e.g., grid- and web-based) computer systems [31] can be used as a basis for the development of techniques to support these capabilities.

3 A Framework for System-of-Systems Development

Our approach to integrating the software engineering techniques analysed in the previous section involves the use of a reconfigurable policy engine with the structure in Figure 1.[1] The **SOA implementation** of this policy engine as a web service [7] takes a model of a system and a set of policies, and ensures that the system achieves the objectives specified by these policies by adapting continually to changes in its environment. The system model has the form

$$model = (S, C, f), \tag{1}$$

where S and C represent the state and configuration parameters of the modelled system, and the partial function

$$f : S \times C \nrightarrow S \tag{2}$$

[1] The use of these techniques is emphasised in **bold text** in the policy engine description.

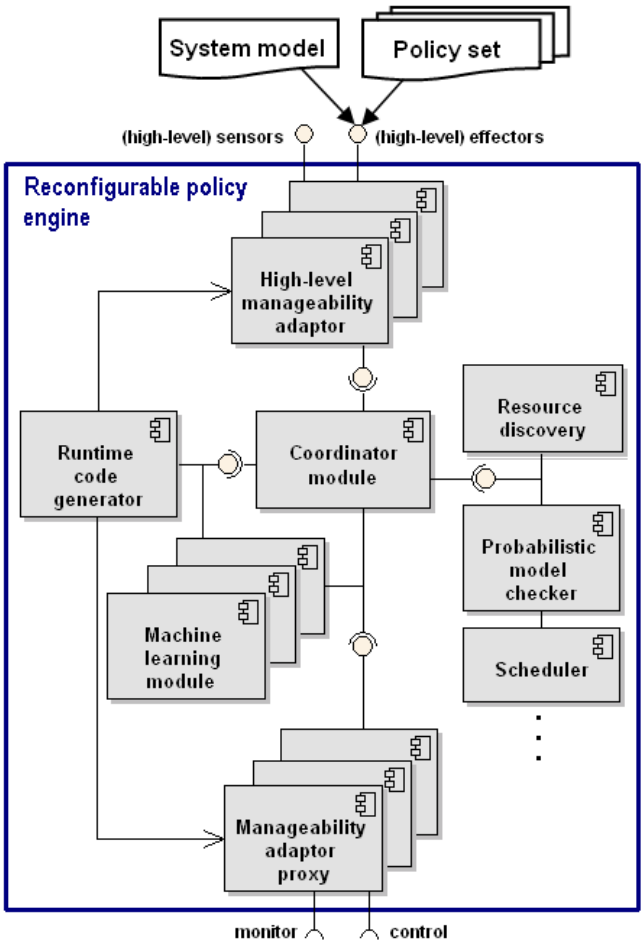

Figure 1: Reconfigurable policy engine

is a (possibly incomplete) specification of the system behaviour. Thus, for any current state $\mathbf{s} \in S$ and for any configuration $\mathbf{c} \in C$ such that $(\mathbf{s}, \mathbf{c}) \in \operatorname{dom} f$, $f(\mathbf{s}, \mathbf{c})$ represents the future state of the system. When a running instance of the policy engine is **dynamically reconfigured** by means of a model (1), its *runtime* ***code generator*** employs **model-driven development** techniques to produce *manageability adaptor proxies* [7]. The *monitor* and *control* interfaces of these proxies allow the engine to read the state and to modify the configuration of the system components, respectively.

The policy engine version described in our previous work [7, 9] supports the three types of policies that are standard in **policy-based autonomic computing** [32]:

$$p_{action} : S \times C \nrightarrow C \tag{3}$$

$$p_{goal} : S \times C \to \{\texttt{false}, \texttt{true}\} \tag{4}$$

$$p_{utility} : S \times C \to \mathbb{R} \tag{5}$$

An *action policy* p_{action} specifies how the system configuration should be changed when the system reaches certain state/configuration combinations—note that such policies can be implemented even when $\operatorname{dom} f = \emptyset$. A *goal policy* p_{goal} specifies the desirable state/configuration combinations for the system, i.e., the system should be maintained in an operation area for which the policy value is always `true`. Finally, a *utility policy* $p_{utility}$ associates a value with each state/configuration combination, and the system configuration should be adjusted to ensure that this value is maximised at all times. The policy engine uses **resource discovery** techniques to identify the system components to which the policies (3)-(5) are applied, and **formal verification** techniques in the form of probabilistic model checking for the realisation of these policies

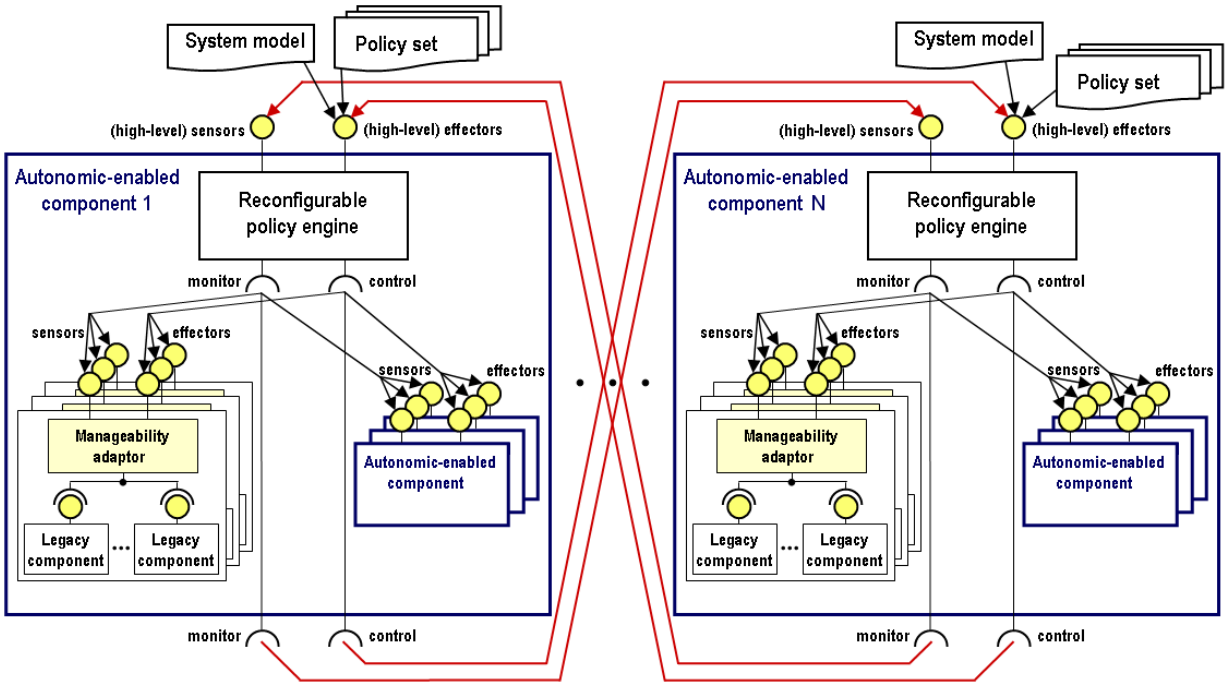

Figure 2: System-of-systems architecture

[9]. In a future version of the policy engine, **online machine learning** will be employed to continually improve the accuracy of (and ultimately to generate) the partial function f from (1) based on the observed behaviour of the system.

In this paper, we describe for the first time a new policy type that supports **component-based development** and enables the policy-driven interoperation and collaboration of heterogeneous collections of legacy and new information systems. The new policy type is termed a *component-definition policy* and is formally defined as

$$p_{definition} : S \times C \rightarrow M \times S' \times C', \tag{6}$$

where M represents the set of all models (1), and

$$\forall\, \mathbf{s} \in S;\ \mathbf{c} \in C \bullet p_{definition}(\mathbf{s}, \mathbf{c}) = \left(model', \mathbf{s}', \mathbf{c}'\right) \Rightarrow model' = (S', C', f'). \tag{7}$$

The policy in (6) specifies how the *high-level sensor* and *high-level effector* interfaces of the policy engine should expose the system (components) under its control as a single entity with model $model'$, state $\mathbf{s}' \in S'$ and configuration $\mathbf{c}' \in C'$.

Figure 2 depicts the generic architecture of an SoS built around an extension of the policy engine from [7] that supports component-definition policies. Each of the top-level *autonomic-enabled components* 1 to N in this architecture is a system managed by an appropriately configured instance of the policy engine. At the SoS level, the policy engine instances expose the state and configuration of their systems, employ resource discovery to identify peer SoS components, and collaborate with these. At the local level, the policy engines organise heterogeneous collections of components into a single system. These collections can comprise legacy components whose interfaces are accessed through *manageability adaptors* [7] and autonomic-enabled components (i.e., new systems that expose *sensor* and *effector* interfaces permitting their direct management by the policy engine, or other instances of the top-level autonomic-enabled components in Figure 2).

The policy engine in [7] was used to architect monolithic self-managing systems in areas including server capacity allocation, dynamic power management and management of datacentre cluster availability [7, 9]. To validate our extended version of the policy engine, we used it to develop a simulated enterprise-wide SoS comprising several collaborating datacentres and a

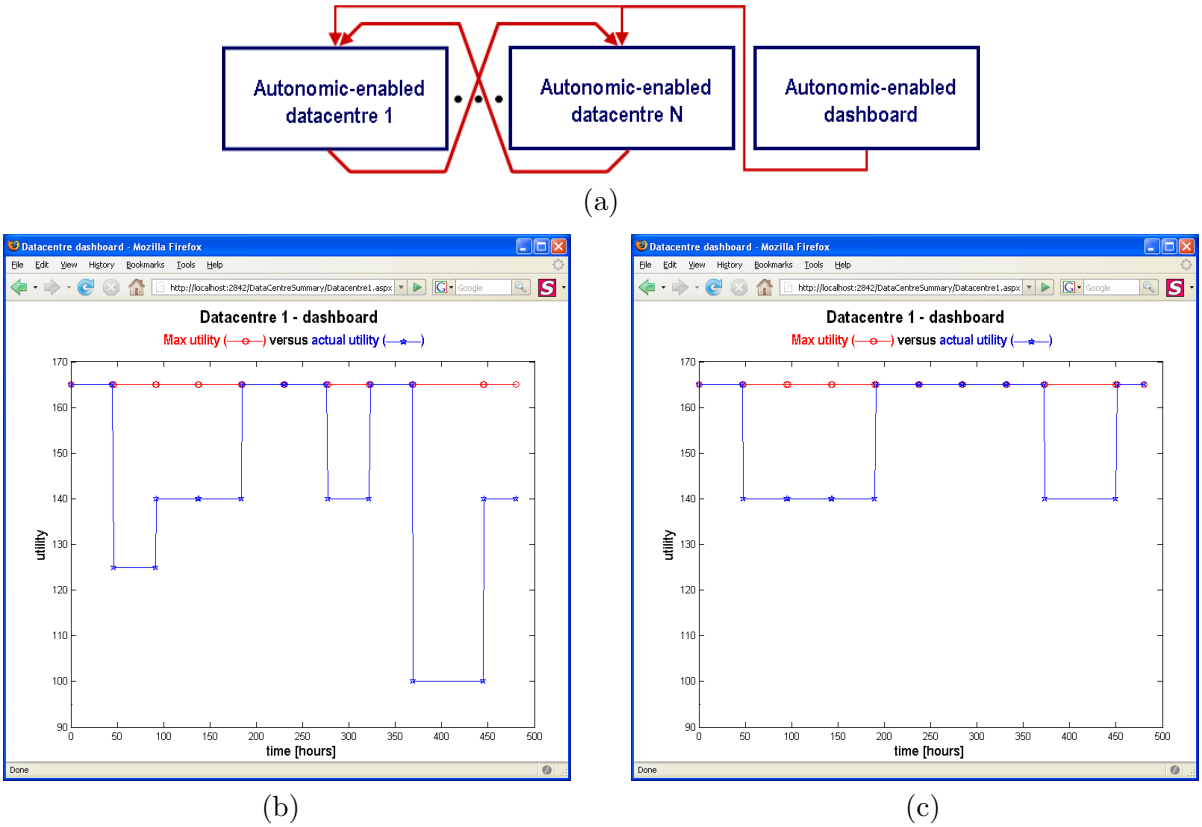

Figure 3: High-level architecture of a datacentre SoS (a), dashboard for isolated datacentre (b), and dashboard for identical datacentre operating as part of a two-datacentre SoS (c)

"dashboard" system for monitoring their performance. Each datacentre consists of several clusters with variable workloads, and uses an instance of the policy engine in Figure 1 to optimise the availability-driven allocation of its servers to these clusters. The architecture of an individual *autonomic-enabled datacentre* and the utility policy (5) it employs are described in detail in [9].

Figure 3(a) shows how $\mathsf{N} \geq 1$ datacentres can be combined into a system of systems. A component-definition policy (6) supplied to each datacentre configures its policy engine to expose: (a) any spare servers, so that they can be leased to other datacentres; and (b) the utility achieved by the datacentre, so that datacentre dashboards can be generated dynamically. With the notation in (6)–(7), the exposed datacentre state and configuration are given by

$$S' = \{id : \texttt{string}, spareServers : \texttt{int}, utility : \texttt{int}, maxUtility : \texttt{int}\}$$
$$C' = \{peerId : \texttt{string}, requestedServers : \texttt{int}, svc : \texttt{string}\}$$
$$\operatorname{dom} f' = \emptyset$$

By appropriately setting the configuration parameters in C', a datacentre *peerId* can ask datacentre *id* to run service *svc* on $requestedServers \leq spareServers$ of its servers, thus gaining the ability to cope with workloads that exceed its local capacity. Figures 3(b) and 3(c) depict the automatically-generated dashboards for the simulation of an isolated datacentre (i.e., $\mathsf{N} = 1$) over a 20-day period in simulated time, and for the simulation of the same datacentre when it operates as part of a two-datacentre SoS (i.e., $\mathsf{N} = 2$), respectively. These dashboards show that a better utility (i.e., a larger number of clusters operating at optimal availability [9]) is achieved when the considered datacentre can take advantage of the spare servers of a peer datacentre.

4 Conclusion

A common finding of SoS studies is that existing techniques and tools are unable to address the whole spectrum of challenges associated with the development of systems of systems. Notwithstanding the disparity between what can be achieved using current approaches and these challenges, the SoS development frameworks of the future are bound to incorporate some of today's software engineering techniques or adapted variants of them. This paper examined techniques that are candidates for this role. Having first identified the SoS challenge(s) that each such technique can help address, we then proposed a new approach to combining these techniques into a policy engine for the development of systems of systems. Areas of future work include the validation of the proposed framework within new application domains, the development of SoS-specific online machine learning techniques, the synthesis of high-level SoS policies from specifications, and the design of metrics for the assessment of global SoS effectiveness.

References

[1] Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, July 1997.

[2] Farhad Arbab. Abstract behavior types: a foundation model for components and their composition. *Science of Computer Programming*, 55(1–3):3–52, March 2005.

[3] Y. Bar-Yam et al. The characteristics and emerging behaviors of system-of-systems. Complex physical, biological and social systems project report, New England Complex Systems Institute, January 2004. http://necsi.org/education/oneweek/winter05/NECSISoS.pdf.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[5] John Boardman and Brian Sauser. System of systems – the meaning of *of*. In *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, pages 118–123, 2006.

[6] Lisa Brownsword, David Fisher, Ed Morris, James Smith, and Patrick Kirwan. System-of-systems navigator: An approach for managing system-of-systems interoperability. Technical Report CMU/SEI-2006-TN-019, Carnegie Mellon Software Engineering Institute, April 2006. http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tn019.pdf.

[7] Radu Calinescu. Implementation of a generic autonomic framework. In D. Greenwood et al., editor, *Proceedings 4th International Conference on Autonomic and Autonomous Systems (ICAS'08)*, pages 124–129. IEEE Computer Society Press, March 2008.

[8] Radu Calinescu. Methodology for the model-driven development of self-managing systems. In *Proceedings of the ACM International Conference on Computing Frontiers*, May 2008. To appear.

[9] Radu Calinescu and Marta Kwiatkowska. Cost-effective development of self-managing systems using probabilistic model checking, 2008. Submitted to the 23rd IEEE/ACM International Conference on Automated Software Engineering.

[10] Carnegie Mellon Software Engineering Institute. *Ultra-Large-Scale Systems. The Software Challenge of the Future*. Carnegie Mellon University, 2006. http://www.sei.cmu.edu/uls/files/ULS_Book2006.pdf.

[11] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2000.

[12] Ivica Crnkovic and Magnus Larsson, editors. *Building Reliable Component-Based Software Systems*. Artech House Publishers, 2002.

[13] William A. Crossley. System of Sytems: An Introduction of Purdue University Schools of Engineering's Signature Area. In *Proceedings of the Engineering Systems Symposium*, 2004. http://esd.mit.edu/symposium/pdfs/papers/crossley.pdf.

[14] Tarek El-Ghazawi, Esam El-Araby, Miaoqing Huang, Kris Gaj, Volodymyr Kindratenko, and Duncan Buell. The promise of high-performance reconfigurable computing. *Computer*, 41(2):69–76, February 2008.

[15] Greg Goth. Ultralarge systems: Redefining software engineering? *IEEE Software*, 25(3):91–94, May/June 2008.

[16] Markus Hannebauer. *Autonomous Dynamic Reconfiguration in Multi-agent Systems*. Springer, 2002.

[17] Integration of Software-Intensive Systems (ISIS) Initiative: Addressing System-of-Systems Interoperability. http://www.sei.cmu.edu/isis.

[18] Jeremy M. Kaplan. A new conceptual framework for net-centric, enterprise-wide, system-of-systems engineering. Defense & Technology Papers 30, US Center for Technology and National Security Policy, July 2006. http://www.ndu.edu/ctnsp/Def_Tech/DTP%2030%20A%20New%20Conceptual%20Framework.pdf.

[19] Charles Keating. Research foundations for system of systems engineering. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2720–2725, 2005.

[20] Charles Keating, Ralph Rogers, Resit Unal, David Dryer, Andres Sousa-Poza, Robert Safford, William Peterson, and Ghaith Rabadi. System of systems engineering. *Engineering Management Journal*, 15(3):36–45, September 2003.

[21] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer Journal*, 36(1):41–50, January 2003.

[22] Marta Kwiatkowska. Quantitative verification: Models, techniques and tools. In *Proc. 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE*, pages 449–458. ACM Press, September 2007.

[23] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

[24] LSCITS Consortium. Large-Scale Complex Information Technology Systems Initiative. http://www.lscits.org.

[25] Mark W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284, February 1999.

[26] Abe Meilich. System of systems engineering (SoSE) and architecture challenges in a net centric environment. In *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, pages 1–5, 2006.

[27] US National Centers for Systems of Systems Engineering (NCSOSE). http://www.eng.odu.edu/ncsose/.

[28] Steven W. Popper, Steven C. Bankes, Robert Callaway, and Daniel DeLaurentis. System of systems symposium: Report on a summer conversation. In *Proceedings of the 1st System of Systems Symposium*, 2004. http://www.potomacinstitute.org/academiccen/SoS%20Summer%20Conversation%20report.pdf.

[29] US System of Systems Engineering Center of Excellence (SoSECE). http://www.sosece.org.

[30] US Industry/University Collaborative Research Center for Ultra-Large-Scale Software-Intensive Systems (ULSSIS). http://ulssis.cs.virginia.edu.

[31] Koen Vanthournout, Geert Deconinck, and Ronnie Belmans. A taxonomy for resource discovery. *Personal and Ubiquitous Computing*, 9(2):81–89, March 2005.

[32] Steve R. White et al. An architectural approach to autonomic computing. In *Proc. 1st IEEE International Conference on Autonomic Computing*, pages 2–9. IEEE Computer Society, 2004.

[33] O. Zimmermann et al. *Perspectives on Web Services: Applying SOAP, WSDL and UDDI to Real-World Projects*. Springer, 2005.