Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions

CLAUS BALLEGAARD NIELSEN, Aarhus University
PETER GORM LARSEN, Aarhus University
JOHN FITZGERALD, Newcastle University
JIM WOODCOCK, University of York
JAN PELESKA, University of Bremen

The term "System of Systems" (SoS) has been used since the 1950s to describe systems that are composed of independent constituent systems, which act jointly towards a common goal through the synergism between them. Examples of SoS arise in areas such as power grid technology, transport, production, and military enterprises. SoS engineering is challenged by the independence, heterogeneity, evolution, and emergence properties found in SoS. This article focuses on the role of model-based techniques within the SoS engineering field. A review of existing attempts to define and classify SoS is used to identify several dimensions that characterise SoS applications. The SoS field is exemplified by a series of representative systems selected from the literature on SoS applications. Within the area of model-based techniques the survey specifically reviews the state of the art for SoS modelling, architectural description, simulation, verification, and testing. Finally, the identified dimensions of SoS characteristics are used to identify research challenges and future research areas of model-based SoS engineering.

Categories and Subject Descriptors: H.1.1 [**Systems and Information Theory**]: General Systems Theory; I.6.0 [**Simulation and Modelling**]: General; F.4.0 [**Mathematical Logic and Formal Languages**]: General

General Terms: Design, Languages

Additional Key Words and Phrases: System of systems, systems engineering, model-based engineering

ACM Reference Format:
Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, and Jan Peleska. 2015. Systems of systems engineering: Basic concepts, model-based techniques, and research directions. ACM Comput. Surv. 48, 2, Article 18 (September 2015), 41 pages.
DOI: http://dx.doi.org/10.1145/2794381

Our work is partially supported by the COMPASS project funded by the European Commission's 7th Framework Programme (Grant Agreement 287829), and the INTO-CPS project funded by the Horizon 2020 Programme (Grant Agreement 644047).
Authors' addresses: C. B. Nielsen, Centre for Systems Engineering, Cranfield University, Defence Academy of the United Kingdom, Shrivenham, SN6 8LA, United Kingdom; email: c.nielsen@cranfield.ac.uk; P. G. Larsen, Department of Engineering, Aarhus University, 8200 Aarhus N, Denmark; email: pgl@eng.au.dk; J. Fitzgerald, School of Computing Science, Newcastle University, NE1 7RU, United Kingdom; email: john.fitzgerald@ncl.ac.uk; J. Woodcock, Department of Computer Science, University of York, YO10 5GH, United Kingdom; email: jim.woodcock@york.ac.uk; J. Peleska, Informatik, Universität Bremen, D-28334 Bremen, Germany; email: p@informatik.uni-bremen.de.

DOI: http://dx.doi.org/10.1145/2794381

1. INTRODUCTION

In many important domains, including infrastructure, healthcare, transportation, emergency response, and defence, reliance is placed on the delivery of a service by a system composed of largely independent, typically preexisting, systems. For example, the successful treatment of a patient in an emergency results from the interaction of several separately owned and managed systems including telephony, ambulance assignment, information sharing, communications, and hospital management. These constituent systems may have existed before requirements (e.g., to meet maximum response times or to guarantee the confidentiality of patient data) were imposed on their collective behaviour. Advances in network and communications technology have made it possible to conceive of deliberately engineering and maintaining such "Systems of Systems" (SoSs).

The SoS engineer faces several important challenges, not least the need to identify the boundaries of the overall SoS and of the independent constituent systems within it. These boundaries relate to both technical aspects such as interfaces, integration and testing, and management aspects such as governance and stakeholder involvement. Further challenges relate to the gaining of confidence in system operation, in terms of behavioural correctness, performance qualities, and their validation. Many of these challenges are already the foci of work in the field of systems engineering [Hitchins 2005]. SoS engineering is not a completely new or opposing discipline, but is a subfield of systems engineering that focuses on the boundaries and interactions between independent, distributed, and evolving constituent systems and their stakeholders.

The distributed constituent systems in an SoS are owned and operated by independent stakeholders, and consequently there are limitations on the exchange of information about them. On the other hand, SoS behaviour is dependent on emergent phenomena observed at the system boundaries. Consequently, there is a need for engineering techniques that address such characteristics. The challenges that SoS engineering encounters to a higher degree than traditional systems engineering are described by Dahmann et al. and can be summarized as follows: stakeholders with competing interests and priorities, no centralised authority over all the systems, added complexity due to multiple system lifecycles, as well as balancing testing, behaviour characteristics, and performance needs between the constituent systems and the SoS [Dahmann et al. 2008].

SoSs arise in increasingly broad domains. A widely used example is in emergency response, in which agencies (such as fire, police, hospital) with independently owned and managed systems nevertheless collaborate to deliver a service on which reliance is placed. Other instances of SoSs arise in infrastructure systems, which typically deliver services through the collaborative operation of multiple providers. In road transport, for example, local, regional, and national agencies, often operating to different priorities, manage flow in interlinked traffic networks, but must offer services that remain safe across their boundaries. Recent applications of SoS engineering have been in less conventional domains, such as transport of radioactive source materials [Mauss et al. 2015], and the design of audio/video networks that stream content from multiple providers to heterogeneous sets of devices [Bryans et al. 2014]. After surveying the state of the art in model-based techniques for SoS engineering, we discuss some significant examples of SoSs in Section 4.

Although SoSs arise in very diverse domains and may differ in architecture and application, the engineering problems that they present have some commonalities. These include the need to gain assurance of key properties of the SoS as a whole in spite of the operational and managerial independence of the constituent systems, their distributed, concurrent character, and their heterogeneity. In addition, the range of stakeholders involved, including the owners and operators of constituent systems,

their integrators, and ultimately those who experience the system behaviour of the SoS, implies the need to employ methods and tools that support collaborative working from the elicitation of requirements to testing and maintenance.

We regard a system as "a combination of interacting elements organized to achieve one or more stated purposes" [INCOSE 2015; International Organization for Standardization 2015]. An SoS is a system, some of whose elements are themselves designated as systems. A discipline of SoS engineering has begun to emerge, aiming to extend systems engineering with the ability to develop, maintain, and adapt SoSs effectively. In common with other areas of systems engineering, there has also been considerable interest in the use of model-based techniques [Cantot and Luzeaux 2011]. We will use the term "model" to refer to an abstract description of a system of interest. The particular abstraction decisions made in a given model are determined by the model's purpose [Kramer 2007]. For example, a model of an emergency response SoS constructed in order to analyse maximum response times is unlikely to include an explicit representation of all the fields in a patient record.

Models may be used to describe real-world objects or phenomena to a certain level of fidelity. Equally, models may be used during design to describe potential systems that are yet to be realised. In SoS engineering, as in systems engineering more generally, both kinds of model arise in the development and maintenance of a system or SoS. In particular, descriptive models of the already existing elements of an SoS may be combined with design models of elements that are to be constructed. Models can cover such diverse aspects of an SoS as its structure, functionality, communications, and behaviour.

To gain confidence that an SoS architecture will respect key properties, it is paramount to have a precise model of the constituents and the connectors between them, the properties of the constituents, and the SoSs environment. Such a model supports the "trade-off" of alternative designs at early development stages and the precise determination of the contract that exists between each constituent system and the SoS. Model-based approaches are already well accepted in industry for their ability to manage and control the overall complexity of a system, reveal and document its key structure and behaviour, and communicate these to stakeholders [Woodcock et al. 2009]. As the use of models evolves and matures, there is a clear need for verification and validation technology that allows the value of models to be exploited.

Model-based SoS engineering is an active area, both of practice and research. There is considerable evidence of activity in forums such as the IEEE Systems Engineering Conference,[1] the IEEE Conference on System of Systems Engineering,[2] and INCOSE,[3] which runs an industry-led working group on SoS engineering, as well as publications of record in media such as the IEEE Systems Engineering Journal and the Journal of System of Systems Engineering. Notable in the area of model-based methods is the joint Model-Based Systems Engineering initiative of INCOSE and the Object Management Group.[4] In 2010, the European Commission funded a group of projects specifically addressing SoS engineering,[5] and has held a series of expert workshops aiming to help determine the potential of research in the field up to 2020 [European Commission 2012]. There is a considerable and growing volume of work in the area, but the subject is young, and general lessons and patterns that cut across applications remain to be

[1]http://www.ieeesyscon.org/.

[2]For example, http://www.sosengineering.org/2015/.

[3]http://www.incose.org.

[4]http://www.omgwiki.org/MBSE.

[5]These include two projects focussing on research in model-oriented methods for the design of SoSs: COMPASS (www.compass-research.eu) and DANSE (www.danse-ip.eu), T-AREA-SOS which promotes transatlantic cooperation (www.tareasos.eu/) and ROAD2SOS (www.road2sos-project.eu).

learned. Therefore, the exact extent to which model-based engineering can be applied to SoS engineering is still an open research question.

The purpose of this article is to provide a structured view of the state of the art in model-based techniques in SoS engineering, and to identify challenges for research in this field. We focus on the use and potential of models that have formal semantic foundations. Our 2009 review of industrial applications of formal methods suggested that the benefits of formal modelling were being increasingly realised in industrial practice, particularly in hardware, and to a greater extent in software design [Woodcock et al. 2009]. The field of SoS engineering is nascent, exciting, and at a very early stage of development with as yet no formal foundation able to scale from hardware and software to the level of the SoS itself. The scope of the article is confined to the modelling of technical systems, since this covers the major part of work to date. However, progress on sociotechnical aspects of the interaction between humans and SoSs, and between humans mediated through the SoS, is essential to success in providing a truly comprehensive approach to SoS engineering [Lock and Sommerville 2010]. Business aspects, including requirements elicitation, tendering, and procurement, although also outside the scope of the article, are essential to a comprehensive understanding of SoS [Holt 2012].

This article concentrates on two aspects of model-based SoS engineering: firstly the characteristics of SoSs that make them particularly challenging, and secondly the engineering activities that form the core of model-based approaches, specifically modelling, architectural description, simulation, testing, and verification. A review of the many attempts to define and classify SoSs (Section 2) suggests that it may be beneficial to view SoSs in terms of the dimensions that categorise them (Section 3), and that pose particular modelling challenges. A review of several exemplars of SoS engineering from the literature helps to ground the subsequent discussion (Section 4). In Section 5, current and promising technologies for formal model-based SoS engineering are explored. In Section 6, research challenges in realising the potential of each of these technologies are identified and the steps necessary to provide a firmer foundation for model-based SoS engineering is examined.

2. DEFINITION AND CHARACTERISTICS

As might be expected in an emerging field, there is yet no precise and widely accepted definition of SoS to which the bulk of the literature conforms, making it difficult to bound the field precisely. The literature is diverse, and there are many attempts to define and characterise SoS. Several reviews have sought to achieve some convergence [Keating 2005; Jamshidi 2005; Sharawi et al. 2006; Lane and Valerdi 2007; Gorod et al. 2008; Jamshidi 2008].[6] In this section, we review attempts to define, characterise, or describe SoS, starting with an historical overview (Section 2.1), followed by a focus on literature that describe SoS via characteristics (Section 2.2) or Taxonomies (Section 2.3). Views on SoS in Industry (Section 2.4), Academia (Section 2.5), and Engineering Handbooks (Section 2.6) are presented, with the section ending with a focus on literature that extend existing SoS definitions (Section 2.7).

2.1. Historical Overview of SoS Definition

The early ideas of SoS come from a variety of sources. Boulding's paper on general systems theory uses the term "system of systems" to describe the organisation of

[6]Note that some surveys include Jackson and Keys [1984] and Müller-Merbach [1994] that use SoS to describe a system for the arrangement and ordering of various "systems methodologies" concepts. These are not included in this survey, as we concentrate on the relationships between and behaviour of operational constituent systems, and not on creating structures for analysis methodologies.

theoretical constructs [Boulding 1956]. The SoS concept is described as "the arrangement of theoretical systems and constructs in a hierarchy of complexity." Boulding's classification of SoS consists of a static structure for system's anatomy as well as a dynamic dimension that enables the system to adapt over time, as the SoS is an "open system" that can be affected by external events. There is a division of labour between differentiated but mutually dependent parts, and the SoS has a "transcendent" and "unknowable" element. Several features of Boulding's description bear a clear resemblance to aspects of the more recent engineering notions of SoS considered in the following. Later, both urban city planning, systems science structures, and biological systems came to be characterised as SoS [Berry 1964; Ackoff 1971; Jacob 1974].

The United States' Strategic Defense Initiative (SDI) from the late 1980s became a key factor in establishing SoS as an engineering concept focused on joining independent systems together [United States, Congress, Senate, Committee on Armed Services 1988]. Subsequently, SoS research intensified in both academia and industry, and attracted increased awareness. Nevertheless, it took another 15 years for SoS Engineering (SoSE) to develop as a recognised discipline, and by the early 21st century was still regarded as being in its infancy [Keating et al. 2003].

2.2. Defining SoS via Characteristics

In a response to a widening recognition of SoSs combined with the lack of a shared agreement on an SoS definition, Maier [1996][7] characterises SoS in terms of five principal features sometimes referred to by the acronym "OMGEE":

***O**perational Independence.* Any system that is part of an SoS is independent and is able to operate serviceably if the SoS is disassembled.
***M**anagerial Independence.* Despite collaborating with the other members of the SoS, the individual systems are self-governing and individually managed so that they "not only can operate independently, they do operate independently."
***G**eographic Distribution.* The parties collaborating in an SoS are distributed over a large geographic extent. Although the geographic extent is defined vaguely, it is stressed that the collaborating systems can only exchange information and not considerable quantities of mass or energy.
***E**volutionary Development.* An SoS's existence and development are evolutionary in the sense that objectives and functionality can be under constant change, as they can be added, modified, or removed with experience. Thus, an SoS never appears completely formed.
***E**mergent Behaviour.* Through the collaboration between the systems in an SoS a synergism is reached in which the system behaviour fulfils a purpose that cannot be achieved by, or attributed to, any of the individual systems.

Boardman and Sauser [2006] seek to identify characteristics that distinguish SoSs from conventional systems and pay particular attention to the merging of new constituents with existing systems to form the SoS. They identify five characteristics for SoS (acronym "ABCDE"):

***A**utonomy.* Each system is free and independent with its own purpose of operation.
***B**elonging.* Systems function collaboratively to meet a common higher purpose.
***C**onnectivity.* Synergism is enabled by the highly dynamic distributed network.

[7]In the SoS literature, Maier [1998a] is widely cited. However, the characteristics originate from Maier [1996], which also exists in a whitepaper version published online [Maier 1998b]. All these versions share the title "Architecting Principles for Systems-of-Systems."

***D**iversity.* The constituents are heterogeneous self-sufficient systems that are open for enhancement by evolution and adaptation.

***E**merging.* The cumulative actions and interactions between the constituents of an SoS give rise to the behaviours that can be attributed to the SoS as a whole.

Abbott argues that the SoS term should be reserved for systems that are qualitatively and structurally different from traditional systems [Abbott 2006]. An SoS should not be seen as a hierarchy of components, but as an environment where systems reside and systems can join, operate, and interact within it. Abbott defines three characteristics of such environments: (1) *Open at the top*, meaning that an SoS is continually open for addition of new applications and systems, without any top-level system defining the SoS. (2) *Open at the bottom*, meaning that the lowest level of the SoS, such as a specific communication stack, may be changed at any time. (3) *Continually evolving, but slowly*: an SoS is never complete as it evolves with changes in the surrounding environment. At least three forms of system evolution exist: standards and interfaces adjustment, technological changes, and feature modification.

2.3. Defining SoS via Taxonomies

Shenhar [1994] proposes a two-dimensional taxonomy for systems; a *technological uncertainty* dimension describing the maturity of the technologies and the *scope level* dimension classifying systems from a single-purpose *assembly* to an *array* of geographically dispersed systems interacting to achieve a common purpose. Shenhar and Bonen [1997] later identify SoS with the array type.

DeLaurentis and Crossley [2005] propose a taxonomy for SoS analysis that described how an SoS materialises when needs are met by a combination of independent systems that rely on the interrelationships between one another. The taxonomy emphasises three dimensions: (*1*) "Connectivity": analysis of interdependencies and the dynamic topology changes over time, (*2*) "Control/autonomy": balance between authority control versus autonomous behaviour, and (*3*) "System type": the balance between hardware/software and human enterprise.

Finally, Karcanias and Hessami [2010] considers SoS as an evolution of composite systems, focused on the integration challenges of autonomous and independent systems in large-scale projects. SoSs are described as complex multisystems that define a global goal and aggregate interdependent constituent systems.

2.4. Industrial Views on SoS

In industry there has been a focus on the SoS challenges that occur in the communication and exchange of data between systems.

Noam describes the change in the interconnection between carriers and "telecommunications integrators" in new telecommunication infrastructures as a move from "network of networks" to SoS, because networks start separating into dynamic systems, which will allow integrators to deliver services to the consumers, without them owning the telecommunication network [Noam 1994].

Focusing on data transmission and hierarchical structures, Kotov [1997] defines an SoS as "large-scale concurrent and distributed systems the components of which are complex systems themselves." These complex systems are described as a result of hardware, software, and network being merged into larger integrated system architectures with constant growing complexity.

The SoS term is used to describe the type of systems emerging from rapidly improving military capabilities on intelligence gathering and sharing [Owens 1995], which expands to a focus on the integration and interoperability between C4I (Command, Control, Communication, Computers, and Intelligence) and ISR (Intelligence,

Surveillance, Reconnaissance) systems [Manthorpe 1996] as being key in future battlefield scenarios [Pei 2000].

Within enterprise systems, SoSs are challenging because the continuous *emerging behaviour* makes it difficult to capture their business value and the heterogeneity of constituents means that each system has its own capabilities, users, and interfaces, making integration a challenge [Carlock and Fenton 2001]. In the same way, Cocks [2006] describes SoS as the result of a system engineering process involving systems for which integration and lifecycle development are not under centralised control.

Boehm [2006] refers to Software-Intensive SoSs (SISoS) as a key factor in the competitiveness of organisations that supply software services. Future demands will require dynamically evolving systems consisting of numerous independently developed systems, that will have emergent requirements and sociotechnical issues as key challenges.

2.5. Academic Views on SoS

Within academia there has been a focus on engineering discipline and engineering education for SoS. Eisner et al. [1991] identify the need of focusing on the challenges that arise from the scale and complexity of SoSs (there denoted "S2"), and builds this around seven characteristics including notions of interdependence and—unusually—a requirement for overall control of the constituents.

In the same way as Eisner et al., Roe assume an overarching authority that has responsibility for overall SoS requirements, but focus on the use of formal specifications to evaluate functional decomposition and interoperability between legacy systems. In one of the first books on SoS simulation and modelling Cantot and Luzeaux describes an SoS as an assembly of systems that are independently acquired and then operated in order to maximise the performance of the global operation of the grouped systems at certain periods [Cantot and Luzeaux 2009, 2011].

With a focus on education Lukasik [1998] describes an SoS as a self-organizing system assembled from multiple distributed individual systems, and assembly that has not been directly designed; instead, it follows from the evolution of the integration of constituents. Chen and Clothier [2003] focuses on systems engineering practice, but attempts to improve and adapt traditional methods by focusing on the design of the environment in which the SoS constituents reside.

Keating et al. [2003] provide a perspective in which an SoS is described as a metasystem of interrelated complex subsystems, constructed out of systems that integrate in order to reach a high goal, despite the individual parts being unlike in technology, geography, and operation. In relation to this, Crossley [2004] suggests that a change in industry and defence acquisition, moving from specification of *single systems* towards less implementation-specific specifications of *capabilities*, has allowed for more existing and future systems interacting to fulfil a common mission.

2.6. Views in Professional Handbooks

The U.S. Department of Defense Systems Engineering Guide for Systems of Systems defines an SoS as "a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities" OUSD(AT&L), DoD [2008] (originally from the Defense Acquisition Guidebook [Department of Defense 2004]). SoS development involves the creation of systems, which are collections of legacy, evolving, and new systems, that must have a high degree of flexibility and adaptability. Two of the main considerations are (1) the *lack of authority* over the constituent systems because of their independent management, funding, and objectives that may not align with those of the SoS as a whole; and

(2) the *emergent behaviour* adding a large degree of unpredictability, as overall system behaviour cannot be predicted by having individual knowledge of each constituent.

In its most recent edition, the INCOSE Systems Engineering Handbook [INCOSE 2015] (an update over the previous 2011 edition [INCOSE 2011]) treats an SoS as a system whose elements are managerially and/or operationally independent systems, and which together usually produce results that cannot be achieved by the individual systems alone. It cites Maier's characterisation of SoSs, and further cites Dahmann's "pain points" [Dahmann 2014] as significant challenges facing SoS engineering: absence of common authorities; leadership in the SoS organisational environment; variety of constituent systems' perspectives; management of diverse requirements and the overall SoS capability; autonomy, interdependencies, and emergence; and the challenges of validation, testing, and learning. Security is further identified as a pressing concern. The handbook observes that SoS engineering demands a balance between linear procedural methods for systematic activity and holistic nonlinear methods in the face of complexity arising from SoS emergence.

2.7. Extensions of Existing Definitions

Krygiel [1999] define SoS as "a set of different systems so connected or related as to produce results unachievable by the individual systems alone,' which is an adaption of Maier and Rechtin [1997]. Sage and Cuppan [2001] build on Maier's SoS characteristics to apply a strategy for organisational structuring, while Bar-Yam et al. [2004] use military, biological, and sociological case studies to supplement Maier's characteristics with a notion of interdependency arising from the interoperability of constituents. Fisher [2006] builds on Maier's characteristics to describe a class of large-scale software-intensive systems that exhibit levels of complexity for which traditional engineering methods are likely to prove fruitless. Baldwin and Sauser [2009] use Maier's characteristics to define an SoS as "a type of system composed of traditional systems and distinguished by the dynamic properties of autonomy, belonging, connectivity, diversity, and emergence."

Cook et al. [1999] attempts to establish a methodology for developing military SoS via a structure of different levels of complexity (essentially based on Boulding [1956]). SoSs are regarded as self-maintaining, evolving, have a large human element and, while they are loosely organised, they have a strong interaction through which they work towards a common goal.

Based on a survey of SoS definitions, Sharawi et al. [2006] propose SoS characteristics that are considered either essential or desirable with respect to the modelling and simulation of SoS, for which essential concepts are independence, interoperability, and a global goal, while a characteristic such as distribution is considered desirable.

2.8. Other System Types Related to SoS

Different kinds of systems and notions exist in the literature that bear a resemblance to SoS by their name, structures, or characteristics but are not identified as such.

Sage and Cuppan [2001] describes a Federation of Systems (FoS) as having many of the same challenges as SoS, but they are much more heterogeneous with respect to the "trans-cultural and trans-national socio-political" dimensions. Krygiel [1999] describes a FoS as a type of SoS, but with a very high degree of heterogeneity, autonomy, and greater geographic distribution.

Ultra-Large-Scale systems (ULS) are described as having most of the characteristics of an SoS, but in a much larger scale. Instead of being seen as an SoS they are regarded as sociotechnical ecosystems on a large scale, as they involve multifaceted interactions between people and technology that reside inside an environment of massively

complex technical and managerial challenges, in which an "erosion of the people/system boundary" will occur [Northrop et al. 2006].

A comparable system type is described in the UK-based Large-Scale Complex IT Systems initiative that focused on the challenges arising from the integration between large, complex IT systems [Sommerville et al. 2012]. These systems consist of a collection of new and existing independently owned and managed software systems that work together, which raise high demands for the interactions between systems, organisations, and people. A Large-Scale Complex IT System consists of systems, potentially systems of systems in their own right, that work together out of mutual interest but are owned by individual organisations that may be competing. This means that they would be categorized as a virtual SoS in the categorization presented in Section 3.10. Sommerville et al. directly address virtual SoS that, however, is found unintuitive, as "virtual" is found to be too ambiguous a term and the "system of" in SoS is seen to imply something that has been intentionally designed to perform a purpose in an organization, using Checkland's definition of a system [Checkland 1999]. Instead, the term a "coalition of systems" is used to describe systems that work together out of mutual interest without being originally design for this purpose and often having hostile relationships between stakeholders.

3. DIMENSIONS OF SOS

As Section 2 suggests, there is no single precise definition of SoS, but the literature offers a rich set of descriptions of SoS properties that allow fine distinctions to be drawn between SoS instances. Equally, these many approaches share common concepts. What do these mean for the use of model-based techniques to develop and maintain SoSs and constituent systems? We argue that the "space" of SoSs might usefully be described in terms of several dimensions based on shared concepts that have a bearing on modelling and analysis. The intention is that positioning an SoS engineering problem in the space defined by these dimensions might suggest *modelling patterns* and *analysis approaches*. In this section, we propose eight such dimensions derived from terms used in the literature, taking account of the contexts in which the terms have been used. For example, the term "independence" may denote independence of operations for self-governing constituents [Maier 1996; Crossley 2004], the independence of capabilities in terms of the variances in resources of constituents [Karcanias and Hessami 2010], or it can refer to independent optimisation of the constituents [Sage and Cuppan 2001].

3.1. Autonomy of Constituents

Autonomy is the extent to which a constituent system's behaviour is governed by its own rules rather than by others external to the constituent. This is especially seen as a result of individual ownership of the systems. The property of managerial independence identified by Maier (Section 2.2) entails that constituents perform their own functions in accordance with their own rules, while also participating in the SoS. Autonomy of constituents is central to SoS Engineering as described by Keating et al. [2003]. For Boardman and Sauser [2006], autonomy refers to the capacity of a constituent system to pursue a specific purpose: constituents that were conceived as parts that exhibit no autonomy are really enabling elements of the SoS, rather than (constituent) systems in their own right. Cook [2001] acknowledges the need for independence in Maier's sense, and also identifies a requirement for constituents to be "purposeful" and set their own goals.

Given the heterogeneity of an SoS, there is likely to be considerable variation in the autonomy exhibited by constituents. Modelling and analysis techniques need to permit the expression of a range of actions that an autonomous constituent may perform, but

that may not be precisely predicted at the SoS level. This suggests that there is a need for looseness or underspecification of constituent system behaviour.

3.2. Independence

Independence is the capacity of constituent systems to operate when detached from the rest of the SoS. This is Maier's "operational independence" characteristic, also identified by Krygiel [1999] as the capability for independent action and by Jamshidi [2008] as the extent to which systems are "independently operable." Independence of both design and operation is key to the SoS definition offered by Sharawi et al. [2006].

Independence implies that a given constituent system may offer a range of behaviours, some related to its role in an SoS, and others independent of it. The relationship between these classes of behaviour, and specifically the dependencies between them, might be hidden from the SoS engineer. Model-based techniques therefore need to be able to support information hiding.

3.3. Distribution

Distribution refers to the extent to which constituent systems are dispersed so that some form of connectivity enables communication or information sharing. Distribution may denote a geographical distance such as "arrays" of systems dispersed over wide geographical areas as described by Shenhar [1994] and Maier's geographical distribution characteristic [Maier 1996]. In Kotov's characterisation of SoS, it is clear that distribution may refer to a network distribution of concurrent processes as well as physical separation [Kotov 1997]. Manthorpe [1996], considering joint military operations, identifies the need to accurately spread data to thousands of locations enabled by mobile platforms and sensors.

Modelling frameworks that support distribution require the ability to assign constituent system processes to computational infrastructure, linked by a communication medium. Descriptions of concurrency, communication, and particularly failures of communication media, are necessary.

3.4. Evolution

Many SoSs are long-lasting and subject to change, whether in the functionality delivered, the quality of that functionality, or in the structure and composition of constituent systems. Maier [1996] identifies evolutionary development as a key characteristic, and Carlock and Fenton [2001] identify the lack of a permanent state of the SoS. Carney et al. [2005] view evolution as taking place through a series of largely deliberate preservative or adaptive interventions caused by, for example, upgrades to constituent systems or a need to respond to an evolving environment, as identified by Crossley [2004] and Despotou et al. [2003]. Abbott [2006] emphasises that an SoS is "continually evolving, but slowly." Bloomfield and Gashi [2008] observe that evolution is "incessant."

Model-based approaches to SoS engineering require support for gaining assurance of the preservation of specified properties under evolution steps. We may characterise this as a need for verification of conformance of a constituent system's interfaces to those of the other constituents with which it must interact. Evolution may be manifested as updates to constituent systems, requiring reverification of conformance.

3.5. Dynamic Reconfiguration

Dynamic reconfiguration is the capacity of an SoS to undertake changes to its structure and composition, typically without planned intervention. Several authors identify the ability to undertake this kind of real-time change as an important characteristic, especially in ensuring resilience of an SoS to faults and other threats. Boardman and Sauser [2006] identify the need for "dynamic determination of connectivity," requiring

the autonomy of the constituent systems to deliver the functions required to disconnect and reconnect constituent systems, and to modify interfaces. Crossley [2004] regards SoS as dynamic entities, while Schneider and Trapp [2009] discuss approaches to the use of runtime safety models to enable dynamic reconfiguration of open SoSs.

In contrast with evolution, which refers to the capacity to support planned changes on a slow scale through intervention, this dimension refers to the technical abilities an SoS has to change its composition during operation, such as on-the-fly swap-in and a pluggable architecture. To support the dynamic reconfiguration, SoS models must have abstractions for the dynamic modification of architectures and interfaces, and the capacity to reason about such changing structures.

3.6. Emergence of Behaviour

Within the SoS literature, emergence refers to the behaviours that arise as a result of the synergistic collaboration of constituents. Reliance is typically placed on the delivery of some emergent behaviour in order to deliver a higher functionality than delivered by the constituents separately. Several significant papers, including Maier [1996] and Boardman and Sauser [2006], refer directly to the need for emergence, and Abbott [2006] develops the characteristic of SoS being "open at the top" in this sense.

The reliance placed on emergence establishes important demands on modelling and analysis methods. It is vital that global properties can be described at the SoS level; it will often be the case that an emergent property on which SoS stakeholders depend can only be sensibly articulated at the SoS level, and not at the level of constituent systems. Modelling and analysis tools should permit the statement and verification of emergence, and permit identification of emergent behaviour that one would like to avoid, such as feature interactions [Zave 1993].

3.7. Interdependence

Interdependence refers to the mutual dependency that arises from the constituent systems having to rely on each other in order to fulfil the common goal of the SoS. If the objective of a constituent system depends on the SoS, then the constituent system itself may have to sacrifice some of its individual behaviour in order to meet the requirements of joining the SoS. Examples of terms that have been joined under "Interdependent" are interrelationships [Krygiel 1999], interdependencies [Sage and Cuppan 2001], interdependency [Bar-Yam et al. 2004], and belonging [Boardman and Sauser 2006].

Including both "Independence" and "Interdependence" may appear contradictory. However, some authors take the view that an SoS requires trade-offs between the degree of independence in the constituent systems and the interdependence required to reach the common goal [Sage and Cuppan 2001; Bar-Yam et al. 2004]. So while the individual constituent systems are independent, the relations and interoperability between requires some degree of interdependence.

Modelling and analysis techniques should allow the explicit identification of interdependence, the tracing of mutual dependencies, and the ability to use these links to assess the impact of constituent system changes.

3.8. Interoperability

Interoperability refers to the ability of the SoS to incorporate a range of heterogeneous constituent systems. This involves the integration and adaptation of interfaces, protocols, and standards to enable bridging between legacy and newly designed systems. The interoperability concept appears in the literature in the notions of simultaneous functioning [Shenhar 1994], integration of capabilities [Manthorpe 1996], interoperability and integration [Krygiel 1999], heterogeneity [Carlock and Fenton 2001; Cook 2001], "open at the bottom" [Abbott 2006], and diversity [Boardman and Sauser 2006].

Table I. Mapping Concepts to SoS Dimensions

Author(s)	Described in Section	Autonomy	Independence	Distribution	Evolution	Dynamic Reconfiguration	Emergence of Behaviour	Interdependence	Interoperability
Boulding [1956]	2.1	•	•					•	•
Ackoff [1971]	2.1	•						•	•
Eisner et al. [1991]	2.5			•			•	•	•
Noam [1994]	2.4	•	•	•			•		•
Shenhar et al. [1994]	2.3			•			•		•
Manthorpe [1996]	2.4				•		•		•
Maier [1996]	2.2	•	•	•	•		•		
Kotov [1997]	2.4		•	•					
Lukasik [1998]	2.5			•	•		•		•
Krygiel [1999]	2.7			•	•		•		•
Roe [1999]	2.5			•	•		•		•
Cook et al. [1999]	2.7	•	•	•	•		•		•
Pei [2000]	2.4			•		•			•
Carlock and Fenton [2001]	2.4		•		•		•	•	•
Sage and Cuppan [2001]	2.7	•	•	•	•				•
Chen and Clothier [2003]	2.5	•	•		•	•	•	•	•
Keating et al. [2003]	2.5	•	•			•	•	•	
Bar-Yam et al. [2004]	2.7		•		•		•	•	•
Crossley [2004]	2.5		•		•	•			•
DeLaurentis and Crossley [2005]	2.3	•	•	•	•	•	•	•	
Abbott [2006]	2.2				•	•			
Boardman and Sauser [2006]	2.2	•		•	•	•	•		•
Cocks [2006]	2.4	•		•	•		•		
Boehm [2006]	2.4				•		•		
Fisher [2006]	2.7	•	•	•	•		•	•	
Sharawi et al. [2006]	2.7	•	•	•	•	•	•		•
DoD SE Guide for SoS [2008]	2.6	•	•	•	•	•	•	•	•
Karcanias and Hessami [2010]	2.3	•	•	•	•		•	•	
INCOSE [2015]	2.6	•	•	•	•	•	•	•	•
Count	–	16	17	19	21	10	22	12	20

The need for interoperability places several requirements on modelling and analysis methods; it reinforces the need for techniques supporting the verification of conformance of constituent system interfaces. Models of SoS exhibiting a need for interoperability are likely to incorporate heterogeneous models of the constituents. Mechanisms for ensuring semantic consistency of diverse models, and rigorous analysis of those very distinct model types, are required in order to meet the needs for verification of both emergence and conformance.

3.9. Mapping Concepts to SoS Dimensions

Table I shows how the eight dimensions are mapped to the descriptions, definitions, and characteristics the original authors have used to perceive and explore the SoS

domain. The purpose of the mapping is to visualise the distribution of the dimensions in relation to both author and year. The mapping shows that some dimensions are used more than others, with "Evolution" and "Emergence of Behaviour" as the most frequent. This result is not unexpected as both are matters that are difficult to grasp and for which there is still limited knowledge in a systems development context. Therefore, they get more attention in defining publications, while areas such as "Dynamic Reconfiguration" and "Distributed" systems are more researched. This, however, does not mean that the other dimensions are not important in an SoS context; instead, it should be seen as an indicator of how they are regarded. For instance, "Evolution" and "Emergence of Behaviour" are properties sought for or managed in SoS engineering, while "Independence" and "Autonomy" can be seen as properties of the constituent systems, indicated by their ownership.

The mapping also shows diversity, as only a few authors have the same marks for the eight terms. This might suggest vagueness in SoS as a concept. Looking at the literature there is some truth to this, but it is also worth mentioning that it seems to be the tendency that the newer publications include more of the eight dimensions than earlier publications, and that the engineering handbooks contain the vast majority of the eight. A reason for this diversity can be that the SoS literature derives from multidisciplinary collection of researchers, developers, and managers, each approaching the SoS field from their specific background. Some take an operational point of view, others a focus on management, while this survey has a modelling perspective.

This survey has studied a large part of the literature in the search for SoS characteristics, as seen from a modelling perspective. This does not entail that the results in Table I cannot be used by other perspectives or as an identification of SoS in general. It is merely important to bear in mind in what perspective the table was created. The eight dimensions in Table I are not intended to define a boundary separating SoSs from non-SoSs, but allow an individual candidate SoS to be characterised in a way that may be useful in determining the model-based engineering techniques that are applicable.

In the remainder of this article, these dimensions are used to structure the discussions on current practice, research challenges, and the anticipated steps needed for a strengthened SoS engineering discipline. This is done within the four areas of modelling and architectures, simulation, testing, and verification. The eight dimensions will be used as reference points to ensure that they are considered with relation to each of these four areas.

3.10. Categorisation

The U.S. Department of Defense makes use of four categories of SoS [OUSD(AT&L), DoD 2008]:

Directed. The SoS is built to fulfil specific purposes. Constituent systems have the ability to operate independently, but are managed to satisfy a concrete purpose.

Collaborative. The constituent systems are not compelled to follow a central management, but voluntarily participate in a collaboration to fulfil the goal.

Acknowledged. The SoS recognises a common purpose and goal, while the constituent systems retain independent control and objectives. Evolution of the common purpose is based on collaboration between the SoS and the constituent systems.

Virtual. The SoS is without either managerial control or a common purpose. This makes the behaviour and the fulfilled goals highly emergent, but also entails that the exact means and structures producing the system functionality are difficult to discern and distinguish.

Three of these categories were originally defined by Maier [1996], while the "Acknowledged" type was later proposed by Dahmann and Baldwin [2008]. Maier argues that SoSs should be considered equivalent merely because they have a similar complexity and scope. A categorisation is required in order to guide the selection of architecting principles. The categories are based on the degree of managerial control because this determines how adaptable and cooperative each constituent system will be with respect to requirements, interfaces, data formats, and technologies. In turn, this influences the challenges faced when constructing the SoS.

In practice, the "Directed" SoS embodies a form of planned emergence because the constituent systems are centrally managed. The other types of SoS have little or no centralised managerial control. The "Collaborative" type has the notion of a centralised management but with very limited or no powers to enforce decisions, while the "Virtual" type is without any degree of management. Dahmann and Baldwin [2008] adds the "Acknowledged" type to describe the scenarios found in many military systems. This type is focused on establishing collaborative management at the SoS level, while keeping the managerial and technical independence at the constituent level. The goal is that autonomy and ownership are maintained, while at the same time ensuring that changes can be collaborative and decided upon on the basis of some common objectives. In practice, one would not expect the SoS type to be uniform within an SoS, but local subsystems of differing types might arise. Indeed the heterogeneity of constituent system ownership could be expected to lead to a wide range of (possibly inconsistent) stakeolders' views of the levels of control actually offered within an SoS.

We hypothesise that differing degrees of control might be reflected in relatively stronger or weaker models of constituent system interfaces. Directed and collaborative SoSs would require a relatively strong specification of a central decision-making authority. Whilst work such as that of Ingram et al. [2014] draws some correspondences between modelling patterns in SysML and SoS types, a theory of SoS types embodied in rigorous models remains to be developed.

4. ILLUSTRATIVE EXAMPLES

The literature on SoSE covers a wide range of application domains. In this section, we review a few applications described in the literature in order to illustrate the SoS domain.

4.1. Transportation

Transport networks are often composed of independently owned and operated systems that are geographically distributed. DeLaurentis [2005] illustrates how transportation can be seen as an SoS covering all the eight dimensions from Section 3, emphasising the evolutionary nature and the emergent properties as the main reasons why the transportation sector needs SoS modelling and analysis techniques.

Air traffic management systems are often cited as examples of de facto SoSs. For example, Ball Sr [1997] and Geddes et al. [1998] consider the movement to a "Free Flight" model of air traffic operations in the United States as a transition from a directed to a more distributed and collaborative SoS structure. Air traffic management routinely has to balance competing concerns that dominate to different extents among stakeholders. It is argued that these changes necessitate explicit mechanisms for the following: promoting initiative, sharing of purpose, situation and planning; and efficient communications. It is argued that the current barriers to collaboration suggest that explicit support for the transition is required, and that a system of "associates" following explicit sharing protocols can help to resolve them.

Mansouri et al. [2009] present a systematic discussion of the derivation of a framework for the management of maritime SoS for the U.S. Maritime Transport System

(MTS). The MTSoS is represented in terms of five "ABCDE" characteristics derived from Boardman and Sauser [2006]. Opposing forces are considered on each dimension. For example, the "Autonomy" dimension represents a tension between conformance and independence; "Belonging" balances centralisation and decentralisation; "Connectivity" ranges from platform-centric to network-centric; "Diversity" from homogeneity to heterogeneity; and "Emergence" ranges from the foreseen to the indeterminable.

4.2. Smart Energy Grids

Smart grids integrate data on energy supply and consumption in order to deliver electricity in a cost-effective manner. There is a need to provide assurance of safety and security of supply, and in doing so there is a significant engineering challenge in balancing dynamically changing availability and price from independent and autonomous suppliers against varying demand [European Commission 2012]. There have been some implementations of smart grid technology, such as the Smart Grid/Distribution Network Management System developed as part of the Customer-led Network Revolution project in the United Kingdom,[8] which integrates computation and network technology on a power generation and distribution network with multiple independent energy suppliers and consumers.

From the perspective of model-based approaches, the heterogeneity of smart grid systems is striking. Smart grids are cited as examples of cyberphysical systems requiring novel combined models, for example, in Dillon et al. [2011] and Ilić et al. [2010]. Miller et al. [2012] use simulation to conduct a sensitivity analysis based on a model that captures the physics of demand response and the behaviours of humans both as individuals and in networks. Agusdinata and DeLaurentis [2008] discuss the role of models in policy-making for the energy sector, illustrating the trade-off between the levels of resolution afforded by models and the ease with which high-level views of SoS behaviour can be obtained. However, few if any model-based studies in SoS link all three types of element: cyber, physical, and human.

4.3. Emergency Management and Response

The agility required of an emergency response SoS, and the often unique circumstances of each substantial emergency, lead to astronomically large models. In order to manage this, Liu [2011] proposes a hierarchical SoS for emergency response in China. Even with such a hierarchical approach, modelling of such a complex structure is only at an early stage. In more constrained environments, simulation can be used to analyse SoS successfully. For example, Mahulkar et al. [2009] report the construction of a substantial MATLAB-based model of an SoS on board a naval vessel in order to explore the consequences of changes in the on-board technology using simulation based on emergency response scenarios. Emergency response technologies, based on networked sensors and actuators, provide a significant application area. Daniel et al. [2009] present an architecture for swarms of micro-Unmanned Aerial Vehicles (UAVs) sensing and mapping toxic atmospheric emissions following an incident such as an explosion or fire.

4.4. E-commerce

An example of SoS that many people encounter on a near daily basis is e-commerce, the buying and selling of goods and services via online shopping. E-commerce involves a number of different independent systems working together in order to reach the overall goal of sale and delivery. These systems perform functions that provide the

[8]See http://www.networkrevolution.co.uk/. The network management system operates areas of network from deployed on networks serving about 20,000 customers, using 17 smart enabled network interventions, as well as central and distributed controllers.

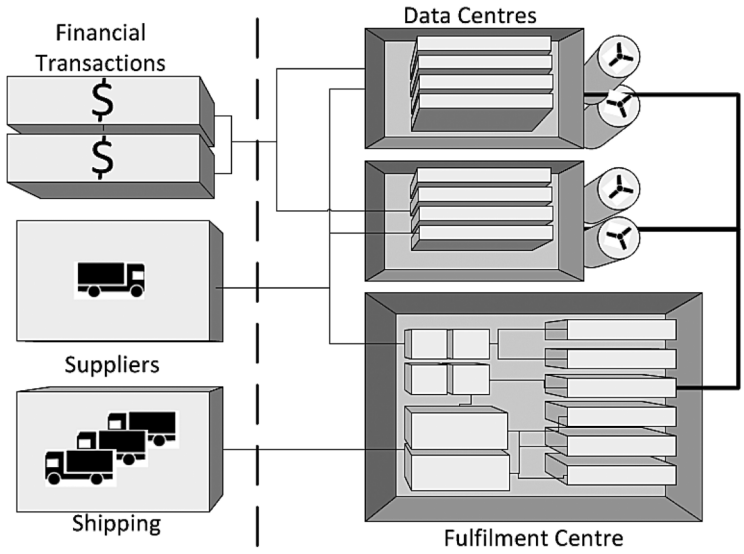

Fig. 1. Illustration of constituent systems in an e-commerce business.

virtual market place, the handling of payment transactions, management of inventory and supply chains, as well as handling shipping and delivery [Ricker and Kalakota 1999]. Making all of these play well together requires an infrastructure that contains the dimensions of a SoS. One of the premier E-commerce companies, Amazon.com requires a highly scalable platform that can provide the performance, reliability, and efficiency needed to support millions of customers [DeCandia et al. 2007]. Amazon's consists of a highly decentralized architecture consisting of hundreds of services and systems that need to integrated as seamlessly as possible.

Figure 1 illustrates the multitude of systems involved in an e-commerce. The right side of the dotted line contains the systems that are under the e-commerce company's ownership, while the left shows systems that are owned by other stakeholders. Large e-commerce companies will have a number of fulfilment centres to physically handle inventory, picking and packaging, as well as multiple data centres to deliver the data storage and processing power required.

In order to run everything from sales, pricing, to warehousing as well as handling a large number of suppliers, the e-commerce companies have a wide range of systems that are under their ownership. All of these systems, which may either be developed in-house or by a third party, all work towards the same goal and the integration between them is key to the efficiency of the company. They do not, however, express all of the dimensions of an SoS. Having ownership of their systems enables the companies to have better control of the development strategies, a greater degree of flexibility, and faster decision-making than they have with systems owned by third parties. There are still vital systems that will be independently owned by other businesses, such as payment transactions as well as the systems of suppliers and shipping companies. This means that e-commerce companies are dependent on distributed development strategies planned for the individual systems and on the quality of relationships and communication between systems owners. This is where a combination of multiple SoS dimensions makes the development much more challenging.

4.5. Observations

The SoS engineering literature covers a very wide range of application areas beyond those illustrative examples mentioned previously, including communication systems [White and Jean 2011] and healthcare [Wickramasinghe et al. 2007]. It is worth noting that SoSs are not always explicitly identified as such. Many of the examples in the literature illustrate "SoS Thinking" for the derivation of management frameworks, rather than the technological activity aimed at developing ICT artefacts like networks, machines, or software. We find comparatively few accounts of applying model-based SoS engineering techniques "in the wild"; the vast majority of application reports remain at the proof of concept or pilot study stages.

5. STATE OF THE ART

Having reviewed a range of systems that form the subject of the SoS literature, we focus on the state of the art in model-based techniques. In this section, we examine the state of the art in modelling and architectural description (Section 5.1), simulation (Section 5.2), testing (Section 5.3), and verification (Section 5.4).

5.1. Modelling and Architecture

As indicated in Section 1, modelling of systems, or specific aspects of systems, has been carried out for many years in many different disciplines, and this has naturally spread to SoS engineering [Cantot and Luzeaux 2011]. Models can be developed at a range of abstraction levels, depending on their purpose and the forms of analysis that are to be performed on them [Kramer 2007]. In order to gain the maximum engineering value from model-based methods, there need to be well-defined relations between models and their realisations. We will say that an implementation I *refines* a model M, written $M \sqsubseteq I$ if the properties shown to hold of M also hold of I.

Models can be expressed in many forms ranging from graphical sketches or text to mathematical formalisms. In this article, we will pay particular attention to models that are expressible in a form that can be given an unambiguous semantics to permit machine-assisted confirmation or refutation of properties of interest. The state of the art of model-based SoS engineering is generally such that models are developed using existing notations and tools tailored to the application at hand. There have been few attempts to define general-purpose languages for model-based SoS engineering [Ludwig et al. 2011], although Woodcock et al. [2012] have proposed a formal language for expression of refinable models of SoSs. The difference between a formal language for expressing refinable models of SoS, as distinct from a language for expressing refinable models of systems, is that the former must directly address the different dimensions of SoS described in Table I.

5.1.1. Enterprise Architecture. In the context of this article, the term "architecture" covers not only structural properties such as system hierarchies and interfaces, but also functional properties that may be affected by architectural decisions and changes. In the existing literature the direct usage of the terms "architectural models" and "architectural modelling" in relation to SoS is rather sparse. Only a few authors have a direct focus on SoS architecture, and mostly in relation to Enterprise Architectures [Lucke et al. 2010].

Different Enterprise Architecture frameworks have been developed over the years, including DoDAF and MODAF, originally to support the system engineering discipline. They are used to create representations of large enterprises and to create methodologies for capturing their structure and dynamics. Looking at the current state of Enterprise Architectures a number of authors aim at generalising such enterprise architectures to make them applicable to the engineering of SoSs [Carlock and Fenton 2001; Harmon

2005]. However, in general, these papers do not take into account the real challenges of engineering SoSs, indicated by the SoS dimensions listed in Section 3.

5.1.2. Architectural Frameworks. Looking outside the scope of Enterprise Architectures, Kilicay-Ergin and Dagli [2008] see a limitation in using static frameworks and methodologies for architecture development of SoS. These do not provide a way to analyse dynamic evolution of system state or behaviour, and consequently they point to a shift towards executable models. Here they take our *Autonomy* and *Evolution* dimensions into account. In conclusion Kilicay-Ergin et al. determines that simulation tools are needed that combine both structural and executable models, in order to include all of the behavioural views needed to comprehend an SoS.

Selberg and Austin [2008] describe the evolution from systems to SoSs and the associated unmanageable increase of complexity. Here, the SoS characteristics *Evolution* and *Emergence of Behaviour* are considered explicitly. This article demonstrates how using more standardised interfaces will better prepare for the integration of new constituent systems in the future. In addition, there is a recommendation for the use of general-purpose formal models supporting abstraction and analysis capabilities. However, there is still a lack of formal methods supporting proper engineering of SoS.

5.2. Simulation

One of the most frequently used forms of model analysis is execution in some form. Typically, this is called model simulation, indicating that it is an approximation of how the real system would behave in the same scenario. In the existing literature on model simulations of SoSs, the focus is primarily on using the models for training purposes for different kinds of personnel. Because of this, the majority of the publications on SoS simulation are also focused on the distribution of simulators connected together and abilities to make use of agent-based systems (typically used to simulate different kinds of people operating the systems).

5.2.1. Agent-Based Simulation. Agent-based technology is typically included when it is desired to include the human in the simulation loop [Axelrod 1997]. Thus, such intelligent agents are primarily used to explore sociotechnical aspects in an SoS setting. This has also been used to explore different design choices for internal interoperability in a ship environment [Mahulkar et al. 2009]. Gutierrez-Garcia et al. [2009] enhance the agent-setting with formally expressed constraints with an emergency management case study. These agent-based technologies support the *Autonomy* dimension very well.

5.2.2. Focused Simulation. Distribution aspects in a simulation setting typically focus on interoperability between simulations of different constituent systems. Here, the majority make use of High-Level Architecture (HLA) simulation interoperability [Lees et al. 2007], which has been adopted as an IEEE standard [IEEE1516 2010]. HLA has primarily been used for high-fidelity, defence-related simulations coupling different existing simulators together to examine, for example, different possible war scenarios. However, HLA has also been used in an SoS setting for exploring design choices around the weight of an aircraft [Sharawi et al. 2006]. HLA primarily improves the *Interoperability* dimension of SoS.

Within the area of modelling and simulation, a large part of the existing literature is military focused. For example, the U.S. Future Combat System (FCS) has been modelled and simulated using a specially developed SoS Application Toolkit (SoSAT) for military missions [Campbell et al. 2005; Eddy et al. 2007]. This simulation uses Statemate [Harel 1987] and fault tree analysis to conduct probabilistic simulations focusing on potential faults appearing in constituent systems.

In other works, professional simulation engines, such as ExtendSim[9] have been used [Jian et al. 2010]. These works are primarily focussed on limiting the SoS from undesirable *Emergence of Behaviour*. Jian et al. uses the simulation engine for the evaluation of SoS architectures using a two-level process. The higher level is focused on capability requirements and on computing the SoS overall measure of performance. At the lower level the quality of the capability requirements is obtained by simulation of the SoS. The model simulates the interactions between the component systems in order to assess the components' ability to meet the specified capability requirements.

5.2.3. General Simulation. A few examples can be found that attempt to create a general modelling and simulation approach aimed directly for SoS. One such example can be found in Kotov [1997], where a C++-based library is presented for modelling and simulation in an SoS setting. Here, constituent systems are hosted at multiple computing nodes and the communication between these is incorporated. A similar generic approach extending the Unity language to an SoS setting can be found in Gamble and Gamble [2008]. All of these works can be seen as enhancing the *Interoperability* between different models of constituent systems.

Sahin et al. [2007] presents a framework for the architectural representation and simulation of an SoS based on Discrete Event System Specification (DEVS) and the exchange between constituent systems defined in XML. DEVS is also used by Berenji and Jamshidi [2011] in a fuzzy-control setting for independent vehicles acting in a swarm context. DEVS is a formalism for describing and simulating hierarchical systems via components and their interconnection including the runtime addition/removal of couplings and components. Support for HLA is also available. The use of DEVS in an SoS setting is explained in Mittal et al. [2009]. This is an example of a practice aiming at increasing the *Interoperability* between models of constituent systems in an SoS, as well as investigating the *Dynamic Reconfiguration* of the system.

SoS simulation capabilities using formal models of SoSs at a higher level of abstraction is found in the CML tool, Symphony [Coleman et al. 2012]. Ideas are also present to cope with simulations of evolution of SoSs dynamically [Nielsen and Larsen 2012].

5.3. Testing

5.3.1. Compositionality Versus SoS System-Level Testing. Considering the time and effort required for SoS system-level testing, it is a scientifically valid question whether it could be made redundant by means of V&V activities performed on the constituent systems that are part of the SoS: the concept of *compositionality* has been elaborated in the field of formal methods and specifies conditions allowing one to deduce emergent properties of the complete system from the "local" properties of its constituents [Hoare 1978; Roscoe 2010]. Indeed, the distribution and local autonomy of constituent systems facilitate the application of compositional arguments, because prerequisites like absence of shared resources (e.g., variables, processors) needed to apply such an argument are generally fulfilled.

These considerations, however, obviously contradict practical experience with SoS testing on system level, where it quite frequently turns out that the composed system does not fulfil its expected emergent properties. This experience is not caused by the preconditions for the compositionality argument being violated. Instead, there are quite different reasons for SoS to fail on the system level:

[9]www.extendsim.com.

(1) Crucial nonfunctional emergent properties—in particular, safety and security [Leveson 1995]—are noncompositional. As a consequence, the combination of safety or security mechanisms does not necessarily lead to a safe or secure system.
(2) Constituent systems frequently show a lack of quality at the point in time when they are first delivered for the purpose of integration testing, so system-level tests simply fail because some constituent systems do not meet their specifications.
(3) Constituent systems may show erroneous behaviour when operating in an SoS configuration, due to undocumented assumptions made by the subsystem suppliers, which are not fulfilled in the SoS configuration.
(4) System-level tests fail because emergent properties have been insufficiently captured during the requirements elaboration phase.

It is the purpose of SoS system-level testing to reveal these deviations, as will be described in the following paragraphs (see also Peleska [2013]).

5.3.2. Coordination of Testing Activities. Using the terms of the military domain, testing activities of constituent systems may be structured into *developmental test and evaluation (DT&E)*, which is a verification activity, and *operational test and evaluation (OT&E)*, which is a validation activity [OUSD(AT&L), DoD 2008, p. 43]. As emphasised in [OUSD(AT&L), DoD [2008, p. 11] and Colombi et al. [2008], the managerial independence between constituent systems generally does not allow one to synchronise lifecycle activities between them. As a consequence, system integration testing on SoS level cannot rely on all constituent systems being ready for this task at a given point in time. The problem becomes even more severe if some constituent systems are members of more than one SoS: at the point in time a system test is to be performed for SoS 1, a constituent system may be occupied with changes targeted at its operation in SoS 2 configurations. As a consequence, a synchronisation point where all constituent systems of a SoS are in a baselined state that is ready for system integration testing might never be reached.

This problem can be mitigated by *interoperability test* campaigns. Interoperability testing verifies two or more constituent systems with the following objectives:

—Ensure their basic capabilities to exchange data over the intended interfaces.
—Verify that the synergetic properties expected from this cooperation are realised in conformance with the SoS requirements.

The latter V&V activity is called *end-to-end* testing, since SoS functionality is investigated along the complete processing chain, from the initiating constituent system to the systems supporting and finally to those utilising the established results.

In a systematically structured SoS test campaign it is first ensured that constituent-level tests have been successfully completed to ensure systems comply with their specifications. In particular, it should be ensured that *conformance tests* have been performed in order to verify that each constituent system conforms to the applicable (communication and/or functional) standards. Acceptance testing should ensure that system-specific functional, structural, and nonfunctional properties are fulfilled. Then, interoperability testing investigates whether the constituents cooperate adequately to ensure the emergent SoS properties required.

5.3.3. Coping with Complexity. Liang and Rubin [2009] address the combinatorial explosion problem caused by the size of SoS state and input vectors by adopting the concepts of *pairwise testing* and *orthogonal arrays* for SoS system-level testing. These concepts have been widely applied in software testing. Pairwise testing with orthogonal arrays advocates test data generation according to the strategy by Tatsumi [1987], which goes back to Taguchi's original ideas on robust design [Taguchi 1987; Phadke 1989]:

(1) Identify the input and state parameters influencing the System Under Test (SUT) behaviour (these parameters are called *factors* in the Taguchi Method). (2) Partition each single input or state vector component into equivalence classes (called *levels* in the Taguchi Method). (3) Select level combinations, that is, vectors of values, each taken from an equivalence class of the associated factor.

The selection of level combinations is typically performed using the orthogonal array approach. This is a method for selecting level combinations that are balanced in the sense that all combinations of a given dimension (n-tuples of levels associated with n factors) occur an equal number of times. Typically, the orthogonal array method is applied to pairs of different factors, so that for each pair the associated levels are exhaustively combined, and each pair is exercised the same number of times ($n = 2$).

As may be expected (also in the light of several critical evaluations of the method, such as Bach and Schroeder [2004]), pairwise testing does not solve all complexity issues in SoS testing, as will be discussed in more detail in Section 6.4.

In the light of SoS dimensions (see Table I), *Autonomy* as well as *Independence* of constituent systems is reflected by independent developmental and operational tests. Conformance and *Interoperability* testing addresses the distributed nature of SoS. *Evolution* of constituent system functions is addressed by coordinated constituent-level and SoS-level testing campaigns, as long as the constituent system functionality affects the SoS under consideration. *Emergence of Behaviour* is checked by means of end-to-end tests.

5.4. Verification

The literature on verification of SoSs is rather thin, reflecting the fact that research in this area is still in its infancy. However, there is an interest in introducing formal verification at different levels of software architectures in an SoS setting [Michael et al. 2009]. In this section, we discuss why this is the case, given that systems development is relatively well developed in comparison, as we review the key developments in SoS verification.

5.4.1. Runtime Execution Monitoring. There are two general strategies that can be used to verify a system: the first is to check in advance that the system has desirable properties; the second is to wait until runtime and check that an actual execution of the system performs satisfactorily. Runtime Execution Monitoring (REM) is in the latter category and is representative of methods for tracking the behaviour of an underlying application. It ranges from the analysis of simple audit logs through to sophisticated checking of states and transitions against formally specified assertions. Example uses include the verification of the NASA Deep Impact fault-protection engine [Drusinsky 2003] and the verification of the U.S. Ballistic Missile Defence System battle manager [Caffall and Michael 2004]. REM was chosen in the latter case because of its ability to scale and its support for real-time assertions. Much of the research on REMs involves temporal logic for specifications [Drusinsky 2003; Havelund and Rosu 2001] and there are both continuous [Maler and Nickovic 2004] and discrete-time applications [Drusinksy and Shing 2005].

5.4.2. Behavioural and Stochastic Verification. A key difficulty in SoS development and deployment is to verify required overall properties while individual component systems are heterogeneous and autonomous. Individual component systems tend to be extraordinarily large and diverse and an SoS relies for its successful operation on *Emergence of Behaviour* and feature interactions. It must be possible to predict the objectives of an SoS in a dependable way, in spite of different and potentially conflicting local goals. Calinescu and Kwiatkowska [2010] suggest that formal analysis and verification, including model checking, quantitative model checking, and quantitative analysis and

verification, are key technologies for the verification of SoS policies, contributing to SoS dependability management and assurance. They envisage a reconfigurable policy engine that uses online formal analysis and verification techniques for the implementation of a wide class of autonomic computing policies [Calinescu and Kwiatkowska 2009].

5.4.3. Contract-Based Specification. Payne and Fitzgerald [2010] argue that explicit contracts should be recorded at the boundaries between constituent systems in order to verify SoS-level properties. They survey different approaches to contract-based specification, both for behavioural and nonfunctional purposes, although they concentrate on the latter. Raising those results to an SoS level has also started [Payne et al. 2012].

In the design-by-contract paradigm, the emphasis is on specifying the interfaces between components, usually involving preconditions, postconditions, and state invariants [Meyer 1992] to document assumptions and commitments. The contract is this: if the interface user guarantees to meet the precondition, then the interface implementer guarantees to meet the postcondition. The implementer's guarantee might be partial correctness (if the interface operation terminates, then it is guaranteed correct), or it may be total (the operation will terminate and be correct). Invariants are used to document an abstract model satisfying these constraints. More sophisticated contracts deal with concurrency and shared resources. For example, rely and guarantee conditions may be used to document progress through intermediate states as well as reaching the final state [Jones 1983a, 1983b]. Reactive systems may be specified using reactive pre- and postconditions, where special auxiliary variables are used to record the history of past interactions and the readiness of current events [Woodcock and Cavalcanti 2001, 2002].

Beugnard et al. [1999] describe contracts at architectural levels, suitable for describing SoSs. In their work, contracts are structured into four layers: (a) The syntactic layer describes operation signatures. (b) The behavioural layer describes the effect of operations using preconditions and postconditions. (c) The synchronisation layer describes real-time scheduling of component interactions and message passing. (d) The quality of service layer describes nonfunctional aspects of operations.

5.4.4. Verifying Emergent Properties. The interaction of multiple system components sometimes leads to the emergence of unexpected properties as the aggregation of different features react in unpredictable ways such as in a basic telephone system where a telephone call involves precisely two parties, the initiator and the recipient. An extension to conference calling would allow many parties to enter and leave a call. An undesirable emergent property is that there may be no one responsible for paying for the call if the initiator is allowed to leave. This is known as weak emergence [Bedau 1997]; other examples occur in natural complex systems, including antforaging and bird flocking; it is a significant technical challenge to model such systems so as to reveal them. A classic example of weak emergence occurs in Conway's Game of Life, where global behaviours arise as the result of purely local rules. The game is played by a cellular automaton with four rules for simulating a population of entities distributed across the cells, whose life and death depends on the occupancy of adjacent cells. Polack and Stepney [2005] argue that the weakly emergent behaviour of the well-known glider pattern, made up from several entities, cannot be shown to be refined by the local rules governing the life of entities in cells. Sanders and Smith [2012] show that it can. They consider a different perspective on this problem: instead of trying to discover emergent properties of existing systems, they consider the development of an implementation with a required emergent property. Engineering emergent properties is seen as the software engineering activity of refinement: taking an abstract property and realising it in terms of local behaviours and component interactions.

5.4.5. Verifying SoS Dimensions. As noted at the beginning of this section, formal verification of SoSs is still in its infancy. This means that there is no best practice available for most of the dimensions mentioned in Table I; however, formal methods do already have something to say about these dimensions. As we have seen, autonomous systems can be verified using model checking and emergent behaviour using refinement. Research at the systems level suggests that the best way to tackle independence, interdependence, and interoperability is through compositional verification techniques, whilst distribution, dynamic behaviour, and evolution can be modelled using process algebra, Petri nets, and temporal logic and verified using theorem proving and model checking. Theorem proving and model checking has been developed for the SoS-specific modelling language CML [Coleman et al. 2012; Couto and Payne 2013].

6. CHALLENGES IN SYSTEM OF SYSTEMS ENGINEERING AND STRENGTHENING THE DISCIPLINE

Having identified the state of the practice in model-based techniques for SoS, the focus is shifted towards the research challenges in each of these areas and on how the challenges can be faced by examining the steps necessary to provide a strengthened foundation for model-based SoS engineering.

In this section, a short review of the literature on SoS challenges is presented (Section 6.1), followed by an examination of the challenges in modelling and architectural description (Section 6.2), simulation (Section 6.3), testing (Section 6.4), and verification (Section 6.5).

6.1. Research Challenges for SoS in the Literature

A research agenda for SoS architecting is presented by Valerdi et al. [2008], with 10 major research challenges linked to academic and industrial problems. The work focuses on the entire lifecycle of an SoS and includes considerations from, for example, buyers, developers, and maintainers. Some of the challenges include (a) evolution: research is needed to develop methodologies that can deal with evolving and emergent behaviours in SoS that dynamically adapt and absorb deviations in the system structure and (b) "System vs. SoS attributes" that denotes a range of trade-offs between attributes such as adaptability and modularity that can be simulated in SoS architecture models.

A different set of research challenges are presented by Maier [2005], that are foremost focused on the representation and analysis of SoS. Methods and tools that can be used for describing and analysing the "upper layers" of SoS are wanted. The upper layers refer to interactions among network elements where division of functionality and the interaction between systems are particular challenges.

The wish for better methods is also included by Ring and Madni [2005], who opt for better theory, methods, and tools in order to anticipate unintended behaviour in SoS operation.

These research challenges match well with those being proposed for system types that resemble SoS, such as for ultra-large-scale systems and large-scale complex IT systems. Northrop et al. [2006] see challenges in topics such as orchestrating activities between diverse stakeholders, in measuring and evaluating the effectiveness of system design, and in creating adaptive system infrastructures. This calls for more research in how to establish stronger support for system design at many levels of abstraction and in increased computational engineering through automated tool support for assessing the behaviour of evolving system compositions. Sommerville et al. [2012] complements this list by proposing research challenges related to the knowledge sharing between dispersed stakeholders, the creation of adaptable models based on real-time system monitoring data, handling resilience and recovery from failure, and on performing verification of systems with no firm specification. These research topics

are all directly applicable to SoS; however, Northrop et al. and Sommerville et al. have a stronger emphasis on research related to sociotechnical systems, human interaction, and the handling of acquisition and regulation policies, than what is seen in most SoS publications.

6.2. Modelling and Architecture

One of the main challenges for modelling languages aimed at SoSs is to ensure that they have well-founded semantics, as many of the modelling languages in the current practice do not [Fitzgerald et al. 2012]. In order to strengthen the discipline of the SoS engineering domain, creating modelling languages with a well-founded semantic basis is essential. Such well-founded notations will enable the conceptual descriptions and evaluations of SoS architectures. Only with the existence of such languages will it be possible to conduct justifiable analysis of conceptual descriptions of an envisaged SoS.

As SoSs have a high degree of *Distribution*, the modelling languages must be able to express the notion of multiple independent execution platforms that can be interconnected. A particular focus should be the languages' abilities to include *Interoperability* challenges as part of modelling the individual constituent systems and their relationships. Challenges connected to this would be how to define well-founded interfaces and express desired policies between the systems. The high number of connections occurring between the constituents in an SoS, calls for modelling languages that include central aspects of *Distribution* through a strong focus on consistency and communication faults. Connections that also entail a high degree of *Interoperability* concerns, that a modelling language must address as well. In order to handle these aspects it is necessary to have ways of defining well-founded interfaces for each of the constituent systems directly in models. Modelling languages that are effective for modelling distribution and those that can express clear interfaces need to be surveyed, in order to provide the knowledge needed to establish a modelling notation with a stronger focus at the SoS level.

In the current modelling efforts agent-based approaches are used to model the *Autonomy* of SoS. A main challenge of doing SoS modelling is to embed this aspect into more modelling languages by including constructs that enable users to describe autonomous behaviour; in particular, of how humans act in relation to the SoS. Therefore, it is key that the modelling languages include constructs that enable the description of such self-determining systems; constructs that can express the capabilities and responsibilities of a system such that it can be self-controlling by using behavioural and reasoning mechanisms and express self-coordinating behaviour based on observations. Likewise, modelling languages need constructs that can describe the *Dynamic Reconfiguration* in order for the models to express the dynamically changing infrastructures which enables the constituent systems to initiate and break their interrelationships. Modelling languages are needed that not only can express the dynamically changing infrastructure, but also model the interruption of data exchange, lost messages, and error handling.

It is clear that modelling an SoS does raise high demands for the modelling language itself. For instance, for a dimension such as *Evolution* it can be a challenge to include all the necessary system aspects in the language initially. This means that the model may not be capable of including important details as the language cannot express it. An SoS modelling language should have features that enable well-founded extensions of the language to be defined by superusers in a semantics preserving fashion. This is by no means trivial, so reaching the scientific basis for this is a long-term goal. Equally, *Emergence of Behaviour* is difficult to express in a model, as it is doubtful if an exactly described behaviour can be considered truly emergent. It is a challenging SoS characteristic to incorporate into a modelling language, as it cannot be accurately described. Instead, it is a characteristic that surfaces as a result of the other SoS

characteristics. In a modelling context the emergence itself can only be expressed as desirable or undesirable behaviours, while the behaviour itself only really becomes apparent through simulation of the model.

The aspect of *Independence* between the different SoS stakeholders is another SoS characteristic that can be challenging to include in a model. Not only can it be difficult to express how the varied individual ownership affects the constituent systems and the *Interdependency* between them, as seen in the e-commerce example in Section 4.4. It can also be difficult to establish trust between stakeholders that may not want to share their model or system details for confidentiality reasons, as will occur between the competing stakeholders as, for instance, in the example of Smart Energy Grids given in Section 4.2.

As the constituent systems in an SoS have independent ownership and will be separately developed it should be possible to work with divided models, such that each constituent system can be defined by its own model with well-defined interfaces. Not only will it allow for the constituents to be modelled independently, it will also facilitate the process of bringing the constituents together in the SoS. Having the *Independence* and *Interdependency* dimensions of the constituents included directly into the modelling language will improve the modelling effort. For instance, expressing the *Interdependency* between systems could be achieved by having rely/guarantee policies directly in the modelling languages.

Concerning architecture, Meilich [2006] focuses on net-centric environments to describe some of the challenges of SoS architecture for nondirected SoSs. These include (1) dependency issues between the constituent systems due to *Interoperability* characteristics, (2) *Emergence of Behaviour* of SoSs due to trade-offs between predictability and composability, and (3) collaboration between different *Independent* stakeholders of the *Distributed* constituent systems.

Approaches are being proposed that shift focus from optimisation to runtime flexibility and interoperability. Adaptive architectures are intended to enable system composition at runtime and to be more suited to the emergent behaviours found in SoSs. Having such a flexible and dynamic architecture may come at the price of predictability and potential lack of clarity on the system boundary. Model simulation is seen as a response to these challenges and the way of engineering an SoS is to "experiment as the system evolves."

As we indicated in Section 1, model-based engineering for SoSs is a young subject, and its limits have not yet been clearly mapped. The development of competent and faithful SoS models is dependent on the ability to observe the state and operation of the SoS itself with known levels of precision and accuracy, and remains an open research topic.

6.3. Simulation

Being able to perform simulations of SoS by using executable subsets of models is essential in the advancement towards a strengthened SoS engineering discipline. Simulations can be a very powerful tool in analysing and understanding the complexity and behaviour of an SoS.

The DoD guidelines recommend modelling and simulation as an effective means for grasping the complexity and *Emergence of Behaviour* of SoS [OUSD(AT&L), DoD 2008]. In order for the simulation of an SoS model to be efficient it must be able to show the characteristics of an SoS. If the main characteristics can be incorporated into the model, the volatile characteristic emergent behaviour has a much better chance of being detected through simulation of the model.

Simulation makes it possible to gain insight into the modelled system's functionality. One SoS characteristic that is particularly well suited for simulation is the aspect

of *Autonomy*. Simulating an SoS will show system behaviours and the constituents' interactions, which would be very difficult to predict by merely analysing the models or systems statically. An example of this would be the robot swarm systems described in Section 4.3. Kilicay-Ergin and Dagli [2008] have proposed a framework named "Artificial Life" that is aimed at analysing the influence architectural changes have on system behaviour. The framework consists of several layers that, among others, include computational intelligence tools, cognitive architecture, and a multiagent models level. While *Autonomy* can be simulated with agent-based simulation tools, further advances are needed for enabling users to interact with a simulation and reveal the autonomous behaviour occurring when humans act in relation to a specific SoS; for instance, via human-in-the-loop simulation.

An SoS simulation environment should also support the *Independence* of the constituent systems, by allowing both stand-alone and combined simulation of models. The constituent systems may be described in separate models that need to be simulated both individually and in combination with models of other parts of the SoS. This is particularly relevant in systems where competing concerns have to be balanced, such with the air traffic management systems used as an example in Section 4.1. Allowing each simulation to be performed individually for each constituent, while at the same time enabling a joint model to be simulated with other parts of the SoS would support the *Independence* of the constituent systems. As mentioned previously, confidentiality may be important as owners of one constituent system participating in the SoS may not wish to reveal the internals of their system through their model. In the same way the simulators will be challenged by model languages that will need to evolve in order to match the *Evolving* SoSs model.

An important outcome of doing simulations of SoS is to identify possible *Emergence of Behaviour*. Simulations may be probabilistic, rather than definitive, so the aim would be to demonstrate a low probability of undesirable emergent behaviour or a high probability of desirable emergent behaviour. This could be determined by observation of the simulation result; however, a more powerful capability would be if such behaviour could be flagged directly by the SoS simulation tool. The exact method for doing this needs to be researched further, but one could focus on expressing the "desire and expectation" policies of the SoS stakeholders directly in the modelling language notations. The tool would then mark simulated variations from these policies as emergent behaviour of the SoS. The Smart Energy Grids, described in Section 4.2, would be a good initial starting point for such research as these systems aim for optimal energy pricing and energy use already is policy based.

Given the many relations and interactions that occur between the constituent systems in an SoS, a simulation must be able to encompass the challenges found in *Distributed* systems, such as concurrency, consistency, and communication faults. Identifying communication faults as well as newly initiated communication is necessary if simulators are to be used for detecting aspects of *Interdependency* during simulation.

Because of the constant *Evolution* of an SoS, SoS simulation tool should enable extensions of the simulator to be made. For instance, by enabling simulators to handle extensions of the modelling language itself, or by enabling communication external to the simulation environment, such as through remote procedure calls [Nielsen et al. 2012].

An SoS simulator should be capable of capturing the *Dynamic Reconfiguration* that occurs as a result of the dynamically changing system topology, such that the dynamic changes can be communicated to stakeholders. There already exist tools that have a semantically well-founded foundation and are capable of performing simulation of distributed systems. Some of these allow for the inclusion of independent execution platform characteristics, with a focus on fault and data sharing. Fewer include

the *Dynamic Reconfiguration* dimension, for which the altering system topologies entail interoperability challenges, as well as overview challenges. A powerful simulator should deliver mechanisms for keeping track of changes; for instance, by visually indicating the current topology. In the same way, a strong SoS simulation tool needs to have a focus on the interaction between the constituents. A simulation should show the *Interdependence* between constituents, dataflow, and failed links between incompatible constituents.

Finally, as an SoS will consist of systems that have different data formats, architectures, and hardware, being able to include *Interoperability* aspects in the simulation is a particularly interesting research topic. One such example could be to conduct joint simulations with heterogeneous models that use different notations. Having this ability could aid the construction of an SoS by allowing the integration of various heterogeneous models defined for different constituent systems that are to be joint in the SoS.

6.4. Testing

The current challenges related to SoS testing can be classified according to the following aspects:

(1) Complexity issues: For very large SoS, such as the Emergency Management and Response example in Section 4.3, the size of the SoS state space will preclude, for example, exhaustive testing [Chow 1978; Springintveld et al. 2001], as well as search-based or exploratory testing [Spillner et al. 2006; Arcuri et al. 2010; Kaner 1999].
(2) Management issues: The multistakeholder situation that often is encountered in SoS development, verification, and validation (V&V), such as the e-commerce example given in Section 4.4, complicates capture of test cases for emergent SoS properties and the control of system integration test activities [Sledge 2006; Colombi et al. 2008].
(3) Applicability of multiple standards: Different standards impose different V&V and tool qualification requirements [Brauer et al. 2012]. This will be seen in SoS that are composed of independently owned and operated systems that are geographically distributed, such as the Transport networks described in Section 4.1.
(4) Dynamic evolution of SoS configurations: complicates test automation techniques and requires acceptance testing at runtime [Gonzalez et al. 2008].

Gonzalez et al. [2008] point out that *Dynamic Reconfiguration* is a crucial aspect of SoS behaviour. From testing and model checking object-oriented systems, it is well known that this conceptually unbounded state space of dynamically generated objects represents additional challenges. It has to be determined which numbers of objects considered in test configurations are meaningful to ensure that the SoS behaviour will be appropriate in *any* configuration that may occur.

In order to mitigate the risks associated with the challenges listed previously, the guideline OUSD(AT&L), DoD [2008] recommends a model-based approach to SoS development and V&V, since semantically well-defined models present a clearer view on system capabilities than informal descriptions, and form the basis for automated development and V&V activities. This view is backed up by various standards applicable to complex, typically safety-critical systems: (1) The IEC standard [IEC61508-3 2010] classifies Model-Based Testing (MBT) as a *highly recommended* method for testing software and systems of the highest safety integrity levels SIL-3 and SIL-4. (2) The avionic standard [RTCA SC-205/EUROCAE WG-71 2011b] devotes a complete supplement to model-based development and verification; the latter includes simulation and testing [RTCA SC-205/EUROCAE WG-71 2011a]. (3) The automotive standard

[International Organization for Standardization 2009] acknowledges the capabilities of model-based development and testing.

The benefits of MBT have been clearly identified [Baker et al. 2008], and case studies show the feasibility of the approach for industrial-size systems [Grieskamp 2010; Peleska et al. 2011]. A particular benefit of the MBT approach consists in the possibility to derive relevant test cases automatically from the model. Moreover, if model elements are already linked to requirements, as supported, for example, by the SysML modelling formalism [SysML1.2 2010], MBT can trace requirements to test cases and procedures in an automated way. The feasibility proof, however, currently applies to subsystems only: As of today, there exists no accepted methodology for combining constituent system test models into an SoS testing model. Model-based testing for small to medium size control systems may be fully automated, with potential additions of user-defined test cases that may be specified, for example, already as parts of the test model [Peleska et al. 2011].

Despite the commitment to MBT, which is visible in international standards, and despite the available evidence about MBT efficiency, the model-based testing approach cannot be regarded today as an industrial best practice of SoS V&V. It is more the case that companies are currently evaluating the potential benefits of the approach. The change of paradigm required for introducing MBT is enforced rather slowly by the responsible management, since it may also require a change of skills in the testing teams involved: The focus of testing activities is shifted from test procedure programming to model development. As a consequence, teaching model-based testing methodology and adapting V&V process models to incorporate the MBT approach is a major challenge for SoS testing. Apart from these educational and managerial challenges, there remain some technical ones to be overcome for applying MBT successfully in SoS testing campaigns. Typical success stories from the MBT application area emphasise the importance of the availability of a complete test model [Löding and Peleska 2010] as the starting point of the test automation tool chain. This may be a major obstacle: partly due to the number of cooperating constituent systems for which submodels have to contribute to the SoS test model, and partly due to the complete SoS test model only being available at the later stages of the development lifecycle. As a consequence, the benefits of automation are in danger of being nullified by the disadvantage of delayed start of SoS system-level testing. We conclude that for these reasons it is mandatory to support incremental development cycles for test models and test objectives, such that each cycle may be concluded with a test generation and execution campaign.

The need for incremental and even partially automated test model development has gained attention in the research communities, and Vaandrager [2012] describes how machine learning techniques can be applied to automatically construct test models by incrementally deriving state machine models from test execution traces observed. This approach is obviously attractive for MBT, since testing already starts while the model is constructed.

Other research areas, however, emerge from the investigation of model-based SoS testing and its specific needs. Extending the list of research foci shown at the beginning of Section 6.4, we anticipate in-depth research on the following:

(5) SoS-specific formalisms for model-based testing: By abstracting constituent system behaviour with the help of contracts, the complexity of SoS-level test models can be considerably reduced [Coleman et al. 2012; Milius et al. 2011]. Relating requirements to model elements—as, for example, provided by SysML [Holt and Perry 2008]—enables automated tracing between requirements, test cases, procedures, and results [Hallerstede et al. 2012].

(6) Identification of test objectives for emergent properties: On SoS level, test objectives can be identified by global mission threads or scenarios, and different constituent system behaviours can be abstracted in equivalence classes [Colombi et al. 2008; Kaner 2003].
(7) Methodology for incremental model-based SoS testing.

6.5. Verification

To realise the full potential of formal methods they must be integrated with other techniques used in the development of SoSs. Verification technology complements simulation and testing by providing exhaustive coverage; it can support requirements analysis, design, testing, and code generation. Work with formal verification has been performed in diverse areas, such as hardware platforms, concurrency, communication protocols, and real-time systems. More research is needed on how the integration of verification techniques into development processes would strengthen their use in an SoS context. In order to advance the SoS engineering discipline by enabling proofs of correctness for SoS, there is a need for both new and extended theories and tools.

It is worth noting that verification assumes that the specification of the system of interest is well understood and is clearly or formally articulated, so as to form a basis for verification judgements. SoSs often arise because reliance comes to be placed on the collective functioning of the constituent systems, and they evolve during operation as constituents change. The derivation and maintenance of SoS specifications that can serve as a basis for verification is thus a significant research challenge in itself, in particular to identify and avoid unwanted emergence.

Aside from the identification and maintenance of specification, other significant challenges for SoS verification include extending the range of techniques available, linking them together, and finding ways to apply them effectively in large. Creating new theories, and providing extensions to existing theories in order to enable SoS verification, would provide a very strong advance in SoS engineering. Not only will it enable the proof of correctness for the critical system embodied by many SoSs, it will also lead to a more systematic engineering process.

An SoS experiences a constant *Evolution*, the self-contained constituents are *Independent* and *Autonomous*, and there is a constant dynamic change of its configuration, state, and operations to respond to spatial and temporal changes. Providing verification support for all of these SoS characteristics is very challenging, as they can seem boundless and indeterminate. Such change will, for instance, occur often in e-commerce, given as an example in Section 4.4, because of the constant development these systems face; for instance, in terms of new suppliers or the addition of new platforms on which the virtual market place need to run.

The large degree of change and continuous development of the SoS topology and behaviour calls for new and extended theories and tools for verifying automatic, adaptive, and self-awareness properties. These include modelling systems that are state rich, concurrent, distributed, autonomous, time sensitive, mobile, hybrid, discrete and continuous, and probabilistic. Each of these areas has separate notations and modelling techniques, and it is a challenge to unify them. The objective is to have a collection of links that allow different aspects of SoSs and their component systems to be modelled and therefore verified in the most suitable theoretical setting. Unification of these modelling techniques would provide a coherent setting for understanding the SoS as a whole. Unification of modelling techniques would address *Independence*, *Autonomy*, and *Distribution*.

The key to scaling verification technology to realistic industrial SoSs is to model and analyse them at different levels of abstraction and to use techniques that compose cleanly. The link between different levels of abstraction is the use of refinement

techniques, and research is needed to develop suitable theories for the unified view discussed previously. A significant challenge is to discover techniques that allow us to divide and conquer the verification of large SoSs. As described with the Emergency Management and Response example in Section 4.3, the special and uncommon circumstances of a major emergency will involve a very large number of systems, which again will lead to very large models. A compositional verification technique allows us to forget the rest of the SoS when verifying a particular property or component system, and yet know that any results we obtain can be safely extrapolated to the SoS as a whole. Introducing techniques for abstraction and compositionality would help with questions of scale and *Dynamic Reconfiguration* and *Emergence of Behaviour*.

In the same way, the challenges of *Interoperability* and *Interdependence* require a strengthening of the theories of compositionality, information flow, and confidentiality as well as techniques for verifying compliance. The most challenging SoS characteristic for verification is the aspect of *Emergence of Behaviour*. There is a strong need for continued research into the process of engineering emergent behaviour in general; research, which can then be used to establish support for the verification of such a process.

It is clear that different tools are required for analysing different properties, such as response time, data integrity, and mutual exclusion of concurrent access, and these too must be integrated. Each of these properties is best checked with a different tool and an integrated tool set is required. This integration could be quite tight with tools cooperating in verification tasks. For example, a model checker could subcontract a theorem prover to establish some of its results. It is a major theoretical and engineering challenge to integrate tools in this way.

Placing a sharp research focus on the development of strong tool support is important, given that the engineers who work with SoS development come from very diverse fields, and as such the expertise with verification techniques may be limited. Having powerful and easy to use tools, for instance with automated proof-checking support, will not only provide stronger results, it will also ease the reception and approval of the tool amongst its users.

7. CONCLUDING REMARKS

Our objective in this article was to review the state of the art in model-based approaches to the engineering of SoSs. Faced with a large body of literature in what is still a varied and lively research field within systems engineering, we analysed a wide range of sources offering definitions or taxonomic bases of SoS concepts and identified eight "dimensions" that might allow one to place examples within the SoS space. We think that the unification conducted in deriving these dimensions will be useful in the future for classifying the focus of different SoS applications. We went on to examine model-based techniques for system description, simulation, testing, and verification, relating these to the eight dimensions identified. We hope that these directions of future challenges towards a strengthened discipline of SoS engineering will be taken up by more researchers.

It should be noted that our survey did not address human or stochastic aspects of SoS description, and we recognise that this means our story is necessarily partial. However, it does reflect a relative paucity of research literature on "humans in the SoS."

The review identifies a need for a growing body of case studies of SoS engineering on which research can draw, either to establish and test new methods and theories or to verify existing ones. Getting access to various types of SoS can, however, prove to be difficult. Much of the work on SoS engineering to date originates in military and aerospace applications, and these typically have a degree of confidentiality that makes it difficult to gain access. Having multiple owners and stakeholders of constituent

systems make data collection and testing difficult. Even for nonmilitary SoS the sheer size and geographical location of constituent systems and stakeholders can make it difficult to gain access and overview for logistical reasons. As the SoS field is still in its infancy, there is a need for case studies that show examples of SoS and show the effect that the SoS dimensions have on the engineering effort. A series of challenging examples, proposed by practitioners and placed in the public domain can galvanise competitive research, as illustrated in some measure by the Verified Software Initiative [Hoare et al. 2009]. This may also involve identifying systems that have not yet been seen as SoS, but may express many of the SoS dimensions.

Although some large-scale SoSs are engineered using conventional techniques, having evolved over a long period, and may not be identified explicitly as SoSs, the opportunities afforded by the convergence of embedded computing platforms and the Internet mean that there may be many more SoSs engineered, and explicitly identified as such, in the future.

A. OVERVIEW OF STATE OF PRACTICE

An overview of the current state of practice within SoS engineering and development is supplied in Table II. The table lists the four focus areas with the eight dimensions derived in Section 3 and for each combination the current state of practice is assessed based on the literature surveyed in the current section.

Table II. Overview of State of Practice

Characteristics	Modelling/Arch.	Simulation	Verification	Testing
Autonomy	Agent-based	Agent-based	Model checking	Independent DT&E, OT&E
Independence	Independent constituent system model	Using stubs	Compositionality	Independent DT&E, OT&E
Distribution	HLA and DEVS	Simulators combined using HLA	Theorem proving, model checking	Conformance testing, Interoperability testing
Evolution	Software engineering principles	No best practices available	Theorem proving, model checking	Repeated interoperability testing
Dynamic Reconfiguration	Domain-specific languages	Domain-specific languages	Theorem proving, model checking	No best practices available
Emergence of Behaviour	No best practices available	No best practices available	Refinement	End-to-end testing
Interdependence	Interface-level	Interface-level	Compositionality	End-to-end testing
Interoperability	HLA and architectural frameworks	HLA and DEVS	Compositionality	Conformance testing and interoperability testing strategies

B. OVERVIEW OF CHALLENGES IN SOS ENGINEERING

An overview of the research challenges in model-based techniques for SoS engineering is supplied in Table III. The table lists the four focus areas with the eight dimensions derived in Section 3 and for each combination the key research challenge is assessed based on the literature surveyed in the current section.

Table III. Overview of Challenges in SoS Engineering

Characteristics	Modelling/Arch.	Simulation	Test	Verification
Autonomy	Higher abstraction level	Human behaviour and constituents' interactions	No SoS-specific challenges	Unification of modelling techniques
Independence	Sharing and trust	Combining models and distributed simulation	No SoS-specific challenges	Unification of modelling techniques
Distribution	Describing independent constituent platforms	Concurrency, consistency, and communication faults	Complexity	Unification of modelling techniques
Evolution	Ease of evolving models	Handling evolving semantics	Incremental test modelling	Evolving verification arguments
Dynamic Reconfiguration	Ability to express dynamic evolution	Communicating dynamic scenarios	Dynamic object management	Techniques for abstraction and compositionality
Emergence of Behaviour	Ability to express desires	Detecting emergent properties appearing	Specification of SoS test objectives	Techniques for abstraction and compositionality
Interdependence	Ability to express rely/ guarantee policies	Detecting violations during simulation	Complexity, specification of SoS test objectives	Unification of verification technologies
Interoperability	Well-founded interfaces and desired policies	Including aspects as data formats, architectures, and hardware	Complexity, specification of SoS test objectives	Unification of verification technologies

C. OVERVIEW OF A STRENGTHENED SOS ENGINEERING DISCIPLINE

This section provides a summation of the areas for future research and improved development methodologies, in order to establish a firmer foundation for model-based SoS engineering. An overview of each discussed area is supplied in Table IV. The table lists the four focus areas with the eight dimensions derived in Section 3 and for each combination the areas of research interest are summarized.

Table IV. Overview of a Strengthened SoS Engineering Discipline

Characteristics	Modelling/Arch.	Simulation	Test	Verification
Autonomy	Expressing capabilities and behaviour formally	Human-in-the-loop simulation	"local" MBT	Verification of self-awareness properties
Independence	Divided models and well-founded sharing rules	Individual and joint model simulation	"local" MBT	Theories for modelling unification and confidentiality
Distribution	Modelling of consistency and communication faults	Simulating platform characteristics, consistency and faults	Contract-based test models	Extending existing verification of concurrency and distribution
Evolution	Well-founded language extensibility	Simulator extensibility for language extensions	Incremental test modelling	Theories for system composition
Dynamic Reconfiguration	Expressing dynamically changing infrastructures and error handling	Easy communication to nontechnical stakeholders	Dynamic object creation/deletion in test models	Verification of adaptive properties and system composition
Emergence of Behaviour	Expressing desirable and undesirable behaviours	Identification of emergence on the basis of policies	Deriving verification theories from more universal research in emergent behaviours	Contract-based test models, test objectives for emergent properties
Interdependence	Well-founded policy for desired properties	Detecting dataflow and connection policy violations	Test objectives for emergent properties	Techniques for verifying compliance and information flow
Interoperability	Well-founded and detailed constituent interfaces	Cosimulation to enable heterogeneous simulation	Test objectives for emergent properties	Unification of verification methods for data integrity and composi-tionality

D. OVERVIEW OF SOS DEFINITIONS

Table V. Historical Overview of Publications Defining SoS

Author(s)	Main characteristics	Section
Boulding [1956]	Static structure of "open systems" from different disciplines	2.1
Ackoff [1971]	System science organising systems into structured framework	2.1
Eisner et al. [1991]	Metasystems engineering framework combining independent systems	2.5
Shenhar et al. [1994]	Taxonomy with technological uncertainty and scope	2.3
Noam [1994]	Telecommunication infrastructure moving from "network of networks" to an SoS	2.4
Manthorpe [1996]	Focus on the jointness between C4I in a defence setting	2.4
Kotov [1997]	Large-scale concurrent and distributed systems	2.4
Lukasik [1998]	The importance of educating of engineers to deal with evolving self-organizing systems	2.5
Maier [1998a]	Most influential paper defining the OMGEE characteristics of SoS	2.2
Roe [1999]	Systems engineering process for military SoS	2.5
Cook et al. [1999]	SoS as a systems methodology for military systems with concerns for hierarchy, emergence, and C2	2.7
Krygiel [1999]	Focus on interoperability of information and data sharing	2.7
Pei [2000]	SoS as a defining factor in future battlefield scenarios	2.4
Carlock and Fenton [2001]	Enterprise Systems Engineering point of view	2.4
Sage and Cuppan [2001]	Use of the strategy "new federalism" for organisational structuring	2.7
Chen and Clothier [2003]	Focus on the SoS environment with a core in architecture interoperability and dynamic behaviour	2.5
Keating et al. [2003]	SoS as a metasystem of interrelated complex subsystems	2.5
Bar-Yam et al. [2004]	Derives SoS characteristic from the fields military, biology, and sociology	2.7
Crossley [2004]	SoS as a multidisciplinary research in interoperability, individual behaviour, and human behaviour	2.5
DeLaurentis and Crossley [2005]	Three dimension taxonomy for SoS analysis and design	2.3
Abbott [2006]	Open at the top, open at the bottom, and continually evolving, but slowly	2.2
Boardman and Sauser [2006]	The alphabet characteristics: autonomy, belonging, connectivity, diversity, and emerging	2.2
Boehm [2006]	Software-Intensive SoSs	2.4
Cocks [2006]	An SoS may not be as much about the system mission, but about the architecture of the selected solution	2.4
Fisher [2006]	Composition of autonomous systems is an SoS	2.7
Sharawi et al. [2006]	Independence, interoperability, and global goal are essential SoS concepts for modelling and simulation	2.7
OUSD(AT&L), DoD [2008]	SoS is an arrangement of independent and useful systems that integrate to deliver unique capabilities	2.6
Karcanias and Hessami [2010]	Evolution of the notion of composite systems to include automony and independence	2.3
Cantot and Luzeaux [2011]	Independently acquired systems operated in order to maximise the performance of the global operation	2.5
INCOSE [2015]	Multiple heterogeneous distributed systems addressing large-scale and interdisciplinary problems; entail ambiguous requirements, unclear boundaries, and interfaces	2.6

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers, to Padraig Lyons, and to many members of the COMPASS project team for valuable discussions during the preparation of this article.

REFERENCES

Russ Abbott. 2006. Open at the top; open at the bottom; and continually (but slowly) evolving. In *2006 IEEE/SMC International Conference on System of Systems Engineering*. IEEE.

Russell L. Ackoff. 1971. Towards a system of systems concept. *Management Science* 17, 11 (July 1971), 661–671.

Datu Buyung Agusdinata and Daniel DeLaurentis. 2008. Specification of system-of-systems for policymaking in the energy sector. *Integrated Assessment Journal* 8, 2 (2008), 1–24.

Andrea Arcuri, Muhammad Zohaib Iqbal, and Lionel Briand. 2010. Black-box system testing of real-time embedded systems using random and search-based testing. In *22nd IFIP International Conference on Testing Software and Systems (ICTSS'10)*. Springer-Verlag, 95–110.

Robert Axelrod. 1997. *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press.

J. Bach and P. Schroeder. 2004. Pairwise testing—A best practice that isn't. In *22nd Pacific Northwest Software Quality Conference*. 180–196.

Paul Baker, Zhen Ru Dai, Jens Grabowski, Øystein Haugen, Ina Schieferdecker, and Clay Williams. 2008. *Model Driven Testing—Using the UML Testing Profile*. Springer.

W. C. Baldwin and B. Sauser. 2009. Modeling the characteristics of system of systems. In *IEEE International Conference on System of Systems Engineering (SoSE'09)*. IEEE.

J. W. Ball, Sr. 1997. Free flight an air traffic management partnership. In *16th Digital Avionics Systems Conference, 1997, AIAA/IEEE*, Vol. 2. IEEE, 9–1.

Yaneer Bar-Yam, Mary Ann Allison, Ron Batdorf, Hao Chen, Hoa Generazio, Harcharanjit Singh, and Steve Tucker. 2004. The characteristics and emerging behaviors of system of systems. *NECSI: Complex Physical, Biological and Social Systems Project*, 1–16.

Mark A. Bedau. 1997. Weak emergence. In *Philosophical Perspectives: Mind, Causation, and World*. Vol. 11. Blackwell, 375–399.

H. Berenji and M. Jamshidi. 2011. Fuzzy reinforcement learning for system of systems (SoS). In *2011 IEEE International Conference on Fuzzy Systems (FUZZ)*. 1689–1694.

Brian J. L. Berry. 1964. Cites as systems within system of cites. *Papers and Proceedings of the Regional Science Association* 13, 1 (Jan. 1964), 149–163.

Antoine Beugnard, Jean-Marc Jezequel, Noel Plouzeau, and Damien Watkins. 1999. Making components contract aware. *IEEE Computer* (July 1999), 38–45.

Robin Bloomfield and Ilir Gashi. 2008. *Evaluating the Resilience and Security of Boundaryless, Evolving Socio-Technical Systems of Systems*. Technical Report. Centre for Software Reliability, City University.

John Boardman and Brian Sauser. 2006. System of systems—The meaning of "of." In *2006 IEEE/SMC International Conference on System of Systems Engineering*. IEEE.

Barry Boehm. 2006. A view of 20th and 21st century software engineering. In *28th International Conference on Software Engineering (ICSE'06)*. ACM, 12–29.

Kenneth E. Boulding. 1956. General systems theory—The skeleton of science. *Management Science* 2, 3 (April 1956).

Jörg Brauer, Jan Peleska, and Uwe Schulze. 2012. Efficient and trustworthy tool qualification for model-based testing tools. In *Testing Software and Systems (LNCS)*. Springer, Berlin, 8–23. DOI:http://dx.doi.org/10.1007/978-3-642-34691-0_3

J. Bryans, J. Fitzgerald, R. Payne, and K. Kristensen. 2014. Maintaining emergence in systems of systems integration: A contractual approach using SysML. In *INCOSE International Symposium on System Engineeering*. INCOSE.

Dale S. Caffall and James Bret Michael. 2004. A new paradigm for requirements specification and analysis of system-of-systems. In *Radical Innovations of Software and Systems Engineering in the Future, 9th International Workshop, RISSEF 2002 (LNCS)*, Vol. 2941. Springer, 108–121.

Radu Calinescu and Marta Z. Kwiatkowska. 2009. Using quantitative analysis to implement autonomic IT systems. In *31st International Conference on Software Engineering (ICSE'09)*. 100–110.

Radu Calinescu and Marta Z. Kwiatkowska. 2010. Software engineering techniques for the development of Systems of Systems. In *Foundations of Computer Software. Future Trends and Techniques for Development, 15th Monterey Workshop 2008 (LNCS)*, Vol. 6028. Springer, 59–82.

James E. Campbell, Dennis E. Longsine, Donald Shirah, and Dennis J. Anderson. 2005. *System of Systems Modeling and Analysis*. Technical Report SAND2005-0020. Sandia National Laboratories.

Pascal Cantot and Dominique Luzeaux. 2009. *Simulation et Modélisation des Systèmes de Systèmes: Vers la Maîtrise de la Complexité*. Hermes Science Publications.

Pascal Cantot and Dominique Luzeaux. 2011. *Simulation and Modeling of Systems of Systems*. Wiley.

Paul G. Carlock and Robert E. Fenton. 2001. System of systems (SoS) enterprise systems engineering for information-intensive organizations. *Systems Engineering* 4, 4 (2001), 242–261.

David Carney, David Fasher, and Patrick Place. 2005. *Topics in Interoperability: System-of-Systems Evolution*. Technical Report CMU/SEI-2005-TN-002. Software Engineering Institute, Carnegie Mellon University.

Peter Checkland. 1999. *Systems Thinking, Systems Practice: Includes a 30-Year Retrospective*. Wiley.

Pin Chen and Jennie Clothier. 2003. Advancing systems engineering for systems-of-systems challenges. *Systems Engineering* 6, 3 (May 2003), 170–183.

Tsun S. Chow. 1978. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering* SE-4, 3 (March 1978), 178–186.

H. Q. Nguyen, C. Kaner, and J. Falk. 1999. *Testing Computer Software*. Wiley.

Dan Cocks. 2006. How should we use the term "System of Systems" and why should we care? In *16th INCOSE International Symposium 2006*. INCOSE.

Joey W. Coleman, Anders Kaels Malmos, Peter Gorm Larsen, Jan Peleska, Ralph Hains, Zoe Andrews, Richard Payne, Simon Foster, Alvaro Miyazawa, Cristiano Bertolini, and André Didier. 2012. COMPASS tool vision for a system of systems collaborative development environment. In *7th International Conference on System of System Engineering (SoSE'12)*. 451–456.

John Colombi, Brannen C. Cohee, and Chuck W. Turner. 2008. Interoperability test and evaluation: A systems of systems field study. *The Journal of Defense Software Engineering* (Nov. 2008), 10–14.

S. Cook, E. Lawson, and J. Allison. 1999. Towards a unified systems methodology for Australian defence systems-of-systems. *Proceedings of the 9th INCOSE International Symposium* (June 1999).

S. C. Cook. 2001. On the acquisition of systems of systems. *2001 INCOSE International Symposium* (July 2001).

Luís Diogo Couto and Richard Payne. 2013. The COMPASS proof obligation generator: A test case of Overture extensibility. In *11th Overture Workshop*.

W. A. Crossley. 2004. System of systems: An introduction of Purdue University schools of Engineering's Signature Area. In *Engineering Systems Symposium*. MIT Engineering Systems Division.

J. Dahmann. 2014. Systems of systems pain points. In *INCOSE International Symposium on Systems Engineering 2014*.

Judith Dahmann and Kristen Baldwin. 2008. Understanding the current state of US defense systems of systems and the implications for systems engineering. In *IEEE Systems Conference*. IEEE.

Judith S. Dahmann, George Rebovich, Jr., and Jo Ann Lane. 2008. Systems engineering for capabilities. *CrossTalk Journal (The Journal of Defense Software Engineering)* 21, 11 (Nov. 2008), 4–9.

K. Daniel, B. Dusza, A. Lewandowski, and C. Wietfeld. 2009. AirShield: A system-of-systems MUAV remote sensing architecture for disaster response. In *2009 3rd Annual IEEE Systems Conference*. 196–200.

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. In *21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP'07)*. ACM, 205–220.

Daniel A. DeLaurentis. 2005. Understanding transportation as system-of-systems design problem. In *43rd AIAA Aerospace Sciences Meeting and Exhibit*. AIAA.

Daniel A. DeLaurentis and W. A. Crossley. 2005. A taxonomy-based perspective for systems of systems design methods. In *IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 86–91.

Department of Defense. 2004. *Defense Acquisition Guidebook*. https://dag.dau.mil/Pages/Default.aspx.

G. Despotou, R. Alexander, and M. Hall-May. 2003. *Key Concepts and Characteristics of Systems of Systems (SoS)*. DARP - HIRTS. University of York.

T. Dillon, V. Potdar, J. Singh, and A. Talevski. 2011. Cyber-physical systems: Providing quality of service (QoS) in a heterogeneous systems-of-systems environment. In *5th IEEE International Conference on Digital Ecosystems and Technologies (DEST'11)*. 330–335.

D. Drusinksy and Man-Tak Shing. 2005. Creation and evaluation of formal specifications for system-of-systems development. In *IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 1864–1869.

Doron Drusinsky. 2003. Monitoring temporal rules combined with time series. In *15th International Conference on Computer Aided Verification (CAV'03) (LNCS)*, Vol. 2725. Springer, 114–117.

John P. Eddy, Dennis J. Anderson, Craig R. Lawton, and James E. Campbell. 2007. *Network and Adaptive System of Systems Modeling and Analysis*. Technical Report SAND2007-2788. Sandia National Laboratories.

H. Eisner, J. Marciniak, and R. McMillan. 1991. Computer-aided system of systems (S2) engineering. In *IEEE International Conference on Systems, Man, and Cybernetics, 1991.* Vol. 1. 531–537.

European Commission. 2012. *Directions in Systems of Systems Engineering*. Technical Report. European Commission, Communications Networks, Content and Technology Directorate—General Unit A3-DG CONNECT.

David A. Fisher. 2006. *An Emergent Perspective on Interoperation in Systems of Systems*. Technical Report. CMU/SEI-2006-TR-003. Software Engineering Institute, Carnegie Mellon University.

John Fitzgerald, Peter Gorm Larsen, and Jim Woodcock. 2012. Modelling and analysis technology for systems of systems engineering: Research challenges. In *INCOSE*.

M. T. Gamble and R. F. Gamble. 2008. Reasoning about hybrid system of systems designs. In *7th International Conference on Composition-Based Software Systems (ICCBSS'08)*. 154–163.

N. D. Geddes, D. M. Smith, and C. S. Lizza. 1998. Fostering collaboration in systems of systems. In *1998 IEEE International Conference on Systems, Man, and Cybernetics, 1998*. IEEE, 950–954, Vol. 1.

Alberto Gonzalez, Eric Piel, Hans-Gerhard Gross, and Maurice Glandrup. 2008. Testing challenges of maritime safety and security systems-of-systems. In *Testing: Academic & Industrial Conference—Practice and Research Techniques*. IEEE Computer Society, 35–39.

A. Gorod, B. Sauser, and J. Boardman. 2008. System-of-systems engineering management: A review of modern history and a path forward. *IEEE Systems Journal* 2, 4 (Dec. 2008), 484–499.

Wolfgang Grieskamp. 2010. Microsoft's protocol documentation program: A success story for model-based testing. In *Testing—Practice and Research Techniques, 5th International Academic and Industrial Conference, TAIC PART 2010. (Lecture Notes in Computer Science)*, Vol. 6303, Leonardo Bottaci and Gordon Fraser (Eds.). Springer, 7. DOI:http://dx.doi.org/10.1007/978-3-642-15585-7_3

J. O. Gutierrez-Garcia, F. F. Ramos-Corchado, and J.-L. Koning. 2009. Obligations as constrainers, descriptors, and linkers of open system of systems. In *IEEE International Conference on System of Systems Engineering (SoSE'09)*. 1–6.

Stefan Hallerstede, Finn Overgaard Hansen, Jon Holt, Rasmus Lauritsen, Lasse Lorenzen, and Jan Peleska. 2012. Technical challenges of SoS requirements engineering. In *7th International Conference on System of System Engineering (SoSE'12*). IEEE, 573–578.

D. Harel. 1987. StateCharts: A visual formalism for complex systems. *Science of Computer Programming* 8, 3 (1987), 231–274.

K. Harmon. 2005. The "systems" nature of enterprise architecture. In *2005 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 78–85, Vol. 1.

Klaus Havelund and Grigore Rosu. 2001. Monitoring programs using rewriting. In *16th IEEE International Conference on Automated Software Engineering (ASE'01)*. 135–143.

Derek K. Hitchins. 2005. Systems methodology. In *Conference on Systems Engineering Research*.

C. A. R. Hoare. 1978. Communicating sequential processes. *Communations of the ACM* 21, 8 (Aug. 1978).

C. A. R. Hoare, Jayadev Misra, Gary T. Leavens, and Natarajan Shankar. 2009. The verified software initiative: A manifesto. *ACM Computing Surveys* 41, 4, Article 22 (Oct. 2009), 8 pages.

Jon Holt. 2012. Model-based requirements engineering for system of systems. In *IEEE 7th International Conference on System of System Engineering (SoSE'12*). IEEE.

J. Holt and S. Perry. 2008. *SysML for Systems Engineering*. IET.

IEC61508-3. 2010. *IEC61508-3: Functional safety of electrical/electronic/programmable electronic safety-related systems—Part 3: Software Requirements*. International Electrotechnical Commission.

IEEE1516 2010. IEEE standard for modeling and simulation: High level architecture (HLA)—Framework and rules. *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)* (Aug. 2010), 1–38.

M. D. Ilić, Le Xie, U. A. Khan, and J. M. F. Moura. 2010. Modeling of future cyber-physical energy systems for distributed sensing and control. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 40, 4 (2010), 825–838.

INCOSE. 2011. *Systems Engineering Handbook. A Guide for System Life Cycle Processes and Activities, Version 3.2.2.* Technical Report INCOSE-TP-2003-002-03.2.2. International Council on Systems Engineering (INCOSE).

INCOSE. 2015. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, Version 3.2.2.* (4th ed.). Wiley.

C. Ingram, R. Payne, S. Perry, J. Holt, F. O. Hansen, and L. D. Couto. 2014. Modelling patterns for systems of systems architectures. In *International Systems Conference (SysCon'14)*. IEEE.

International Organization for Standardization. 2009. *ISO 26262—Road Vehicles—Functional Safety—Part 6: Product Development: Software Level*. ICS 43.040.10.

International Organization for Standardization. 2015. *ISO/IEC/IEEE 15288—Systems and Software Eningeering—System Life Cycle Processes*.

M. C. Jackson and P. Keys. 1984. Towards a system of systems methodologies. *The Journal of the Operational Research Society* 35, 6 (June 1984), 473–486.

Francçois Jacob. 1974. *The Logic of Living Systems: A History of Heredity*. Allen Lane.

M. Jamshidi. 2005. System-of-systems engineering—A definition. In *International Conference on Systems, Man, and Cybernetics*. IEEE.

M. Jamshidi. 2008. System of systems engineering—New challenges for the 21st century. *IEEE Aerospace and Electronic Systems Magazine* 23, 5 (May 2008), 4–19.

Xiong Jian, Ge Bing-feng, Zhang Xiao-ke, Yang Ke-wei, and Chen Ying-wu. 2010. Evaluation method of system-of-systems architecture using knowledge-based executable model. In *2010 International Conference on Management Science and Engineering (ICMSE)*. 141–147.

C. B. Jones. 1983a. Specification and design of (parallel) programs. In *IFIP'83*. IFIP, North-Holland, 321–332.

Cliff B. Jones. 1983b. Tentative steps toward a development method for interfering programs. *ACM Transactions on Programming Languages and Systems* 5, 4 (Oct. 1983), 596–619.

Cem Kaner. 2003. Cem Kaner on scenario testing: The power of "what if..." and nine ways to fuel your imagination. *Testing & Quality Engineering Magazine* (October 2003).

N. Karcanias and A. G. Hessami. 2010. Complexity and the notion of system of systems: Part (II): Defining the notion of system of systems. In *World Automation Congress (WAC), 2010*. 1–7.

C. B. Keating. 2005. Research foundations for system of systems engineering. In *2005 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2720–2725.

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

N. Kilicay-Ergin and C. Dagli. 2008. Executable modeling for system of systems architecting: An artificial life framework. In *2008 2nd Annual IEEE Systems Conference*. 1–5.

Vadim. Kotov. 1997. *Systems-of-Systems as Communicating Structures*. Technical Report HPL-97-124. Hewlett Packard Computer Systems Laboratory Paper.

Jeff Kramer. 2007. Is abstraction the key to computing? *Communications of the ACM* 50, 4 (2007), 37–42. DOI:http://dx.doi.org/10.1145/1232743.1232745

Anette J. Krygiel. 1999. *Behind the Wizard's Curtain, An Integration Environment for a System of Systems*. CCRP publication series.

Jo Ann Lane and Ricardo Valerdi. 2007. Synthesizing SoS concepts for use in cost modeling. *Systems Engineering* 10, 4 (Dec. 2007), 297–308.

Michael Lees, Brian Logan, and Georgios Theodoropoulos. 2007. Distributed simulation of agent-based systems with HLA. *ACM Transactions on Modeling and Computer Simulation* 17, 3 (July 2007).

Nancy Leveson. 1995. *SAFEWARE: System Safety and Computers*. Addison-Wesley.

Qianhui Liang and Stuart H. Rubin. 2009. Randomization for testing systems of systems. In *IEEE International Conference on Information Reuse & Integration (IRI'09)*. IEEE, 110–114. DOI:http://dx.doi.org/10.1109/IRI.2009.5211597

Shiyong Liu. 2011. Employing system of systems engineering in China's emergency management. *2010 4th Annual IEEE Systems Conference* 5, 2 (April 2011), 541–546.

R. Lock and I. Sommerville. 2010. Modelling and analysis of socio-technical system of systems. In *15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*. 224–232.

Helge Löding and Jan Peleska. 2010. Timed Moore automata: Test data generation and model checking. In *International Conference on Software Testing, Verification, and Validation (ICST'08)*. IEEE, 449–458. DOI:http://dx.doi.org/10.1109/ICST.2010.60

Carsten Lucke, Sascha Krell, and Ulrike Lechner. 2010. Critical issues in enterprise architecting—A literature review. In *AMCIS 2010 Proceedings*.

Marie Ludwig, Nicolas Farcet, Jean-Philippe Babau, and Joël Champeau. 2011. Integrating design and runtime variability support into a system ADL. In *7th European Conference on Modelling Foundations and Applications*. Springer-Verlag, 270–281.

Stephen J. Lukasik. 1998. Systems, systems of systems, and the education of engineers. In *AI EDAM*, Vol. 12. Cambridge University, 55–60.

V. Mahulkar, S. McKay, D. E. Adams, and A. R. Chaturvedi. 2009. System-of-systems modeling and simulation of a ship environment with wireless and intelligent maintenance technologies. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans,* 39, 6 (Nov. 2009), 1255–1270.

Mark W. Maier. 1996. Architecting principles for systems-of-systems. In *INCOSE 1996 6th Annual International Symposium of the International Council on Systems Engineering*. INCOSE.

Mark W. Maier. 1998a. Architecting principles for systems-of-systems. *Systems Engineering* 1, 4 (1998), 267–284.

Mark W. Maier. 1998b. Architecting principles for systems-of-systems. The Information Architects Cooperative (TIAC) whitepaper, www.infoed.com/Open/PAPERS/systems.htm.

Mark W. Maier. 2005. Research challenges for systems-of-systems. In *2005 IEEE International Conference onSystems, Man and Cybernetics*. IEEE.

Mark W. Maier and Eberhardt Rechtin. 1997. *The Art of Systems Architecting*. CRC Press LLC.

Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *FORMATS 2004 and FTRTFT 2004 (LNCS)*, Vol. 3253. Springer, 152–166.

Mo Mansouri, Alex Gorod, Thomas H. Wakeman, and Brian Sauser. 2009. Maritime transportation system of systems management framework: A system of systems engineering approach. *International Journal of Ocean Systems Management* 1, 2 (2009), 200–226.

William H. J. Manthorpe. 1996. The emerging joint system of systems: A systems engineering challenge and opportunity for APL. *John Hopkins APL Technical Digest* 17, 3 (1996), 305–310.

Fredrick Mauss, Juan Valencia, Brian Hatchell, Kurt Silvers, and Shannon Crowell. 2015. System of systems approaches for mobile source transit security. In *15th INCOSE International Symposium (IS'15)*. INCOSE.

Abe Meilich. 2006. System of systems (SoS) engineering & architecture challenges in a net centric environment. In *2006 IEEE/SMC International Conference on System of Systems Engineering*. IEEE.

Bertrand Meyer. 1992. Applying design by contract. *IEEE Computer* 25, 10 (1992), 40–51.

J. B. Michael, R. Riehle, and Man-Tak Shing. 2009. The verification and validation of software architecture for systems of systems. In *IEEE International Conference on System of Systems Engineering (SoSE'09)*. 1–6.

Stefan Milius, Jan Peleska, and Martin Sulzmann. 2011. *Contract Specification and Domain Specific Modeling Language for GALS Systems An Approach to System Validation*. Technical Report. Technische Universität Braunschweig.

M. Z. Miller, K. Griendling, and D. N. Mavris. 2012. Exploring human factors effects in the smart grid system of systems demand response. In *2012 7th International Conference on System of Systems Engineering*. IEEE, 1–6.

Saurabh Mittal, Bernard Zeigler, José Martin, Ferat Sahin, and Mo Jamshidi. 2009. *System of Systems—Innovations for the 21st Century*. Wiley. 101–149.

Heiner Müller-Merbach. 1994. A system of systems approaches. *Interfaces* 24, 4 (1994), 16–25.

Claus Ballegaard Nielsen and Peter Gorm Larsen. 2012. Extending VDM-RT to enable the formal modelling of system of systems. In *IEEE 7th International Conference on System of System Engineering (SoSE'12)*. IEEE.

Claus Ballegaard Nielsen, Kenneth Lausdahl, and Peter Gorm Larsen. 2012. Combining VDM with executable code. In *Abstract State Machines, Alloy, B, VDM, and Z (Lecture Notes in Computer Science)*, Vol. 7316, John Derrick, John Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves, and Elvinia Riccobene (Eds.). Springer-Verlag, Berlin, 266–279. http://dx.doi.org/10.1007/978-3-642-30885-7_19 ISBN 978-3-642-30884-0.

Eli M. Noam. 1994. Beyond liberalization: From the network of networks to the system of systems. *Telecommunications Policy* 18, 4 (May 1994), 286–294.

L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau. 2006. *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Technical Report. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

OUSD(AT&L), DoD. 2008. *Systems and Software Engineering. Systems Engineering Guide for Systems of Systems*. Technical Report Version 1.0. Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Department of Defense.

A. William A. Owens. 1995. *Dominant Battlespace Knowledge*. Institute for National Strategic Studies, The Emerging U.S. System-of-Systems.

Richard Payne, Jeremy Bryans, John Fitzgerald, and Steve Riddle. 2012. Interface specification for system-of-systems architectures. In *IEEE 7th International Conference on System of System Engineering, (SoSE'12)*. IEEE.

Richard J. Payne and John S. Fitzgerald. 2010. *Evaluation of Architectural Frameworks Supporting Contract-Based Specification*. Technical Report CS-TR-1233. School of Computing Science, Newcastle University.

R. S. Pei. 2000. Systems of systems integration (SoSI)-a smart way of acquiring army C4I2WS systems. In *Summer Computer Simulation Conference*.

Jan Peleska. 2013. Industrial-strength model-based testing—State of the art and current challenges. *Electronic Proceedings in Theoretical Computer Science* abs/1303.1006, 3–28.

Jan Peleska, Artur Honisch, Florian Lapschies, Helge Löding, Hermann Schmid, Peer Smuda, Elena Vorobev, and Cornelia Zahlten. 2011. A real-world benchmark model for testing concurrent real-time systems in the automotive domain. In *International Conference on Testing Software and Systems. (ICTSS'11) (LNCS)*, Vol. 7019. Springer, 146–161.

M. S. Phadke. 1989. *Quality Engineering Using Robust Design*. Prentice Hall.

Fiona Polack and Susan Stepney. 2005. Emergent properties do not refine. *Electronic Notes in Theoretical Computer Science* 137, 2 (2005), 163–181.

Fred R. Ricker and Ravi Kalakota. 1999. Order fulfillment: The hidden key to e-commerce success. *Supply Chain Management Review* 11, 3 (Fall 1999), 60–70.

J. Ring and A. M. Madni. 2005. Key challenges and opportunities in "system of systems" engineering. In *2005 IEEE International Conference on Systems, Man and Cybernetics*. 973–978.

A. W. Roscoe. 2010. *Understanding Concurrent Systems*. Springer.

RTCA SC-205/EUROCAE WG-71. 2011a. *Model-Based Development and Verification Supplement to DO-178C and DO-278A*. Number RTCA/DO-331. RTCA, Inc., 1140 Connecticut Avenue, N.W., Suite 1020, Washington, D.C. 20036.

RTCA SC-205/EUROCAE WG-71. 2011b. *Software Considerations in Airborne Systems and Equipment Certification*. Number RTCA/DO-178C. RTCA, Inc., 1140 Connecticut Avenue, N.W., Suite 1020, Washington, D.C. 20036.

Andrew P. Sage and Christopher D. Cuppan. 2001. On the systems engineering and management of systems of systems and federations of systems. *Information Knowledge Systems Management* 2, 4 (Dec. 2001), 325–345.

F. Sahin, M. Jamshidi, and P. Sridhar. 2007. A discrete event XML based simulation framework for system of systems architectures. In *IEEE International Conference on System of Systems Engineering (SoSE'07)*.

J. W. Sanders and Graeme Smith. 2012. Emergence and refinement. *Formal Aspects of Computing* 24, 1 (2012), 45–65.

Daniel Schneider and Mario Trapp. 2009. Runtime safety models in open systems of systems. *IEEE International Symposium on Dependable, Autonomic and Secure Computing*, 455–460.

Scott A. Selberg and Mark A. Austin. 2008. *Toward an Evolutionary System of Systems Architecture*. Technical Report. Institute for Systems Research. INCOSE.

Abeer Sharawi, Serge N. Sala-Diakanda, Sergio Quijada, Nabeel Yousef, Luis Rabelo, and Jose Sepulveda. 2006. A distributed simulation approach for modeling and analyzing system of systems. In *2006 Winter Simulation Conference*.

Aaron J. Shenhar. 1994. A new systems engineering taxonomy. In *4th Annual International Symposium of The National Council on Systems Engineering*, Vol. 2. 261–276.

Aaron J. Shenhar and Zeev Bonen. 1997. The new taxonomy of systems: Toward an adaptive systems engineering framework. *IEEE Transactions on Systems, Man and Cybernetics* 27, 2 (1997), 137–145.

Carol A. Sledge. 2006. *Army ASSIP Systems-of-Systems Test Metrics Task*. Technical Report. CMU/SEI-2006-SR-011. Software Engineering Institute, Carnegie Mellon University.

Ian Sommerville, Dave Cliff, Radu Calinescu, Justin Keen, Tim Kelly, Marta Kwiatkowska, John McDermid, and Richard Paige. 2012. Large-scale complex IT systems. *Communications of the ACM* 55, 7 (2012).

Andreas Spillner, Tilo Linz, and Hans Schaefer. 2006. *Software Testing Foundations*. dpunkt.verlag.

J. G. Springintveld, F. W. Vaandrager, and P. R. D'Argenio. 2001. Testing timed automata. *Theoretical Computer Science* 254, 1–2 (March 2001), 225–257.

SysML1.2 2010. *OMG Systems Modeling Language (OMG SysML™)*. Technical Report Version 1.2. SysML Modelling team. http://www.sysml.org/docs/specs/OMGSysML-v1.2-10-06-02.pdf.

Genichi Taguchi. 1987. *System of Experimental Design, Volume 1 & 2*. UNIPUB/Kraus Intl. Publications.

K. Tatsumi. 1987. Test case design support system. In *International Conference on Quality Control (ICQC)*. 615–620.

United States, Congress, Senate, Committee on Armed Services. 1988. *Restructuring of the Strategic Defense Initiative (SDI) Program: US Senate*. University of Michigan Library.

Frits Vaandrager. 2012. Active learning of extended finite state machines. In *24th IFIP International Conference on Testing Software and Systems (ICTSS'12) (LNCS)*. Springer, 5–7.

Ricardo Valerdi, Elliot Axelband, Barry Boehm Thomas Baehren, Dave Dorenbos, Scott Jackson, Azad Madni, Gerald Nadler, Paul Robitaille, and Stan Settles. 2008. A research agenda for systems of systems architecting. *International Journal of System of Systems Engineering* 1, 1/2 (2008), 171–188.

B. E. White and P. N. Jean. 2011. Case study in system of systems engineering: NASA's advanced communications technology satellite. In *2011 6th International Conference on System of Systems Engineering (SoSE)*. 237–244.

N. Wickramasinghe, S. Chalasani, R. V. Boppana, and A. M. Madni. 2007. Healthcare system of systems. In *IEEE International Conference on System of Systems Engineering (SoSE'07)*. 1–6.

Jim Woodcock and Ana Cavalcanti. 2002. The semantics of circus. In *2nd International Conference of B and Z Users on Formal Specification and Development in Z and B (ZB'02)*. Springer-Verlag, 184–203.

Jim Woodcock, Ana Cavalcanti, John Fitzgerald, Peter Larsen, A. Miyazawa, and S. Perry. 2012. Features of CML: A formal modelling language for systems of systems. In *7th International Conference on System of Systems Engineering (SoSE'12)*. IEEE.

Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal methods: Practice and experience. *ACM Computing Surveys* 41, 4 (Oct. 2009), 1–36. DOI:http://dx.doi.org/10.1145/1592434.1592436

Jim C. P. Woodcock and Ana L. C. Cavalcanti. 2001. A concurrent language for refinement. In *5th Irish Workshop in Formal Methods (IWFM'01) (BCS Electronic Workshops in Computing)*.

P. Zave. 1993. Feature interactions and formal specifications in telecommunications. *Computer* 26, 8 (1993), 20–28. DOI:http://dx.doi.org/10.1109/2.223539

Received July 2013; revised January 2015; accepted June 2015