Technical Challenges of SoS Requirements Engineering

Stefan Hallerstede[1], Finn Overgaard Hansen[1], Jon Holt[2], Rasmus Lauritsen[3],
Lasse Lorenzen[2] and Jan Peleska[4]
[1] Aarhus University School of Engineering, Denmark, [2] Atego, UK
[3] Bang & Olufsen, Denmark, [4] University of Bremen, Germany

Abstract – *Taken by themselves separate aspects of systems of systems (SoS) can be addressed by conventional system engineering techniques. That is, at least to a large degree, we know how to address the problems of distribution, emergence and evolution. The specific challenges posed by SoS arise from their combination. Additionally, we have to deal with independence of constituent systems (CS) of SoS, in particular, managerial independence. In this article we focus on technical challenges of mastering SoS requirements. Based on techniques for systems engineering we sketch problems that appear specifically in SoS engineering if we want to be able to use conventional engineering techniques as much as possible. The ultimate aim of our work is to develop tools that can support SoS requirements engineering.*

1 Introduction

The main objective of our work is an integrated approach to requirements engineering of SoS that comprises modeling and validation of requirements as well as verification of design and implementation relative to requirements models. Within our current research we do not address the problem of requirements elicitation of SoS. This research is carried as part of the EU FP7 project COMPASS, addressing the modeling of SoS by formal and informal means. Aspects such as requirements elicitation are investigated separately within the COMPASS project.

The purpose of this article is to provide an overview of the technical challenges we face and their intimate relationships. For example, dealing with requirements validation and verification in the sense above while allowing an SoS to evolve continually demands strong support for requirements tracing.

We consider the following widely recognized characteristics of SoS engineering [1] the main source of the technical challenges we face:

i. *Independence.* The component systems are able to operate and are managed separately.
ii. *Distribution.* The component systems are dispersed and communicate over larger distances.
iii. *Emergence.* The behavior of an SoS exceeds the behavior of its constituent systems (CS).
iv. *Evolution.* The SoS is in continual development and can never be considered fully completed.

Some of the characteristics of SoS engineering pertain generally to systems engineering but are more pressing in the context of SoS, e.g., evolution.

These characteristics are present to varying degrees in the different types of SoS [2]. For our current work we assume that an SoS falls into one of the two following categories:

i. *Acknowledged.* It is under control of one authority.
ii. *Collaborative.* It is *not* under control of one authority.

The industrial case studies described in Sections 2 and 3 present a case of each type. In Section 4 we sketch a requirements engineering process for SoS. Specific technical challenges are discussed in subsequent sections: validation in Section 5, tracing in Section 6 and verification in Section 7. Finally, section 8 draws some conclusions. Sections 2 to 7 have been structured into subsections dealing with *independence*, *distribution*, *emergence* and *evolution* in turn.

2 Acknowledged Case

A case study of an acknowledged SoS is brought forward by the Italian company Insiel. After describing this system in general terms we analyze the case study with respect to the SoS characteristics.

This case study deals with a new *unified emergency call-center* SoS that is to operate in the Fruili Venezia Giulia region of North Italy. One key service of the center to the general public is responding and coordinating efforts to handle emergencies. A similar SoS for the services of the London Emergency Services Liaison Panel (LESLP) has been studied in [3]. The Insiel system offers two views of the current status of the SoS to the call operators. One view permits call operators to get an overview of incoming calls. The other view provides the call operator with a map of the region that overlay on-going emergencies with deployed emergency response units. The system has a room where several call operators are situated. Each operator uses computer equipment and radio to communicate with the system. The system allows an operator to handle and group calls and to dispatch emergency response vehicles to an event. In a major crisis situation many incoming calls regarding the same crisis will appear and operators will have means to group such calls together. An additional set of wall mounted monitors visible to all operators in the call-center room are available to provide a common operational view in such situations.

The principal stakeholders for the system include:
- The Italian government.
- Health care services.
- Police and Fire brigades.
- Insiel.
- The general public at large in case of emergencies.

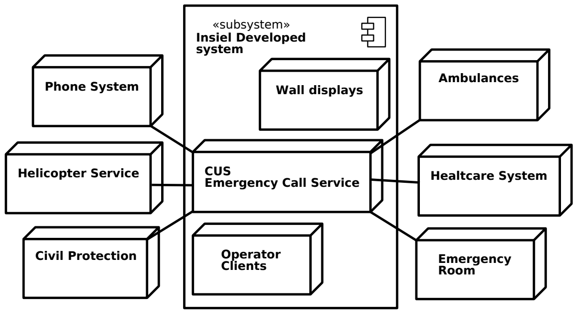

Figure 1: Overview of the Insiel CUS System of systems.

The CS developed by Insiel is communicating with several other CS as depicted in Fig. 1. The most prominent ones are:
- The Phone System.
- The Helicopter Service for mountain and forest rescues.
- The Civil Protection authorities.
- The Healthcare System.
- The Ambulances.

Based on the preceding general description of the case study, we analyse its SoS characteristics. The SoS is ordered by the Italian government, which is considered its single responsible authority. However, on the managerial level stakeholders owning the constituent systems clearly maintain their independence.

2.1 Independence

Accounting for independence of the constituent system is straightforward. As an example consider the phone system. It is operationally and managerially independent. A private phone company independent of the Italian government and Insiel is responsible for it.

2.2 Emergence

Emergent behavior is clearly identifiable. For instance, the overview of the region provided to the operators along with a view of all allocated emergency response resources cannot be provided by the constituent systems alone.

2.3 Evolution

The emergency response service is intended to have a long lifespan. Therefore this system will inevitably evolve on the basis of technological progress alone. One also expects that within its lifetime laws and regulations concerning its operation are going to change. As an example of technological change take the advent of the GPS that made navigational systems omnipresent. Similar technological advances in the future will place new demands on the capabilities of SoS in operation.

3 Collaborative Case study

A collaborative SoS case study is represented by B&O in form of the *connected Audio-Video (AV) products challenge*.

The AV range in the 1980ies was not as diverse as it is today. At that time B&O could deliver a complete range of AV equipment using proprietary technology to connect the equipment. Offering a complete range of products, B&O did not consider to make the proprietary technology publicly available but kept it private to be used only with B&O products and a few select Home-Automation (HA) installers.

Today the trend is towards more open technologies with interoperability as a major concern. The range of products in the AV domain today is much more diverse and complex than in the 1980ies making it expensive and difficult to deliver quality products for the entire range. It is no longer feasible to provide all connected AV products in a home installation from one company.

The modern AV system is no longer limited to just one supplier. Furthermore the border between the AV system and the HA is disappearing, being replaced by the concept of intelligent homes. In future B&O will no longer be able to rely solely on proprietary technology but instead have to follow open standards. Despite of this, even closer integration with HA installers will be necessary. To fulfill the goal of interoperability B&O needs an organisation do drive and specify an interoperability protocol for AV products, so that B&O, the HA installers and other consumer electronics providers have a common baseline for their products. Today the *Digital Living Network Alliance (DLNA)* is starting to become such an organisation, and it provides a protocol [4] that specifies basic interoperability standards.

3.1 Independence

The focus of modern AV systems is to connect stand-alone products into a combined system experience. The products themselves are considered as independent systems and so is their development which takes place at competing electronics providers. In addition, the DLNA organisation—which counts many electronics providers as its members—is developing the protocol and standard for product interoperability. DLNA itself is an independent organisation.

In general B&O has no influence or control over how other products are developed and how the DLNA organisation works with the interoperability protocol.

3.2 Distribution

An AV system is not necessarily situated in the same room or even house, but can be dispersed over several different geographic locations, e.g., summer house and car. However the more common scenario would be different AV products located in one house.

3.3 Emergence

The characteristic features on an AV systems are emergent. A simple example of such a feature is the cooperation of separate audio and video devices that can be controlled by one remote control. The user expects that the devices and the remote control work together without any additional intervention just by placing them close enough to each other.

A more intricate example is dealing with Digital Rights Management (DRM). Today each constituent system will have to comply with DRM regulations. The difficulty with DRM appears when many products are connected. It is necessary to ensure that the AV system still complies with all the constituent systems' DRM regulations. One might even end up in a situation where the constituent systems should be able to form an AV system, but due to legal constraints in the DRM regulations it is not allowed to do this.

3.4 Evolution

AV systems evolve per installation and by way of the underlying technology. AV Systems may simply evolve by adding new products, some of which may result in new emergent properties. If an existing product is added it may also lead to emergent properties such as audio streaming to multiple target devices.

As the AV market is very competitive and in constant evolution, is is evident that the AV system has to follow that evolution. For each new technology the SoS needs to adopt this, but still ensure that the original SoS is not destructed.

3.5 Collaboration

A separate body, the DLNA, drives the interoperability protocol for AV systems. The manufacturers of AV products try to attain collaboratively the minimum interoperability level specified by the DLNA.

4 Requirements Process

4.1 Traditional versus SoS Requirement Engineering

Traditional system engineering requirements processes are defined to support development of complete new systems, where all important requirements are defined up-front, before the system is architected and implemented. Another typical characteristic is that these systems have a single authority or customer, who controls the system development. These processes are not adequate for the development of System of Systems as described in [5] and [6]. As shown in

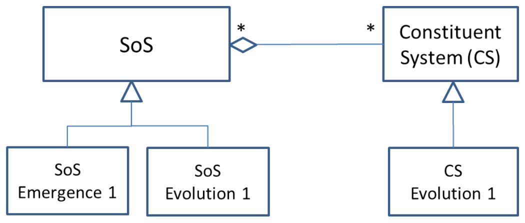

Figure 2: SoS System Model - at an early life-cycle stage

Figure 2, a System of Systems is composed of a number of constituent systems, some of which already exist and have a purpose of their own and are managed by their own authority. Another important aspect to consider is, that a given constituent system in principle can belong to more than one SoS, which could lead to conflicting requirements for the CS.

Figure 2 shows how the concepts of evolution and emergence are related to the SoS and its CS, illustrating that evolution can occur both at the SoS- and the CS-level, whereas emergent behaviour is defined to be only at the SoS-level.

4.2 Independence

In the SoS literature high level system goals are called *system capabilities* which are broken down to more specific requirements for either the SoS as a whole or allocated as more detailed requirements for the constituent systems participating in a given SoS.

Independence means operational as well as managerial independence of the constituent systems. This implies that new system capabilities, requirements and changes shall be dealt with at two levels, the SoS- and the CS-level. In relation to a process for SoS requirement engineering this process should account for this situation, where a given system capability has to be broken down to requirements that belong to either the SoS or to one or more of the constituent systems.

Handling an SoS requirement can be regarded as a top-down process whereas handling a requirement for a constituent system can be regarded as a bottom-up process.

An SoS requirement process has to account for these two different scenarios and offer a different approach for each. When a new capability, a specific requirement or a change request is introduced it should, as one of the first steps, be analyzed and characterized as either belonging to the SoS- or to the CS-level and be handled by the corresponding authority.

4.3 Distribution

An SoS will normally be a distributed system, where the constituent systems are geographically distributed and exchange information based on commonly agreed upon communication protocols. This implies that the stakeholders require a high coordination effort with respect to the development process as well as other engineering aspects such as verification of the complete SoS. Consequently, handling of a distributed stakeholder and user community should be included in a SoS requirement engineering process.

4.4 Emergence

As illustrated in Figure 2, emergence is a characteristic which belongs to the SoS-level and describes behaviour obtained at the SoS-level based on the collaboration of a number of CS to obtain this new behaviour. Somewhat simplified, an emergent SoS requirement is shown as a subtype but can also have other relations to the original SoS. One example could be an emergent SoS which both enhances and changes some of the original SoS behaviour.

4.5 Evolution

As shown in Figure 3 evolution can be introduced at the SoS-level or at the CS-level. Evolution is natural for these long living SoS, where changes can be caused by technological changes, by new or changed user capabilities, or by new legal regulations, introduced, for example, by the government.

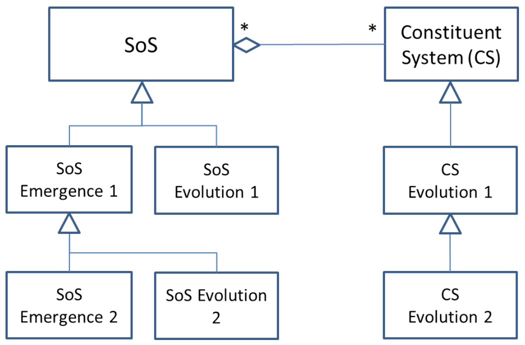

Figure 3: SoS System Model - at a later life-cycle stage

In addition to this an SoS and its constituent systems have long system life times, where each CS can be in a different stage of its individual system life-cycle.

An SoS requirement process should therefore cover a continual development life-cycle scenario, where new capabilities, requirements and changes to existing requirements are to be handled by the process at the SoS- and the CS-level.

4.6 An SoS requirement engineering process proposal

In further work we will fully define the *COMPASS SoS requirements engineering process* proposal based on the process requirements described above and plan to include capability engineering aspects as described in [7], where the authors propose the following two parallel steps: (i) develop SoS capability objectives and (ii) develop a concept of operations to improve the quality of the requirements. Following these two steps a set of high level SoS requirements can be formulated.

Another source to be incorporated in the COMPASS SoS requirements process is the SoS requirement process proposal described in [5] which consists of the following steps:

1. Identify SoS context
2. Identify SoS and individual system goals
3. Understand SoS interactions
4. Identify individual system capabilities and
5. Analyse the gap

We apply the initial proposal for the COMPASS SoS requirement process to the two industrial cases described in Sections 2 and 3.

5 Requirements Modeling

We consider semi-formal approaches, e.g. SysML [8] as well as formal ones, e.g. VDM [9] to requirements modeling and validation. In our experience semi-formal and formal methods can be complementary and sometimes they can only be deployed jointly [10].

The modeling approaches above (and similar ones) have been proved useful for systems modeling and have been used in industry to varying degrees. They assume that the development and maintenance is under control of one authority. Independence is not provided for. They may be able to deal with evolving requirements but there is no explicit support for this. The approaches have not been developed for this purpose. A similar statement can be made about emergence.

5.1 Independence

Dealing with independence requires agreement among the constituent systems about their functionality. Contract-based modeling [11] could be used to enforce agreement. Note that SoS requirements are naturally linked with architectural concerns. It is not possible to delay architectural design that concerns the composition of the CS themselves. This constraint introduces the partitioning of the requirements with respect to emergence. The contracts describe non-emergent properties of CS. Emergent properties are discussed below.

5.2 Distribution

Constituent systems of an SoS often are in *distant locations*. We take distant location to mean that the CS can only communicate by means of some communication network. Because of the resulting architectural concerns we must also deal with communication facilities during requirements modeling. Specification techniques for high-level modeling of communicating systems are known, e.g. [12]. However, specification of communication details –even abstract details– can make requirement models overly complex. Traditionally, one tries to avoid mentioning communication details in requirements for that reason. The partitioning of the requirements into emergent and non-emergent ones could help to identify those requirements that can be modeled on the level of the CS.

5.3 Emergence

We aim at an engineering method for SoS. As such we have a limited view of emergence as properties that exceed those of the CS but are attainable by engineering.[1] This is also referred to as *weak emergence* [13]. A formal engineering-oriented view of weak emergence based on refinement is discussed in [14]. Refinement [15] is used to bridge the abstraction gap between the emergent properties to be verified and the model consisting solely of the CS. Refinement has also been used in [16, 17] to model requirements, and validation of such models has been discussed in [18]. We believe, that some form of refinement will be necessary in order to address the problem of emergence.

5.4 Evolution

System evolution is an issue for any system intended for long-term use. What changes in the context of SoS is that

[1]As opposed to far-reaching philosophical definitions concerning, e.g., consciousness.

evolution is considered the normal case. Any method not taking evolution into account will be of no use. For our purposes we understand by evolution any continued development of an SoS and its CS. This may happen by adding, changing or removing constituent systems, as well as, changing emergent requirements. We observe that we have only little control over what is inside the CS. We know the contracts and can change them or may be forced to change them. As a consequence, emergent properties may fail to hold or new emergent properties may hold. The most interesting problem of evolution of SoS appears to be related to emergence and high-level requirements on CS. In [19] a method and tool for incremental modeling is discussed. It is incremental in two respects: It supports frequent changes to models and it supports a notion of refinement that permits to add detail to a model gradually. We believe, a similar approach could also benefit the modeling of SoS requirements.

6 Requirements Tracing

6.1 Independence

Independence has two different aspects [1]: operational independence and managerial independence. Operational independence imposes architectural constraints on the SoS. In principle, it is known how to deal with this in conventional requirements engineering for systems. Managerial independence imposes that requirements may have to be distributed among the stakeholders that supply constituent systems. As a consequence, tracing requirements to stakeholders becomes a dominant problem. Of course, conventional techniques of tracing to designs and programs still apply. But special focus needs to be placed on impact analysis, in particular, to constraining impact: If requirement changes affect too many suppliers of constituent systems at once, they may be difficult and expensive to implement.

6.2 Distribution

Distribution will not pose new problems that do not already appear in conventional systems engineering. One would expect, however, that there is some overlap between independence and distribution. The architecture of an SoS will usually match with managerial independence. Hence, the SoS architecture may serve as the basis for tracing requirements to different stakeholders. This is more evidence in favor of including architectural concerns early in the requirements modeling as opposed to the practice in traditional requirements engineering.

6.3 Emergence

Emergent properties cannot be expressed on the level of the CS. So we cannot expect to identify emergent requirements with CS. We have argued above that refinement could address emergence in SoS. The employed notion of refinement will have to provide the means to trace requirements. The abstraction gap closed by refinement with respect to property validation must also be closed with respect to requirements tracing. Such a method of refinement with requirements tracing is outlined in [17] for conventional systems engineering. It is based on the WRSPM model [20] for reasoning about system requirements.

6.4 Evolution

Evolution can be split into the two broad categories: CS evolution and SoS evolution. Potentially any change in the configuration of an SoS concerns all stakeholders. In practice, often more than one will be concerned. Changes in the contracts of some CS often affect correctness assumptions made in other constituent systems. Because regular changes are considered to be the normal case, predicting the effect of the changes and determining who is involved is of central importance.

When emergent requirements change, it will be important to recognise as soon as possible who will be involved in addressing the change and in what way. It is possible that, in order to implement the new emergent requirement, contracts have to be changed. It will be necessary to see which parts of the contracts are affected and how. Such information can feed into the effort prediction of the different stakeholders.

7 Requirements Verification

Verification needs to take into account the heterogenous characteristics of an SoS. Neither formal verification nor dynamic testing by themselves are sufficient for verifying SoS on a realistic scale. In particular, the SoS size will nearly always defeat any approach to comprehensively model software and hardware in a formal way. As a consequence, the proper integration of software and hardware has to be investigated by means of testing.

7.1 Independence

Verification has to be coordinated between the different stakeholders. Contracts will form the basis for this. In COMPASS we favor an approach where algorithmic properties of constituent systems should be formally verified, while hardware/software integration is verified by means of testing. On SoS level, constituent system properties are represented by contract abstractions, so that formal verification can be performed again for checking component cooperation logic. Again, system integration aspects have to be verified by means of dynamic testing.

7.2 Distribution

SoS have open architectures where CS may join or leave. On the one hand, this makes testing of the SoS difficult because only specific SoS configurations can be checked. On the other hand, it makes formal verification difficult because the assumptions that can be made about the CS are weak. Combining the two approaches, we expect that the most critical properties can be formally verified while the other behavioural requirements can be checked by dynamic testing. In the former case completeness of the verification is crucial, whereas in the latter we only verify the existence of a suitable implementation.

7.3 Emergence

Emergent properties cannot be verified on the various CS separately. Whether formal verification is used for specific properties or dynamic testing, the effort needs to be coordinated. Both methods should support a notion of refinement that could be used to bridge the abstraction gap between SoS and CS.

7.4 Evolution

Regular changes in the requirements and their models should be automatically taken into account as much as possible. A record about the amount of manually developed tests connected to requirements should be available so that the actual effort of a change can be predicted. To deal with frequent changes tracing of requirements to the tests that verify them is required: failing tests should be related to the requirements that they invalidate.

8 Conclusion

We have described technical aspects of requirements engineering in the context of SoS. In the coming years we intend to fully define and evaluate the development process sketched in Section 4. The evaluation will be based on the two case studies that we have described in Sections 2 and 3.

We have related the main technical activities of the requirements process, validation, tracing and verification to the core characteristics of SoS, namely, independence, distribution, emergence and evolution. This permits us to identify the main technical challenges of SoS requirements engineering. Ultimately, defining the process and addressing these challenges should clarify what kind of tool is needed in order to support SoS requirements engineering. We believe, the task of SoS requirements engineering will benefit from tools for modelling, verifying and simulating scenarios of SoS as does standard requirements engineering.

Acknowledgment

This research is funded by the EU FP7 COMPASS project under grant agreement no.287829.

References

[1] M. W. Maier, "Architecting Principles for Systems-of-Systems," *Systems Engineering*, vol. 1, no. 4, 1998.

[2] J. S. Dahmann, G. R. Jr., and J. A. Lane., "Systems engineering for capabilities," *CrossTalk Journal (The Journal of Defense Software Engineering)*, vol. 21, no. 11, pp. 4–9, 2008.

[3] J. W. Bryans, J. S. Fitzgerald, and T. McCutcheon, "Refinement-based techniques in the analysis of information flow policies for dynamic virtual organisations," in *PRO-VE*, 2011, pp. 314–321.

[4] ipTVnews., "Installed base of dlna devices exceeds 440mn," January 2011. [Online]. Available: http://archive.iptv-news.com/iptv_news/january_2011_2/installed_base_of_dlna_devices_exceeds_440mn

[5] G. A. Lewis, E. Morris, P. Place, S. Simanta, and D. B. Smith, "Requirements engineering for systems of systems," in *IEEE SysCon*, 2009, pp. 1–6.

[6] C. Ncube, "On the engineering of systems of systems: Key challenges for the requirements engineering community," in *RESS*, 2011, pp. 1–4.

[7] J. S. Dahmann, G. R. Jr., J. A. Lane, and R. Lowry., "Systems engineering artifacts for sos," *"IEEE A&E Systems Magazine"*, pp. 22–28, January 2011.

[8] J. Holt and S. Perry, *SysML for Systems Engineering*. IET, 2008.

[9] J. Fitzgerald and P. G. Larsen, *Modelling Systems – Practical Tools and Techniques in Software Development*, 2nd ed. Cambridge University Press, 2009.

[10] R. Gmehlich, K. Grau, S. Hallerstede, M. Leuschel, F. Lösch, and D. Plagge, "On fitting a formal method into practice," in *ICFEM'2011*, ser. LNCS, S. Qin and Z. Qiu, Eds., vol. 6991. Springer, 2011, pp. 195–210.

[11] R. Payne and J. Fitzgerald, "Evaluation of architectural frameworks supporting contract-based specification," University of Newcastle upon Tyne, Computing Science, Tech. Rep., 2010.

[12] J. C. P. Woodcock and A. L. C. Cavalcanti, "A concurrent language for refinement," in *IWFM'01: 5th Irish Workshop in Formal Methods*, ser. BCS Electronic Workshops in Computing, A. Butterfield and C. Pahl, Eds., 2001.

[13] M. A. Bedau, "Weak emergence," in *Philosophical Perspectives 11: Mind, Causation, and World*, J. E. Tomberlin, Ed. Blackwell, 1997, pp. 375–399.

[14] J. Sanders and G. Smith, "Emergence and refinement," *Form. Asp. Comput*, vol. 24, no. 1, pp. 45–65, 2012.

[15] C. C. Morgan, *Programming from Specifications*, 2nd ed. Prentice Hall, 1994.

[16] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[17] M. Jastram, S. Hallerstede, and L. Ladenberger, "Mixing formal and informal model elements for tracing requirements," in *AVOCS 2011*, 2011.

[18] S. Hallerstede, M. Leuschel, and D. Plagge, "Validation of formal models by refinement animation," *Sci. Comput. Program.*, 2011.

[19] J.-R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, "Rodin: an open toolset for modelling and reasoning in event-b," *STTT*, vol. 12, no. 6, pp. 447–466, 2010.

[20] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave, "A reference model for requirements and specifications," *IEEE Softw.*, vol. 17, pp. 37–43, 2000.