From Models to Management and Back: Towards a System-of-Systems Engineering Toolchain

Géza Kulcsár
IncQuery Labs Ltd.
Budapest, Hungary
geza.kulcsar@incquerylabs.com

Kadosa Koltai
IncQuery Labs Ltd.
Budapest, Hungary
kadosa.koltai@incquerylabs.com

Szvetlin Tanyi
AITIA, Inc.
Budapest, Hungary
szvetlin@aitia.ai

Bálint Péceli
evopro Innovation Kft.
Budapest, Hungary
balint.peceli@eilabs.com

Ákos Horváth
IncQuery Labs Ltd.
Budapest, Hungary
akos.horvath@incquerylabs.com

Zoltán Micskei
Budapest University of Technology and Economics
zoltan.micskei@mit.bme.hu

Pál Varga
Budapest University of Technology and Economics
pvarga@tmit.bme.hu

***Abstract*—Through the increasing complexity and dynamics of industrial automation scenarios, the notion of Systems of Systems (SoS) has gained importance in this field. Here, interconnected constituent (hardware as well as software) systems collaborate in order to achieve common goals and to increase the efficiency of certain industrial applications.**

The Arrowhead Tools project aims at proposing a comprehensive, flexible platform for supporting SoS engineering in its every phase, in the form of an engineering toolchain. Thereby, although the SoS operation aspect has been already elaborated on, the design phase and the usage of design artifacts as part of a continous tool interoperability scenario received less attention so far. In this paper, we describe such a toolchain interoperability scenario, paving the way towards an established, integrated solution, by linking systems modeling practices with SoS operational management. In particular, we propose a custom-tailored abstract SysML profile for Arrowhead SoS design, and a prototype implementation for a bidirectional link between SoS models and a tool for managing Arrowhead SoS via a custom-tailored textual format. We illustrate the approach through a realistic industrial application.

I. Introduction

Recent trends in industrial automation bring forth an ever-increasing need for reliable techniques to support the various engineering phases connected to design, deployment as well as operation, maintenance and evolution of those systems, and also *Systems of Systems* (SoS) as considered in the ISO/IEC/IEEE 21839 standard and a number of further publications (e.g., among the first ones, [1]). The notion of SoS is not a single, clearly delineated solution, but rather refers to a number of related approaches [2]. Essentially, an SoS consists of complex cooperating *constituent* systems, brought together as an SoS to reach a goal that is not achievable by the constituent parts on their own. In turn, the SoS concept has become a central notion for advancements in the field of large-scale industrial and manufacturing optimizations based on the widely interpreted *Internet of Things* (IoT) concept. In this particular context, devices being interconnected with a varying, but accentuated degree of autonomy helps in increasing productivity by improving on the self-organizational ability of robotic devices, sensors as well as other hardware and software components [3].

Regarding the latter, i.e., the software components of complex SoS, the inherent dynamics and intended interoperability of such systems poses a considerable challenge when it comes to maintaining an overview of the different systems within the SoS and also a balance between their correct inner behavior and their ability and willingness for an active, yet secure cooperation with other systems and SoS. Thus, established engineering principles for building reliable *closed* systems have to be considered, but extended towards SoS interoperability.[1]

That issue of conflicting engineering goals calls for a twofold, but interrelated design methodology: on the one hand, *model-based engineering* techniques can support a founded planning process and initial SoS overview, which, however, should be coupled with an SoS-specific operational management interface allowing for dynamic SoS reconfiguration.

The *Arrowhead* initiative[2] aims at providing a full-fledged engineering solution fostering digitalization and industrial automation via *service-oriented* SoS design and operation – with special interest on interoperability and integrability. In particular, the *Arrowhead Tools* project[3] aims at proposing a toolchain which covers our envisioned comprehensive SoS engineering scenario encompassing every typical *engineering phase* from requirements specification to SoS deployment to operation and maintenance. Therefore, the Arrowhead framework provide an adequate starting point for proposals towards the aforementioned multi-objective SoS engineering paradigm, creating an additional layer of interoperability, not just on the level of devices but also already on the level of the toolchain linking the different engineering phases.

Particularly, to address some aforementioned issues of SoS modeling, we take a first step towards bridging a perceived

[1]Here, by interoperability, we mean a general concept of emerging component interaction as apparent in SoS, and not necessarily just inter-vendor compatibility.

[2]https://arrowhead.eu

[3]https://arrowhead.eu/arrowheadtools

gap between *design-time* and *run-time* (or *operational)* SoS engineering artifacts, as appearing in the context of Arrowhead. Currently, operational management of service-based SoS has already gained some maturity; as for Arrowhead, the *Arrowhead Framework* provides an open-source, platform-independent software offering[4] for implementing, launching and managing service-oriented SoS instances.

However, supporting the systems design aspect is lagging behind due to the aforementioned conceptual challenges. Moreover, even if conceptually considered together, design-time and run-time artifacts are conceived in complete separation, hampering the interoperability of the engineering toolchain. Although the *model-based systems engineering* (MBSE) community has a substantial knowledge base on systems engineering best practices, similar interoperability shortcomings apply, namely, (i) mostly considering model artifacts separately from their context and (ii) maintaining an implicit closedness assumption of the modeled system, leading to a clash with SoS principles.

In the present paper, we aim at improving on state-of-the-art, through a bridging approach building on both sytems modeling and Arrowhead SoS management practice, articulating a proposal towards filling the gap. To address the challenges mentioned above, we make the following contributions:

- We formulate a lightweight, *high-level SoS modeling profile* for devising Arrowhead SoS instances on an abstract level. The simplicity of the profile makes it accessible for a broad audience and a variety of different stakeholders.
- Building on existing SoS architecture concepts, we define a *custom-tailored exchange format* between SoS models (based on the profile above) and corresponding run-time SoS instances.
- Based on this exchange format, we realize a *bidirectional import/export functionality* between those domains, thus, manifesting an actual link between design-time and run-time artifacts.
- We demonstrate the added methodological value of the achieved toolchain interoperability by creating a *validation module* realized over SoS models, considering changes and model evolution both in the planning as well as the execution phase, i.e., both within the models and in their operational counterparts.

As for the systems modeling aspect, we rely on concepts and tooling around the *Systems Modeling Language* (SysML), an established standard, modeling language and methodology for systems engineering [4]. Note, however, that the minimalistic design of our profile allows for an easy portability for, e.g. UML or other related modeling approaches. In particular, the implementation is based on *MagicDraw*[5], one of the most wide-spread systems modeling tools; the modeling part of our contribution is realized technically as a plugin for MagicDraw.

Regarding Arrowhead operational management, the Arrowhead Framework already includes a web-based *Management Tool*[6] for creating and, as the name suggests, managing (i.e., reconfiguring and maintaining) Arrowhead SoS instances without having to directly work with their programmatic details. The import/export functionality has been realized on both the MagicDraw and the Management Tool side, allowing for a bidirectional communication of abstract SoS descriptions.

In the following, we provide details on our modeling and tooling, backed by a proof-of-concept implementation of the aforementioned contributions. The rest of the paper is organized as follows: after introducing some necessary background knowledge on Arrowhead and SysML concepts as well as the considered engineering phases (Sect. II), we illustrate, using an intuitive running example, on the concepts of our abstract SoS modeling profile (Sect. III), the exchange format and its run-time compatibility via the Management Tool (Sect. IV), as well as the model validation feature (Sect. V). Afterwards, we review some related approaches (Sect. VI) and conclude while also outlining future work (Sect. VII).

II. Background

A. Engineering phases in Arrowhead

The *engineering workflow* and the aforementioned engineering phases are central methodological artifacts in Arrowhead, adopting a well-founded (slightly customized) model from the *ISO/IEC 81346* standard, referred to as the *Extended Automation Engineering Model* (EAEM).

The original standard is rooted in an engineering worldview where the workflow follows a strict sequence. However, recent technological advancements and the increasing need for dynamics and adaptivity require a flexible, still founded engineering workflow. To meet those criteria, Arrowhead retains the traditional engineering phases as in the original workflow model, but proposes some additions and imagines them not as a chain, but as a co-existence of phases with a hinted, but not obligatory ordering: (i) Requirements, (ii) Functional Design, (iii) Procurement and Engineering, (iv) Deployment and Commissioning, (v) Operation and Management, (vi) Maintenance, (vii) Evolution, and (viii) Training and Education.

The relevance of these separated phases to our present contribution lies mostly in them revealing a clear separation of *design* and *operation* activities (noting that phases (iii) and (iv) do not necessarily have a clear status and their interpretation might vary in different communities and applications). In the following, we focus on the two arguably most characteristic phase on each side of the distinction: we demonstrate toolchain-level interoperability (as advocated by the Arrowhead Tools project) *and* the lowering of the design–operation barrier by proposing a combined functionality naturally addressing both the Functional Design and the Management phases.

B. Functional Systems Design in SysML

As for Functional Design, we rely on the established systems modeling language and methodology, SysML [4]. SysML

[4]https://github.com/arrowhead-f

[5]https://www.nomagic.com/products/magicdraw

[6]https://github.com/arrowhead-tools/mgmt-tool-js

is a dialect of the well-known Unified Modeling Language (UML), custom-tailored to meet the specific needs of systems engineering activities. Although various textual representations exist, *diagrams* remain the main representation format of UML and derived modeling languages such as SysML. UML and, in turn, SysML provide a number of different diagram types to represent different facets of the system to be modeled, from requirements to static structures to communication protocols. For further general details, there is a large variety of textbooks available — e.g., for practical details of SysML, refer to the comprehensive book of Friedenthal et al. [4].

In the present paper, we focus on a simple approach to functional design, therefore, we argue, we are on the level of the basic high-level SysML elements called *blocks* as represented on a *block definition diagram*.

An UML dialect is realized by the so-called *profile* mechanism. Note that profiles are generally available in UML and, thus, the same principles can be applied even within SysML, being a UML profile itself: in turn, we exploit the same mechanism, especially its *stereotype* element, to represent our simple domain of Arrowhead SoS design (cf. Sect. III).

C. SoS Management in Arrowhead

As for the SoS Management phase, systems themselves should be configured in order to allow their interworking. In the Arrowhead nomenclature, mandatory core systems (Service Registry, Orchestration, Authorization) provide initial security methods and functions for the Service Oriented Architecture (SOA) approach [5]. The supporting core systems of Arrowhead (i.e., Gatekeeper, Gateway, Event Handler, etc.) aid SoS integrators and developers with native functionalities instead of having to use legacy entities for common features.

The specific Application Systems are those sensors, actuators, data processing and visualisation systems, etc., that provide the main functional purpose of the SoS. While their initial operations are also supported via Arrowhead (i.e. the Configuration core system or the systems executing the On-Boarding procedure), their dynamic connection reconfiguration is also done by the Arrowhead mandatory core systems in a SOA manner (Lookup, Late-binding, Loose Coupling).

The SoS management is supported by the *Arrowhead Management Tool*, which allows the SoS setup for the operator through configuring the (mandatory and possibly some supporting) core systems. This can be done manually *or partly even automatically*, as described in the rest of the paper.

D. Running Example: Semi-Automated Factory

We illustrate our proposed SoS engineering solution through an industry-relevant use-case of manufacturing task management in a so-called *semi-automated factory*, where the basic requirements reflect actual industrial needs, but the presentation has been simplified for the sake of compactness.

In this scenario, the manufacturer (i.e., a factory owner) aims at increasing the level of automation from 70% to 95% in one of its production plants creating medical care products. That imposes the requirement of supporting each production line by only five human operators performing some remaining (i) material handling, and (ii) machine maintenance tasks that cannot be automated. In order to achieve that, manufacturing and supporting tasks must be carefully designed and scheduled, especially when it comes to the direct collaboration of humans and machinery (the remaining human factor being the rationale behind the term *semi*-automated factory).

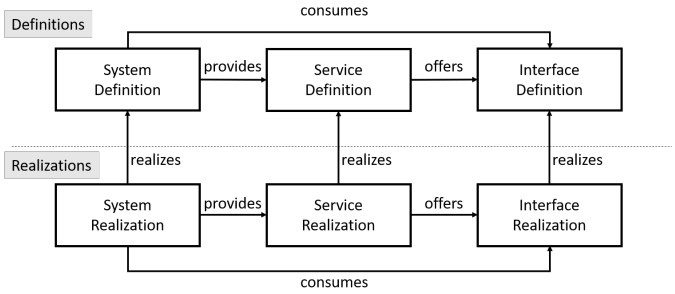

Fig. 1. Arrowhead SoS Functional Profile

III. Systems Modeling Profile for Abstract SoS Design

As introduced in Section II-B, we use high-level SysML building blocks (called *blocks* themselves) to represent the functional design of an Arrowhead SoS. Note that the nature of such a design description inherently and purposedly reflects an ideal, planning-phase view on the SoS — nevertheless, we also implement support for *updates* in the considered SoS, probably caused by operational events. Although we have been using different terms for different perspectives on the same design artifacts, in the following, in accordance with the usual terminology, we call an SoS functional design as provided in our scenario an *SoS model*.

We exploit *profiles* [6], the native language extension mechanism of UML/SysML, to define a "template" for SoS models. To be more precise, we use the notion of *stereotypes* to serve as identifiers w.r.t. the functional role of SoS blocks. As we are focusing on high-level functional elements, we take inspiration from the already established Arrowhead design principles and *SoS design description documents* in particular [7].

We identify a minimal set of basic service-oriented SoS building blocks: (i) *systems*, (ii) *services* and (iii) *interfaces*; we also introduce a separation of our SoS functional domain into *Definitions*, i.e., reusable design artifacts and *Realizations*, i.e., actual instances conforming to a definition, resulting in a total of six object stereotypes along with four link stereotypes expressing their relations, as shown in Fig. 1.

As shown in the figure, the three main concepts (system, service, interface) each have a corresponding definition as well as a realization element, indicated by the corresponding stereotype name suffixes. Relations between those "object" concepts are also expressed by stereotypes, where we employ a verb-form naming convention to emphasize their relational nature: a system *provides* a service, which in turn *offers* an interface, which might be *consumed* by other systems (cf. Sect. II-C

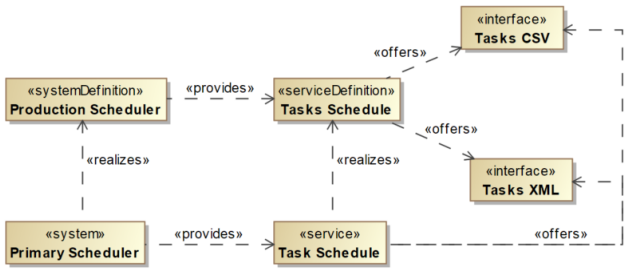

Fig. 2. Excerpt from an SoS Design of a Semi-Automated Factory

for details on the underlying architectural concepts). Note that the definition and the realization domain are perfectly aligned to each other, having the exact same structure as described above, with realizations neatly linked to the definitions by a single *realizes* stereotype. (For readability, we omit the ⟪...⟫ brackets for stereotypes as in the UML concrete syntax.)

This profile definition only provides a loose collection of the ingredients of an SoS design; although we have visualized them in an intuitive way emphasizing their interrelations, during an actual SoS modeling activity, this structure would only get revealed during the creation of the actual SoS model. During this creation process, the modeler chooses the corresponding Arrowhead SoS stereotype for each block representing an actual element of the SoS design. We illustrate both this modeling principle and how its result would look like in real life by showing a small excerpt of a fictive *semi-automated factory* (cf. Sect. II-D) on Fig. 2 (containing examples of each of our stereotypes except for *consumes*).

In this small example, we have a generic system definition for any *Production Scheduler* in the factory, which provide a *Tasks Schedule* service definition, offering a *CSV* as well as an *XML* interfaces (unifying definitions and realizations for the sake of neatness). On the realization side (bottom row), a *Primary Scheduler* (simply known as *scheduler* later) system provides a corresponding *Task Schedule* service.

Note that our semi-automated factory is completely specified and consists of multiple diagrams, also showing how the *consumes* relation is reflected in our profile and modeling principles. The complete semi-automated factory example and the profile itself, realized with MagicDraw, are available in our public GitHub project repository.[7]

IV. From Models to Operation and Back: Towards Toolchain Interoperability

Originally, there exists a principal gap between SoS design and SoS operation in Arrowhead — which we aim at diminishing through our present contribution. In particular, the main observation behind our proposal is that (i) enriching the SoS model by the few necessary configuration information and (ii) establishing a file format understood both by the modeling as well as the management framework allows for realizing a *bidirectional import/export feature* between those frameworks.

Although the concept of such a bidirectional link is generic, the current prototype implementation focuses on a particular workflow:

1) first, *a priori*, the modeling engineer creates a high-level functional design of the SoS application using the profile described above;
2) thereupon (at any design stage involving some basic operational information), the *model export* functionality can be invoked to save a textual SoS description file in JSON format;
3) then, that JSON can be *imported* to the web-based *Arrowhead Management Tool* running in a browser, resulting in an SoS instance ready to perform management operations and configuration changes as foreseen on that interface; and
4) at any evolution stage, the Management Tool is able to *export* an SoS description in the exact same format, which, if *imported* again into the original model, will result in the management changes being highlighted in a model-based view.

As for the current prototype implementation in particular, step 1 and, accordingly, also 2 are implemented in the *MagicDraw* systems modeling tool. The resulting JSON essentially reflects the same design information as our model, in a serialized textual form directly compatible with the REST API interface of the Management Tool.

Although prototypically realized for MagicDraw for now, the *pattern matching* procedure underlying the model discovery in the export step is fully general and customizable, with discovery patterns being realized in the open-source VIATRA framework,[8] being part of the Eclipse infrastructure.

As for the next step 3, the JSON produced in step 2 is imported into the Management Tool; in particular, the REST API interface of the tool extracts exactly those data pieces which are needed for successfully configuring the SoS skeleton for management. Accordingly, the JSON consists of three main blocks: (i) the list of the *systems*, (ii) the list of the *services* (called *serviceRegistryEntries* in the implementation), and (iii) *authorization rules* representing data exchange between systems derived from consumer-producer relationships (called *authRules* in the implementation). Fig. 3 shows an excerpt of a JSON generated from our semi-automated factory example. Note that the example *Robotic Arm* system appears as both provider and as consumer on different sides of services.

After performing any changes in the Management Tool (details of such operations are out of scope here), we can create a new JSON by pushing the Export button; the resulting JSON is functionally linked with the previous JSON derived from the model via custom element identifiers. This design allows for an additional feature on the modeling side (in MagicDraw): if the new JSON is *imported* again, we are able to identify and thus highlight the occurred changes in the model, while leaving the rest of the model representation untouched.

[7]https://github.com/IncQueryLabs/arrowhead-tools/tree/master/MagicDraw

[8]https://www.eclipse.org/viatra/

```
{
   "name": "Factory",
   "operator": "FactoryOperator",
   "systems": [
      {"systemName": "roboticarm"},
      ...
   ],
   "serviceRegistryEntries": [
      {
        "serviceDefinition": "
           task_schedule",
        "secure": "NOT_SECURE",
        "provider": {"systemName": "
           scheduler"},
        "interfaces": [
            "HTTP-INSECURE-CSV",
            "HTTP-INSECURE-XML"
        ]
      },
      ...
   ],
   "authRules": [
      {
        "consumers": ["scheduler"],
        "offerer": "status",
        "interface": "HTTP-INSECURE-CSV",
        "provider": "roboticarm"
      },
      {
        "consumers": ["roboticarm"],
        "offerer": "task_schedule",
        "interface": "HTTP-INSECURE-XML",
        "provider": "scheduler"
      },
      ...
   ]
}
```

Fig. 3. Excerpt of a JSON description for a Semi-Automated Factory

V. Static Validation of SoS Design

Model-based techniques are appropriate and often applied for maintaining an overview of designs and potentially also *validating* their structure according to the principles and requirements of their respective engineering domain. Accordingly, we provide such an additional feature to our contribution: a model-based support of static *validation*, i.e., the checking of well-formedness criteria over the SoS model representation.

In particular, there are some implicit requirement and structural assumptions behind our profile which we are able to make explicit through defining the corresponding *validation rules*: (i) each Realization element should have exactly one Definition which is *realized* by it, being of the corresponding

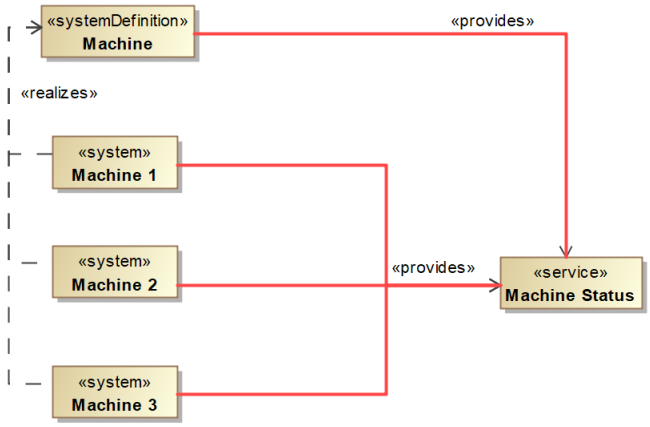

Fig. 4. Validation Example Visualization: Misplaced *provides* Relation

type (e.g., a SystemRealization is realized by a SystemDefinition etc.), and (ii) *consumes*, *provides* and *offers* relations on the Realization side has to be reflected on the Definition side between the corresponding Definitions.

Our implementation in MagicDraw, together with our VIATRA-based plugin and pattern matching engine, allows for expressing the above validation rules as structural constraints, whose occurrences are automatically, reactively checked and if they appear in the model (indicating a violation of structural constraints), they get visualized in the model itself in an intuitive red color, as shown in the screenshot in Fig. 4. In particular, the example here shows four invalid relations, caused by a misplaced *provides* relation disrespecting the conceptual layering: a SystemDefinition provides a ServiceRealization. The invalidity of this link also renders the other (well-typed) *provides* relations invalid: their SystemDefinition does not provide a corresponding ServiceDefinition. Our tooling also displays intuitive warning messages, informing the modeler about the details of the issue at hand (not shown here due to space restrictions).

VI. Related Work

Our work is conceptually rooted in the literature on service-oriented design as well as model-based systems engineering. Concerning the former, as already observed in a joint white paper of the important standardization groups OASIS, OMG and the Open Group [8], there exists a plethora of concurrent service-oriented reference models, largely overlapping in their main concepts, but still differing in their details enough to hamper unification endeavors. The SOA Source Book by the Open Group[9] proposes a layered hierarchy of (some) modeling approaches, and claims that the SOA Reference Architecture (now superseded by the ISO/IEC 16384 standard) provides the most general frame for service-oriented architecture principles. Although our functional design approach bears some resemblance to such high-level SOA reference frameworks, it has strong ties to systems engineering practice which is not

[9]https://www.opengroup.org/soa/source-book/intro/index.htm

typical for SOA references, and it has an additional distinct Arrowhead flavor.

Further modeling languages and practices are proposed and used for service-oriented architectures such as *SoaML* [9], a UML profile for SOA modeling, with no direct links to SysML though; or the high-level patterns and principles described by Michael Bell [10]. Such approaches are considered complementary to ours, which might be used in combination with an adjustable level of integration. In addition, the Open Group has published online the SOA Source Book, which can be consulted as a reference collection of diverse SOA modeling frameworks and approaches. In particular, the Open Group has defined a SOA ontology, addressing a wider knowledge context than our approach, but still allowing for an alignment. Also, besides SOA principles and focusing more on the industrial aspect of our focus field, there is a number of similar, but differently oriented ontologies for the digital industry and Industry 4.0 in particular; most notably, the RAMI 4.0 vocabulary has gained significant acceptance in the field [11]. As for the application of SOA principles in particular, this has been typically considered so far in the context of developing documenting actual applications (cf., e.g., [7]). Note that none of the mentioned approaches and knowledge representations are excluded by our modeling principles, and their integration into our high-level systems modeling approach is facilitated by our design.

Regarding the systems engineering aspect, that community has long been active in investigating ways of efficient and integrated architecture design processes for various classes of systems [12]. There exist ISO as well as OMG standards relevant here, such as ISO 15288 and 10303-233 for general systems engineering principles and data exchange in particular, respectively; also, OMG has proposed a general *Model-Driven Architecture* as well as further languages for advancing model-based integration scenarios, such as the *Unified Architecture Framework* (UAF).[10] Furthermore, INCOSE has a working group dedicated to *Digital Engineering Information Exchange*.[11]

VII. Conclusion and Future Work

The modeling of service-oriented systems-of-systems (SoS) in the context of industrial automation poses challenges for reconciling rigorous SoS design and planning principles with the highly dynamic character of SoS interoperability. In this paper, we have proposed and demonstrated a concept of *toolchain interoperability*, addressing a bidirectional interaction between design and run-time artifacts, respectively. In particular, we have (i) defined a high-level SysML profile for abstract Arrowhead SoS models (*design* aspect), (ii) implemented an export/import functionality to transfer the models into the Arrowhead management tool, i.e., actual operation and maintenance (*toolchain interoperability*) and (iii) established a validation module for checking well-formedness criteria both during design and operation, the latter via the ability to reflect run-time changes in the model (*run-time models*).

This proof-of-concept demonstration shows the potential of supporting toolchain interoperability in comprehensive engineering scenarios, even via lightweight coupling methods. The most important next step is to establish an actual technical bridge between the presented modeling approach and the real-life counterparts of the represented elements—the Arrowhead Framework offers an ideal platform to do so, integrating both the modeling and the device layer. Note that such an enhancement necessitates a careful investigation of security concerns during system interactions, which have been out of scope so far. In future work, we will also investigate how more elaborate versions of the SysML profile affect the usability and accessibility our comprehensive SoS modeling approach, and how the general-purpose JSON SoS description can be utilized in further integration scenarios with a broader scope.

[10]https://www.omg.org/mda/, https://www.omg.org/spec/UAF/About-UAF/

[11]http://www.omgwiki.org/MBSE/doku.php?id=mbse:deix

Acknowledgment

The research has received funding from the EU ECSEL JU under the H2020 Framework Programme, JU grant nr. 826452 (Arrowhead Tools project, https://www.arrowhead.eu) and from the partners' national funding authorities.

Project no. 2019-2.1.3-NEMZ_ECSEL-2019-00003 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-2.1.3-NEMZ_ECSEL funding scheme.

References

[1] M. W. Maier, "Architecting principles for systems-of-systems," *Systems Engineering: The Journal of the International Council on Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.

[2] M. Jamshidi, *System of systems engineering: innovations for the twenty-first century*. John Wiley & Sons Incorporated, 2009, vol. 58.

[3] H. P. Breivold and K. Sandström, "Internet of things for industrial automation–challenges and technical solutions," in *2015 IEEE International Conference on Data Science and Data Intensive Systems*. IEEE, 2015, pp. 532–539.

[4] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[5] J. Delsing, P. Varga, L. Ferreira, M. Albano, P. P. Pereira, J. Eliasson, O. Carlsson, and H. Derhamy, "The arrowhead framework architecture," in *IoT Automation*. CRC Press, 2017.

[6] L. Fuentes-Fernández and A. Vallecillo-Moreno, "An introduction to UML profiles," *UML and Model Engineering*, vol. 2, pp. 6–13, 2004.

[7] F. Blomstedt, L. L. Ferreira, M. Klisics, C. Chrysoulas, I. M. de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, "The Arrowhead Approach for SOA Application Development and Documentation," *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, pp. 2631–2637, 2014.

[8] H. Kreger and J. Estefan, "Navigating the SOA open standards landscape around architecture," *Joint Paper, The Open Group, OASIS, and OMG*, 2009.

[9] B. Elvesæter, C. Carrez, P. Mohagheghi, A.-J. Berre, S. G. Johnsen, and A. Solberg, "Model-driven service engineering with soaml," in *Service Engineering*. Springer, 2011, pp. 25–54.

[10] M. Bell, *SOA Modeling patterns for service-oriented discovery and analysis*. John Wiley & Sons, 2009.

[11] I. Grangel-González, L. Halilaj, G. Coskun, S. Auer, D. Collarana, and M. Hoffmeister, "Towards a semantic administrative shell for industry 4.0 components," in *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, 2016, pp. 230–237.

[12] P. Micouin, *Model Based Systems Engineering: Fundamentals and Methods*. John Wiley & Sons, 2014.