Communication Architectures for System of Systems Engineering

Jorge L. LOVACO[1, a, *] and Karl ZUBER[1, b, *]Raghu Chaitanya MUNJULURY[1, 2, c]Petter KRUS[1, d]

[1]Department of Management and Engineering, Linköping University, Olaus Magnus väg, Linköping, Sweden
[2]Saab Aeronautics, Bröderna Ugglas gata, Linköping, Sweden
[a]jorge.lovaco@liu.se, [b]karl.zuber@liu.se, [c]raghu.munjulury@liu.se, [d]petter.krus@liu.se

Keywords: *System of Systems Engineering, System of Systems Architecture, Communication, Software Architecture, Agent-Based Modelling and Simulation*

Abstract System of Systems Engineering (SoSE) studies how independent constituent systems interact to generate emergent capabilities. However, it is challenging linking the study operations and SoSE to software implementation and agent-based simulations. This paper considers communication as the primary architectural concern and examines how software architectures can improve SoS modelling and simulation. In particular, by relating software architectures to Observe–Orient–Decide–Act (OODA) loops, which are a promising structure for aligning communication, decision logic, and agent interactions with modular and traceable agent architectures. The goal of this work is to explore how communication within SoS architectures can be better structured, represented, and analysed through software engineering and agent-based modelling frameworks in the context of hierarchical and cooperative operational environments, where information exchange directly influences system performance and emergent outcomes.

Background

System of Systems Engineering (SoSE) studies the methodologies for developing assemblies of systems in such a way that the collective brings new capabilities [1]. These systems operate independently but also as a part of an overarching System of Systems (SoS) architecture designed to form a collaborative and communicative network. The communication and interaction between systems are what makes the new capabilities emerge [1]. It is difficult, however, to provide an architecture for the SoS without being aware of the emergent capabilities beforehand, for that reason, modelling and simulation is used for finding these unforeseen capabilities. However, when using software for implementing a simulation tool

for the SoS, not knowing the overall architecture represents a challenge that could make the models diverge from reality. The present work proposes the use of communication as the starting point for proposing a software architecture that could be easily relatable when considering options for the SoS architecture. By considering communication early on in the process, issues related to communication in the form of delays, scalability, traceability, or interoperability can be addressed so they do not become an issue later on.

Systems Engineering as Starting Point

There have been attempts to connect the Systems Engineering (SE) Vee Model to SoSE [2], but SE design and analysis assumptions are not always suitable for every SoS case [1]. There are accepted heuristics for architecting an SoS that can be found in the literature [3], being considered the correct modelling of interactions as the most critical one for capturing emergence. A very interesting approach for defining system architectures is *heuristic extrapolation* [4]. This method is particularly relevant as it is context sensitive and abstraction is removed iteratively in the design representation. The heuristic extrapolation method takes a general descriptive heuristic to derive domain-specific heuristics and rationale for decisions.

Adopting Software Engineering

The process of modelling and analysis of SoS architectures is inherently iterative. For

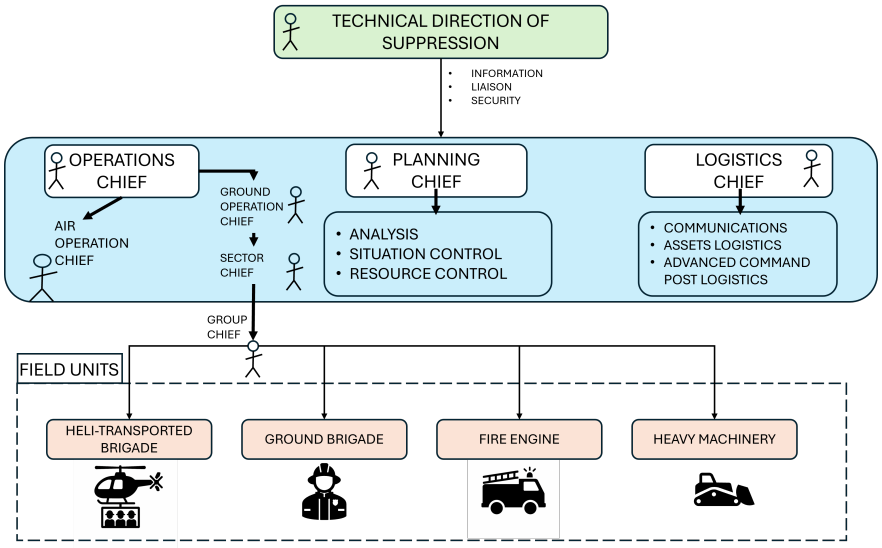

Figure 1: *Incident Command Structure for Wildfire Suppression. Adapted from [10].*

SoSE studies, the development of a simulation model is going to depend on the software architecture chosen for representing interactions during an operation. However, the goal of implementing such model to be used during SoSE processes is to be able to make decisions for defining the SoS architecture and interactions. Moreover, for a proper analysis, if the decided software architecture is representative of the SoS architecture, obvious parallelisms between them will rise, and engineers should be able to extrapolate from one another more easily and intuitively. This coupling usually requires an iterative process with numerous trade-off discussions. For parallelisms, considering similar architectures that are already in use simplifies the process. Shown in Fig. 1 is a simplified hierarchy for wildfire suppres-

sion operations, and by using these established structures, it is possible to choose amongst the already defined software architectures. Many of them are well-known [5] and provide guidance for implementing the models required for SoS analysis. Moreover, many of these software architectures have well-defined layers, a clear structure of how the information is propagated, and whether a layer can access resources from other layers. Using these software architectures as a starting point for discussing SoS architectures is considered optimal, as they are well defined and they can be used directly for the implementation of models while keeping traceability of the reasoning behind the decisions taken. To improve the process, one of the most recent methods proposed combines mathematical formulations with model theory, which has been validated and used for system and software design bringing numerous advantages [6].

In terms of modelling and simulation for SoSE, agent-based models are usually used [1].

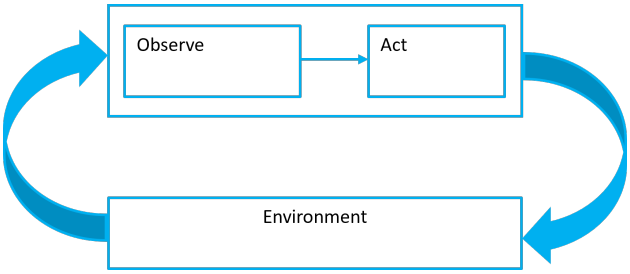

Figure 2: *High-level agent architecture.*

They focus on the interactions of agents, both between them and with their environment, for finding emergent behaviours or missing capabilities that could define requirements for designing new constituent systems. A traditional high-level software architecture of an agent separates the functions for observation of the environment and the functions for acting upon the environment [7], as shown in Fig. 2. However, this architecture is insufficient when trying to represent agents with more advanced interaction capabilities, such as an incident commander agent in a wildfire suppression operation whose interactions are roughly represented in Fig. 3. The goal behind these agent architectures and agent-oriented programming is to

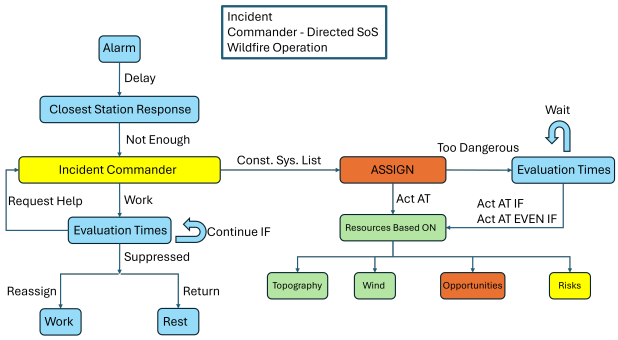

Figure 3: *Incident commander agent interactions.*

perform all the coding considerations as if they were mental states [9], being the difference with object-oriented programming that agents can choose whether to perform a task, and objects cannot. The agent could also be implemented with beliefs and commitment rules.

Agents then could be defined in terms of being communicative or not [7]. For the specific case of SoS, an interesting option for implementing agents would be to adopt an architecture based on *Observe-Orient-Decide-Act* (OODA) loops, which would allow to compartmentalise the implementation of the agents' behaviour in each part of the loop, define clear flows of information and how it is shared, and include interactions in the *Decide* and *Act* steps of the loop. This architecture could include forms of artificial intelligence and even be extended with agent intentions [8].

Conclusion

Despite the ongoing research about SoSE and software architectures, there seems to be no method considered standard for linking SoS architectures and software implementation, especially through communication logic. The method of extrapolating heuristics and the consideration of software structures do not fully relate to how information flows, transforms, and supports decisions during an SoS operation. Software architectures have well-defined communication layers, but those layers do not fully represent SoS decision dynamics. To address this disconnection, this paper has proposed communication-centred architectures as the starting point, considering OODA loops particularly relevant, as they can relate system behaviour during real operations directly to agent implementations in simulation models. The OODA loops have structured information flow and a traceable and modular separation of concerns, which maps in a straightforward manner to both software implementation and SoS interactions, and how information evolves into decisions. These reasons make the loop valuable for SoSE applications where coordination and communication fidelity are necessary to find any emergent system performance. While OODA loops alone do not solve all SoSE challenges, their simplicity, modularity, and operational clarity provide a framework for agent-based models.

Future work should therefore focus on implementation and verification of an OODA-based framework with special attention to the communication layer for SoS simulation. Particularly relevant topics for future research are the definition of messaging (to represent human communication), evaluate the scalability and interoperability in a model with heterogeneous agents, and verify that the OODA loops do indeed improve traceability between SoS architecture decisions and software implementations.

ACKNOWLEDGEMENTS

The research presented in this paper has been performed in the framework of the COLOSSUS project (Collaborative System of Systems Exploration of Aviation Products, Services and Business Models) and has received funding from the European Union Horizon Europe programme under grant agreement No. 101097120. The Swiss participation in the Colossus project is supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 22.00609. The authors would also like to acknowledge the Swedish Innovation Agency (VINNOVA) and The Swedish Defence Materiel Administration (FMV) for financial support through the grant 2019-05371.

References

[1] D. A. M. K. &. G. C. DeLaurentis, System of systems modeling and analysis., CRC Press., 2022.

[2] O. Hoehne, The SoS-VEE Model: Mastering the Socio-Technical Aspects and Complexity of Systems of Systems Engineering (SoSE)., INCOSE International Symposium, vol. 26, n° 11,

[3] M. W. Maier, Architecting principles of system-of-systems, Systems Engineering, vol. 1, n° 14, pp. 267-274, 1998.

[4] Maier, M. W. (1994, August). Heuristic extrapolation in system architecture. In INCOSE International Symposium (Vol. 4, No. 1, pp. 486-493).

[5] Bass, L., Clements, P., & Kazman, R. (2021). Software architecture in practice. Addison-Wesley Professional.

[6] Dickerson, C. E., Wilkinson, M., Hunsicker, E., Ji, S., Li, M., Bernard, Y., ...& Denno, P. (2020). Architecture definition in complex system design using model theory. IEEE Systems Journal, 15(2), 1847-1860.

[7] M. Wooldridge, An Introduction to Multiagent Systems, John Wiley & Sons, LTD., 2002.

[8] Silvander, J., & Angelin, L. (2019). Introducing intents to the OODA-loop. Procedia Computer Science, 159, 878-883.

[9] Y. Shoham, Agent-oriented programming, Artificial intelligence, vol. 60, n° 11, pp. 51-92,

[10] C. d. L. C. I. Forestales, ESTRUCTURA ORGANIZATIVA DEL SISTEMA DE GESTIÓN DE EMERGENCIAS EN INCENDIOS FORESTALES, Gobierno de España, 2008.