A Summary on Systems of Systems Engineering[1]

Johan Lukkien

May 2015

Introduction

'Systems-of-systems' (SoS) is a relatively modern term for systems that are composed of independent (autonomous) subsystems that are full-blown systems by themselves in every way. While there is growing awareness of its importance there is no clear agreement about the architectural principles guiding the design of SoS nor about the process of engineering them.

Thinking in terms of SoS brings a 180° change in viewpoint. In traditional embedded systems design (being a subsystem in itself) we focus on how to effectively design those embedded subsystems to implement the functions of a device.This device plays a role in the physical world. With the introduction of embedded networking we move to cyber-physical systems in which these embedded subsystems now form the nodes in a larger whole, also crossing the borderlines of different systems. From this perspective we regard the larger ICT context as the enabler of the involved embedded subsystems. SoS is about how to design, engineer, maintain and evolve a composition of subsystems while acknowledging the fact that these subsystems remain independent, serving their own functions, with their own management and lifecycles.

In this section we discuss what SoS really is about based on a review of existing literature on the topic. We examine the state of the work in this area, focusing on the following aspects:

a) What are engineering methodologies, processes and lifecycle management defined for SoS engineering?
b) What are architecting principles and perhaps styles suggested for SoS?
c) Which existing examples of SoS, that have benefited from recent insight, can we learn from?
d) Which concrete solutions exist, and what are open issues?

As above we speak about 'subsystems' in order to discriminate between the parts. Subsystems, however, are complete systems by themselves as explained above.

Systems-of-Systems

The work of Maier [Maier 1998] is one of the earlier systematic discussions on SoS. Maier introduces some defining properties intrinsic to an SoS: *operational independence* (subsystems have an autonomous behavior, goal and useful existence), *managerial independence* (subsystems are managed by different authorities) and *evolutionary independence* (subsystems evolve independently). In addition,

[1] This document was the basis for a section on State-of-the-Art in the Artemis Accus project.

geographic distribution is often a characteristic as wel as exibiting *emergent behavior*. According to Fisher [Fisher 2006], geographic distribution *supports* the three independency properties (while not being a necessary condition), and emergent behavior is the *result* of the subsystems having the three independency properties. DeLaurentis [DeLaurentis 2005] adds to the characteristics *heterogeneity* (of subsystems), *networks* (as the predominant means of connecting subsystems) and *trans-domain* collaboration (the need for differrent disciplines to collaborate, i.e., engineering, economy, policy makers etc.). Examples of SoS are typically indicated in military, transportation and avionic systems.

SoS is clearly about the design and engineering problems of combining existing systems into a larger whole that yields new functionality, not available through any of the constituent systems. These problems comprise the architectural principles of such combination, the engineering process and the technical solutions. We examine some concepts that refer to this combining of (sub)systems to understand the difference between *monolithic systems* (ML) and SoS.

System Integration

System Integration in the context of ML refers to the concept of synthesizing (independently developed) subsystems. Typically, the specification of these subsystems follows from a decomposition of an original design of an ML, which is a well-established engineering practice. The focus lies here on interface definition, on integration methodology (e.g. horizontal and vertical integration), on managing and reducing dependencies (coupling) and maximizing cohesion. The goal is to obtain a single integrated system in which the subsystems are there for serving the combined goal.

Within SoS there is also a need to integrate subsystems but there are important differences. Fisher [Fisher 2006] describes some implicit assumptions of system integration that are not true for SoS, viz. that the architecture is frozen in an earlier stage of the design, that the control flow and data flow is known, and that requirements as well as properties of subsystems are known upon integration. For SoS, the requirements for subsystems are not specified in a hierarchical manner; the function of each subsystem is defined by its original purpose as well as its own internal context, data, processes, etc. We call this aspect *uncorrelated requirements.* Second, in case of SoS there are two types of control flow: the control flow for the original purpose and the control flow coming from the SoS. We call this aspect *competition of control.* The result is that the subsystems retain their own functionality as well as independent existence (mentioned as operational and managerial independence before). Third, within SoS there is no a priori architectural principle that guides the design of the subsystems. We call this *architectural diversity.*

Interoperation

Interoperation refers to cooperative interactions between two or more partners to achieve objectives. These objectives can be shared (e.g., manage the traffic in an area), but can also be private to each partner and can be as simple as obeying actively some policy. E.g., in a client-server interaction, the server achieves its goal by serving clients.

Premise to interoperability is the ability to communicate which in networks is addressed until the transport layer in the OSI stack. On top of that meaningful information is exchanged and interpreted between interfaces. Interoperation requires a certain trust between cooperating partners. In addition an understanding is required on three aspects (which can be seen as stages in the interoperation): first, on *how to reach the interface* (discovering the interface), second on *how to perform the interoperation* (understanding the interface), and third, on *how the interaction contributes to the objectives* (understanding the semantics).

Aspects of interoperation are defined in several domains with slight differences, but always following this main line of reasoning. Within ML these three aspects can be entirely contained within the design, e.g. through embedding of implicit or explicit assumptions or through protocol standardization. More recent works on component based systems and distributed systems introduce concepts that allow late binding like service discovery, service descriptions (e.g. within UPnP [UPnP Forum 2008]) and the Service Oriented Architectural Style [Erl 2005]. Standards in semantic-level descriptions are RDF and OWL.

Interoperability within SoS needs to be based on a high-level description of goals and of services since the architectural diversity implies that no knowledge is available about the inner workings of subsystems. This is called *semantic interoperability.* This means that such late binding techniques needs to be further investigated and developed for SoS. The separation between service and implementation needs to be emphasized even further, in particular using *rich service interfaces* that include *semantic descriptions* and that *expose extra-functional information* as well (see below).

Emergence

Emergent properties refer to properties exhibited by the system as a whole that cannot be attributed to any of its subsystems in isolation [Steels 1991]. Examples are extra-functional properties like latency and throughput as well as security, reliability and availability, which typically arise from system behavior over time. Hence, also the term *emergent behavior* is often used. While in monolithic systems emergent properties are typically addressed within the architecture giving them a place in the process of hierarchical decomposition, within SoS these properties require explicit attention at subsystem boundaries. In line with the discussion on interoperability, these properties must be managed at subsystem interfaces.

Because of the properties of SoS there is an intrinsic uncertainty about the effect of operations, about failures etc. This must be taken into account at subsystem boundaries, in particular, by adopting failures, unpredictable behavior and conflicts of control as the natural mode of operation rather than as the exception.

For SoS *emergent functionality* is also identified as a defining property. Such functionality is achieved through interoperation. Since we cannot expect to have direct and detailed control within a subsystem, emergent functionality must be the result of *policy specification* at subsystem boundaries. The emergence can furthermore be the result from *directed control* (see below), or from self-organization.

Phenomena at the system level can have a weak or strong emergent character, depending on their relation to well-known “laws” in the constituent sub-systems. Weak emergent behavior is unexpected, strong emergent behavior is intrinsically not deducible from first principle truths of the constituent sub-systems [Chalmers 2006].

Concept	Monolithic systems (ML)	ML assumptions	Systems of Systems
Subsystem Integration	. Horizontal and vertical integration . Interface and function definition based on design & decomposition . Reduce coupling, maximize cohesion . Subsystems have no private goal	. Architecture frozen in early design stage . Known (and controlled) control and data flow . Requirements and properties of subsystems known upon integration	. *Uncorrelated requirements:* …..Functions of subsystems not selected by design …..Subsystem behaviour defined by original, independent purpose and local state . *Competition of control:* competing control flows from SoS and subsystem . *Architectural diversity:* no common, frozen architecture . *Fully independent lifecycles* . *Control:* virtual, directed or collaborative
Interoperation . discover interface . understand (and use) interface operation . understand interface semantics	. contained in design . late binding and semantic descriptions (RDF, OWL, SOA) . directed control	. knowledge about semantics, embedded in code . knowledge about particular technologies	. high-level description of goals and services (semantic interoperability) . extend late-binding techniques . rich service interfaces, including extra-functional properties . negotiation
Emergence . properties of systems as a whole not to be attributed to any subsystem in isolation . weak: derivable from known rules in the subsystems . strong: essentially not-deductible	. addressed within the architecture . weak . not explicitly available within the system	. emergent properties addressed in (de)composition	. make properties explicit at subsystem boundaries (rich interfaces) . weak emergence based on policy specification for interoperation . emergence through directed control or self-organization

Figure 1: Summary of properties and comparison between Monolithic System and SoS

Classification

Maier [Maier 1998] discerns three types of SoS: virtual, directed, and collaborative SoS. A *directed* SoS looks mostly like an ML with a centralized control. It means, in fact, that the restriction of managerial independence is dropped. The distinction remains that subsystems can also function autonomously. In a *collaborative* SoS the centralized control cannot enforce cooperation. Applications rely on the voluntary

collaboration between subsystems. In a *virtual* SoS there is no central control: they lack a central agreement process upon purpose; this just *emerges* from the constituent systems.

In all three cases, but especially in the last two, a signaling type of interaction ('commands') is not the right mode; instead, interoperation is based on *negotiation*.

Examining the earlier questions

What are engineering methodologies, processes methodologies, processes and lifecycle defined for SoS engineering?

The research reported in the SoS domain is mainly of a reflective nature: researchers and practitioners recognize that the problems they encounter go beyond traditional system design and integration. In order to increase understanding they have generalized and subsequently taxonomized the concepts and the problems, as summarized above.

Area	Systems engineering	Systems of systems engineering
Focus	Single complex system	Multiple integrated complex systems
Objective	Optimization	Satisficing
Approach	Process	Methodology
Expectation	Solution	Initial response
Problem	Defined	Emergent
Analysis	Technical dominance	Contextual influence dominance
Goals	Unitary	Pluralistic
Boundaries	Fixed	Fluid

Figure 2: distinction between System Engineering and SoS Engineering (from [Keating et. al. 2003])

Keating et al. [Keating et al. 2003] discuss SoS engineering in combination with systems engineering and identify a number of differences between these domains (see Figure 2). They explain that a SoSE process must address issues differently from traditional systems engineering and address system evaluation and evolution as well as system transformation. DSL-based generation of agent (collaborative SoS (directed SoS) are examples of initiatives to manage evolution / transformation / evaluation of systems.

Lewis et. al. [Lewis et.al. 2008] describe SoSE as addressing a double challenge, viz., of generating responses extremely flexibly in changing situations while collaborating effectively across system boundaries. They define an abstract lifecycle addressing the software development in systems of systems. This lifecycle consists of three steps:

1. The independent subsystems contribute a pool of software elements.
2. SoS engineers search through this pool for elements to build integrated SoS capabilities.
3. Subsequently, the relation between the SoS and the original subsystem needs to be established. The nature of this relation defines the dependencies between subsystem and the SoS. For example, if the SoS requires access to certain data the subsystem must allow this access which has to be aligned with the local security policies.

This lifecycle appears to approach the design problem from a similar perspective as Component-Based Software Engineering, i.e., by composing functionality from existing components. The resulting problems are resolved after the fact. In particular, solving problems like control conflicts, performance problems and security conflicts seem to be difficult to address in this way. In addition, the approach seems rather static. Instead, we need to develop an SoSE process in which functionality of a subsystem is represented by services using rich interfaces that admit management of extra-functional properties. By using policies at interfaces we separate the negotiation from interaction.

Not many specific engineering methodologies for SoS are in place. Nevertheless, the state-of-the-art for monolithic systems is relevant to take into account when considering SoS engineering. The role of reference designs and architectures (e.g. the time-triggered architecture), integration verification and tools for that, application of standards for connectivity and middleware (e.g. AUTOSAR), ever increasing use of simulators, software synthesis/code generation, and verification tools is eminent and increasing. Rather new for SoS would be the need for fast integration techniques (smart adapters, system-level awareness for control of operations, design-space exploration optimizing choice and opportunity, test policies). These techniques are only partly available.

Borth et al [Borth 2012] argue that central to the challenge of SoSE is the fact that the system lifecycles have become truly independent. This causes that engineering processes become non-monotonic, divide-and-conquer approaches fail for design, integration, and test, and the consequences of failures cannot be judged easily. Nevertheless, there is a gradual increase going from cooperative systems in a known environment to adaptive systems in an evolving systems-of-systems environment, meaning that not all techniques and methods have to be in place from the start. Ultimately, SoS development requires a new cultural setting in many companies, i.e. involvement of other systems that are not (fully) under control.

What are architecting principles and perhaps styles suggested for SoS?

As discussed before, the essential ingredients of SoS follow from the independence of the subsystems. The architecting principles must therefore address these properties. Operational and managerial independence require negotiations at interfaces to obtain access and budgets. Evolutionary independence requires loose coupling techniques and binding based on introspection and rich descriptions. In addition it calls for fault tolerance techniques taking unexpected behavior as the norm. Taking this further, self-organization or a health subsystem is necessary. Plug-and-play is the best known form of self-organization, but a newer form is found in cognitive networks. Agent-based systems provide means to build such systems. Explicit reasoning of the system about its own health is a relatively new field, closely related to adaptive systems and scenario detection mechanisms. Machine learning and anomaly detection have been around for some time, but only very application-specific.

In order to enable proper functioning of systems in a system-of-systems context, the observe-and-compare-with-expectations paradigm appears inevitable. [Borth et al.] claim that this should be done at multiple levels in the system, resulting in local and global self-reflection strategies.

In the literature four architectural aspects are further emphasized. The first is that SoS is based on communication and networking. The required flexibility and decoupling of operating systems and languages, the robustness for device failures and the likes are naturally achieved by placing system boundaries at networks. Although it might be debatable, this seems a good first choice.

Second, the concept of Service Oriented Architectures is mentioned as a relevant style since it decouples clearly the concepts of service, implementation and specification. In the Genesys project (FP7) [Genesys 2009], a standard service-oriented reference architecture for embedded systems is proposed and several follow-up projects to deploy this have been defined. The current uses of SOA, however, might not be adequate as it often assumes shared ontologies and centralized discovery mechanisms that may not be appropriate *across* systems. In addition, there is no clear and integrated general concept of negotiation, although in the agent-based systems, standards for negotiation protocols have been developed, such as the contract net interaction protocol (CNP) by FIPA. The use of services and explicit specifications admits dynamic integration using model-based adapter technology, where formal methods support compliancy with specifications on the basis of model descriptions.

Third, some authors mention that in order for systems to evolve further, stable intermediate subsystems are required. Hence, in order to build larger systems with more functionality we have to leave the subsystems closed, and build on top of their interfaces. Leaving a system closed means really not interfering with its private architecture, operation, management, evolution, installation etc., but also that it is essentially self-contained in terms of self-adaptation and management. This includes join-and-leave scenarios: what happens with the system if constituent systems are joining and leaving at run-time. This especially imposes challenges on integration and testing of the resulting SOS, which have to take into account runtime integration and testing constituent systems with profoundly different architectures.

Fourth, the following may be considered as a general principle: any addition or change added to the SoS must at least retain existing functionality and quality within the subsystems. Differently phrased, changes must make things better.

Concluding, the concepts of Service Oriented Architectures will be the basis to deal with policies and negotiation at system boundaries. Rather than direct access to internal services a semantic abstraction is provided that admits semantic interoperability but also the negotiation of reservation and access control.

The last two questions are left open for now.

Which existing examples from SoS can we learn from that have benefited from recent insight?

Maritime situational awareness perhaps. The architecture development for such systems have learned that the information flow view is very relevant, and the ability to deal with uncertain information.

Which concrete solutions exist to the set of problems and issues listed and what are the open issues?

DoD SoSes and net-centric systems (USA) have given insights in the essential engineering artifacts that can be used as guidance for the systems engineering approaches of SOS [DoD 2008].
Traffic information systems: radio & TMC broadcast, combined with dynamic road signs; alternatively, TMC in conjunction with navigation systems that also collect GSM location data to predict the traffic situation better.

Finally, a generic diagram summarizes the structural aspects of the above discussion. It is an example how an integration framework for Sstems of Systems can be setup.

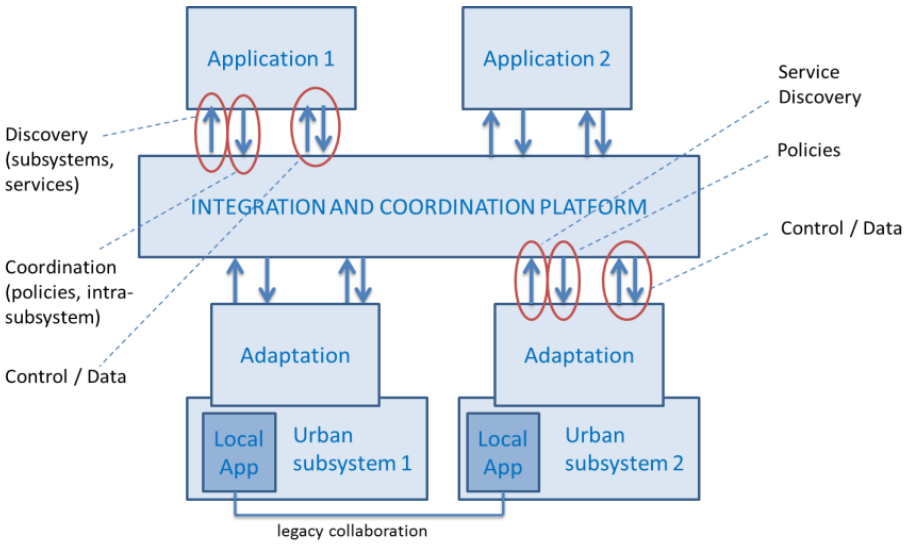

Figure 3: Conceptual diagram of Systems of Systems integration for the case of Urban Subsystems (Smart Cities)

References

[Aarts 2003] Aarts, E., Marzano, S. (editors), "The New Everyday: Visions of Ambient Intelligence", 010 Publishers, 2003

[Borth 2011] Borth, M., and van Loo, S.J., "Engineering Resilient Systems-of-Systems", CSD&M 2011, December 7-9, 2011.

[Chalmers 2006] Chalmers, D.J., "Varieties of Emergence", Chapter 11, of Clayton, P., and Davies, P. (eds.), "The Re-Emergence of Emergence: The Emergentist Hypothesis from Science to Religion", Oxford University Press, 2006

[DeLaurentis 2005] DeLaurentis, D. "Understanding Transportation as a System of Systems Design Problem," 43rd AIAA Aerospace Sciences Meeting, Reno, Nevada, January 10-13, 2005. AIAA-2005-0123.

[DoD 2008] Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering. Systems Engineering Guide for Systems of Systems, Version 1.0. Washington, DC: ODUSD(A&T)SSE, 2008.

[Erl 2005] Erl, Th., "Service-Oriented Architecture: Concepts, Technology, and Design", Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005

[Fisher 2006] David A. Fisher, "An Emergent Perspective on Interoperation in Systems of Systems", TECHNICAL REPORT CMU/SEI-2006-TR-003, ESC-TR-2006-003.

[Genesys 2009] Obermaisser, R., and Kopetz, H., "A Candidate for an Artemis cross-domain Reference Architecture for Embedded Systems", SVH, Germany, 2009

[Lewis et. al. 2008] Grace Lewis, Ed Morris, Pat Place, Soumya Simanta, Dennis Smith, Lutz Wrage, “Engineering Systems of Systems”, SysCon 2008 - IEEE International Systems Conference, Montreal, Canada, April 7-10, 2008.

[Keating et. al. 2003] Charles Keating et. al., “Systems of Systems Engineering”, Engineering Mangement Journal, Sept 2003, 15,3.

[Koestler 1967] Koestler, A.,” The Ghost in the Machine”, 1967. (1990 reprint edition ed.). Penguin Group. ISBN 0-14-019192-5.

[Maier 1998] Maier, M.W., "Architecting Principles for System of Systems," Systems Engineering, Vol. 1, No. 4, 1998, pp. 267-284.
[Poseidon 2012] van de Laar, P., Tretmans, J., and Borth, M., “Situation Awareness with Systems of Systems”, Springer, to be published.

[SAFESPOT 2009] Integrated project FP7, “Cooperative vehicles and road infrastructure for road safety”.

[SOFIA 2009] Artemis project, “Smart objects for intelligent applications”.

[Steels 1991] Steels, L., “Towards a theory of emergent functionality”, Proceedings of the First International Conference on Simulation of Adaptive Behavior, 451-461, February 1991

[UPnP Forum 2008] UPnP Forum, “UPnP Device Architecture 1.0”, Document Revision Date 15 October 2008.