Innovations in Systems and Software Engineering
https://doi.org/10.1007/s11334-022-00455-5

S.I. : SYSTEMS MODELLING AND MANAGEMENT

SysML modeling of service-oriented system-of-systems

Jerker Delsing[1] · Géza Kulcsár[2] · Øystein Haugen[3]

Received: 5 March 2021 / Accepted: 28 March 2022

Abstract
The success of the ongoing fourth industrial revolution largely depends on our ways to cope with the novel design challenges arising from a combination of an enormous increase in process and product complexity, as well as the expected autonomy and self-organization of complex and diverse industrial hardware–software installments, often called systems-of-systems. In this paper, we employ the service-oriented architectural paradigm, as materialized in the Eclipse Arrowhead framework, to represent modern systems engineering principles and their open structural principles and, thus, relevance to flexible and adaptive systems. As for adequately capturing the structural aspect, we propose using model-based engineering techniques and, in particular, a SysML-based specialization of systems modeling. The approach is illustrated by a real-life use-case in industrial automation.

Keywords Service-oriented architectures · Industrial IoT design · System-of-Systems modeling

1 Introduction

In the new era of industrial digitalization, engineers are facing a new set of diverse challenges. Essentially, these challenges arise as a combination of (1) an increased scale and complexity in industrial engineering and automation processes, as well as in the partaking hardware and software systems and products, (2) an accompanying expectation toward those systems to organize, adapt and optimize themselves autonomously along flexible architectural principles, and, finally, (3) a desire to maintain or even strengthen the use of rigorous design practices and standards while devising these complex systems and processes, thus, retaining a comprehensive overview of the components of the overall industrial scenario, regardless of their individual behavior and their modes of interaction. Notice that the crux of this novel kind of challenges lies within the contradictory nature of those wishes and goals, in short, (1) complexity, (2) autonomy, and (3) modelability. (In particular, (3) appears to be in opposition to the first two.) It is now crucial to achieve a thorough understanding of the interplay of these conceptual points: the success of the ongoing fourth industrial revolution, often called *Industry 4.0*, depends on finding the right ways to harmonize the aforementioned goals.

The present paper proposes an approach to live up to these challenges by identifying an adequate conceptual domain for each of the three goals, and then examining their combination in the light of the state-of-the-art in each of those domains, as well as a current, real-life industrial case study.

As for *complexity* of industrial installments, we refer to the more and more prevalent notion of *System-of-Systems* (SoS) [1–3], denoting complex software and hardware (i.e., often, cyber-physical) systems, as well as their large-scale, dynamic compositions. In addition, not unrelated to the layered, diverse and inter-domain nature of SoS, instantiations of this concept are expected to exhibit a level of *autonomy* and self-organization not seen before. Recently, a structuring and communication concept called *service-oriented architecture* (SOA) has drawn much attraction, representing a promising compromise between tractable, well-founded design principles and the flexibility and adaptability of the (systems of) systems emerging from SOA design processes.

As for *modellability*, the complex and critical nature of systems engineering activities still necessitates a well-

✉ Jerker Delsing
jerker.delsing@ltu.se

Géza Kulcsár
geza.kulcsar@incquerylabs.com

Øystein Haugen
oystein.haugen@hiof.no

1 Luleå University of Technology, Luleå, Sweden

2 IncQuery Labs cPlc., Budapest, Hungary

3 Østfold University College, Halden, Norway

Published online: 09 May 2022

founded methodology for the design of complex systems and SoS, from understanding their requirements to deploying and maintaining them according to design-time expectations.

A go-to methodology for designing and maintaining complex systems, model-based systems engineering (MBSE) [4] offers a comprehensive yet practical foundation for such activities. However, a number of open questions remain around the application of MBSE techniques and, in particular, the SysML modeling language as a conceptual basis, in Industry 4.0 scenarios involving industrial SoS applications. Furthermore, the engineering of SoS-based solution architectures is currently largely based on experience, tools and toolchains primarily adopted for legacy architectures. Thus, we perceive an enormous potential for cross-fertilization between SoS engineering and MBSE.

Summarizing, the present paper offers a crossover approach: *we present an MBSE approach to design SOA-based SoS*.

Specifically, we rely on the standard language of MBSE, the *systems modeling language* (SysML) [5,6] maintained by the Object Management Group (OMG), which is intensively used for large-scale, comprehensive industrial systems modeling activities. In this paper, we address both the current, industrially established SysML version (v1.6, Sects. 3 and 4) and the upcoming new major version, SysML v2, constituting a fundamental change in the internal and external structure of SysML (Sect. 5).

As for the technological frame of our work, we build upon the concepts of *Eclipse Arrowhead*, an SOA-based industrial IoT framework, also considering its reference implementation.[1]

In this paper, we report the following specific contributions:

- we propose a *SysML v1.6 profile* for modeling service-oriented SoS, inspired by Arrowhead design principle but being generic in nature (Sect. 3);
- we demonstrate how that profile is utilized to create a full-fledged model library of the so-called core systems of the arrowhead architecture (Sect. 4);
- we describe how our arrowhead-related endeavors serve as a major SOA validation case study in the development of SysML v2 (Sect. 5);
- additionally, a real-life, comprehensive industrial use case is presented in both SysML v1.6 and v2 fashion (Sect. 6); and
- we discuss to what extent our profile captures the standard, generally accepted principles of the two underlying conceptual frameworks, namely SOA and SoS (Sect. 7).

A SysML profile for SOA-based SoS, inspired by the Eclipse Arrowhead base architecture, has been devised. This profile is then used to devise a comprehensive model library of the Eclipse Arrowhead core system. This development is now impacting the SysML v2 standardization. As a first verification, an industrial case is been modeled using both versions of SysML.

[1] https://github.com/eclipse-arrowhead

2 Background

In this section, we provide an overview of the relevant technologies and concept models related to the industrial use of service-oriented architectures (SOA) from a more technical perspective, being the conceptual backbone of our proposal and its industrial utilization through the Eclipse Arrowhead framework.

In the following, we provide a brief overview (including related work) on SOA, solution architectures and concepts, and their engineering methods.

2.1 Service-oriented architectures

SOA, introduced by IBM in the mid 1990s [7], is often referred to as micro-service architecture. From a more fundamental perspective, these terms are considered equivalent [8]. In this context, a micro-service is produced/consumed by a micro-system. A SOA reference model has been published by OASIS [9]. The key properties of SOA/micro-service architectures are:

- Look-up
- Late binding
- Loose coupling

often referred to as the 3 Ls.

Further, important properties are as follows:

- A micro-system performs its function independently.
- A micro-system can be
  - stateful and is then responsible for storing its own state; or
  - stateless.

Fulfillment of these properties requires a "service registry" and an "orchestration capability." A prominent early usage of these fundamentals was the Jini add-on package to Java [10]. Unfortunately, this approach did not experience wide acceptance.

SOA is currently the main approach for achieving automation and digitalization architectures and their reference implementations and platforms for concrete solution implementation. Examples include FiWare [11], Eclipse BaSyx [12], and Eclipse Arrowhead [13,14]. A comparison of the

current most relevant initiatives related to automation and digitalization can be found in [15].

For the implementation of SOA, there is a wide range of protocols that can be considered service protocols, e.g., HTTP (REST), CoAP, MQTT, DDS, and AMQP [16].

2.2 Solution architectures and concepts

The design and development of complex industrial automation and digitalization solutions is currently based on older, structured and more rigid architectures, such as the ISA-95 standard [17]. With the emergence of Industry 4.0, the demand for and complexity of such solutions is expected to grow substantially. Accordingly, there is an increasing need for approaches to reduce costs and design and engineering time for such solutions. Here, appropriate implementation architectures, reference platforms and implementation frameworks, and associated tools and toolchains become highly important.

SOA is a major approach to implement automation and digitalization solutions following Industry 4.0 architectures such as RAMI4.0 [18] and IIRA [19]. Their implementations are often achieved using an SOA/micro-service approach and principles, properties and terminologies based on SoS and the IoT. Some SoS fundamentals have been formulated by Maier [1,2] and then updated and extended by Boardman and Sauser [3]. A further extension to the biology of systems has been proposed by Sauser et al. [20]. Therefore, the SoS properties that must be captured in modeling can be summarized as follows:

- *Operational independence/autonomy of the elements* The constituent systems can operate independently in a meaningful way and are useful in their own right.
- *Belonging* The autonomous constituent systems choose to belong to the SoS because they see value for themselves to give up some of their autonomy to receive benefits.
- *Connectivity* To let the constituent systems interact, they must be connected, and unless they provide sufficiently generic interfaces, they must be modified to provide such interoperability. Connectivity in an SoS is, thus, dynamic, with interfaces and links forming and vanishing as the need arises.
- *Diversity* Whereas many other systems strive to minimize diversity to simplify the system, increased diversity in an SoS gives it the ability to better address unforeseen situations during its life-cycle.
- *Emergent behavior* appears in any system, and in many systems, this is deliberately and intentionally designed in and tested. In an SoS, emergent behavior is not restricted to what can be foreseen. Instead, the system should have the capability to detect (in an early stage) and eliminate bad behavior that emerges.
- *Managerial independence of the elements* The constituent systems not only can but do operate independently, even while being part of the SoS. Further, elements can be added individually to the SoS.
- *Evolutionary development* The SoS does not appear fully formed, and functions and purposes are added based on experience.
- *Geographical distribution* The constituent systems only exchange information and not substantial quantities of mass or energy.
- *Secure and safe* Malicious behavior in an SoS and its constituent systems must be detected and mitigated to ensure information, system and SoS integrity.

2.3 Solution engineering

The knowledge and experience of complex automation engineering is currently guided by a number of engineering standards, e.g., ISO 81346 and ISO 18828 [21,22].

In the transition from legacy production architecture, such as ISA-95 [23], to Industry 4.0 architectures, such as RAMI4.0 and IIRA, SOA and micro-service based implementation architectures and platforms were pioneered by Colombo and Karnouskos [24–26]. Properties, principles and concepts formulated around SoS have become necessary to consider in the engineering of Industry 4.0 automation solutions.

In this context, we need to reason about capturing SoS fundamentals in an MBSE context; thus far, only a limited amount of work has been conducted in this area. In this respect, a UML-based SOA profile, *SoaML*, was proposed in 2009 [27]. A number of the fundamental properties and keywords of SOA were not included in the SoaML profile; thus, wide adoption did not occur.

A few SOA modeling approaches were devised in the early 2000s. The service-oriented design and development methodology (SDDM) [28,29] provides methods and techniques used in service-oriented design and approaches the service development methodology from the perspective of both service producers and consumers. In 2004, IBM announced service-oriented modeling and architecture (SOMA) as its SOA-related methodology. Further details of this approach have been published by Arsanjani [30]. The service-oriented modeling framework, SOMF, was devised by Michael Bell as a holistic and anthropomorphic modeling language for software development that employs discipline and a universal language to provide tactical and strategic solutions to enterprise problems [31].

A recent paper by Urgese et al. [32] provides an automation solution engineering approach that extends the IEC 81346 standard. The approach models both the engineering process and the engineering tool usage in an SOA fashion.

Moreover, the approach can be modeled directly using the SysML approach proposed in this paper.

Of the SoS properties listed above, operational independence, belonging, connectivity, diversity, secure and safe, and geographical distribution are potentially captured using modeling languages such as SysML. This paper discusses the SysML v1.6 and v2 approaches, enabling solution modeling of industrial automation and digitalization based on a SOA approach, where several fundamental SoS properties are addressed.

3 Concepts for system-of-systems modeling

In this section, we introduce a conceptual space for modeling complex, hierarchical SoS according to service-oriented principles, as discussed above. More concretely, we are proposing a SysML v1.6 profile for SoS modeling. (The modeling tool used for the current implementation of the profile is Cameo Systems Modeler.[2])

The profile is inspired by and validated against the principles proposed in the Eclipse Arrowhead framework. Eclipse Arrowhead provides an SoS architecture approach and a reference implementation platform. The concepts we identify form a general conceptual framework and are intended to be used as reference points for modeling large-scale, dynamic entities in industrial IoT.

The notion of a *profile* has already been proposed in the UML language [33] and serves as its central language extension and specialization mechanism [34]. In turn, SysML has been conceived as a profile of UML. A profile is a collection of *stereotypes*, abstract element descriptions, which can be used for fixing the modeling vocabulary for a given (emerging) modeling domain.

Our profile is conceived to support the creation of SOA-based SoS solutions making use of the Eclipse Arrowhead framework and its reference implementation. The profile also makes use of and extends the Eclipse Arrowhead documentation structure [35].

Thus, the profile addresses the following design levels, constituting black- and white-box descriptions of SoS elements. As a convention, those concepts representing a black-box aspect have a *Description* suffix in their names, while their white-box counterparts end with *Design Description*.

Specifically, the profile captures the following layers of an SoS:

- service and interfaces,
- systems with producing and/or consuming services, and
- two different abstractions for compound SoS:
  - *Local clouds* [36] as basic organizational units and
  - *Systems of local clouds*, i.e., their compositions.

The profile also includes a basic form of design support regarding data exchange properties. The current version of the profile is available in the GitHub repository of the project.

Following the principles of SOAs, the main ingredients of our conceptual space are *services* and *systems*. Therefore, we proceed with first introducing stereotypes used to represent services (Sect. 3.1 and then continue with systems stereotypes (Sect. 3.2). Next, we model the concept of local clouds [36] (LCs) and systems of LCs (SoLCs), originally proposed within the Arrowhead ecosystem. The present section concerns the concepts themselves; for their use, we refer to Sect. 4, with a detailed description of the Arrowhead SysML library based on the profile.

3.1 Service stereotypes

Figure 1 shows the three stereotypes (`SD`, `SD-DD` and `IDD`) used for service modeling in our approach, including their connection to the SysML language. (The figure comes from an actual implementation of the profile in Cameo Systems Modeler.)

The connection to SysML is represented by the box in the upper left, representing the `Interface Block`, one of the central meta-classes of SysML. An interface block, as the name suggests, is typically applied for entities representing communication or exchange interfaces between other entities (systems in our case). Therefore, it is a perfect fit for expressing service-related concepts in a service-oriented SoS model.

We distinguish between two abstraction levels: `SD` (*Service Description*) is the abstract unifying concept for *services*, as generally understood according to the SOA paradigm (cf. Sect. 2.1). In our setting, an SD is a general description of some functionality or data service (e.g., temperature measurements), which can be either *provided* or *consumed* by systems.

While an `SD` represents a black-box perspective on services, its `SD-DD` (*Service Description–Design Description*) counterpart captures the realization of a service as it is actually provided (or consumed) by a system.

The actual communication then happens via *interfaces*, which are represented as `IDD`s (*Interface Design Descriptions*), i.e., refinements (*specializations* in UML terminology) of `SD`s, also providing information on communication details, allowing actual communication to occur between a provider-consumer pair of systems. Here, the applied security, usage of encryption, compression, encoding, etc., are defined, along with the applied payload data model. Instances of *IDD*s would typically appear as contained in an `SD-DD`

[2] https://www.nomagic.com/products/cameo-systems-modeler.

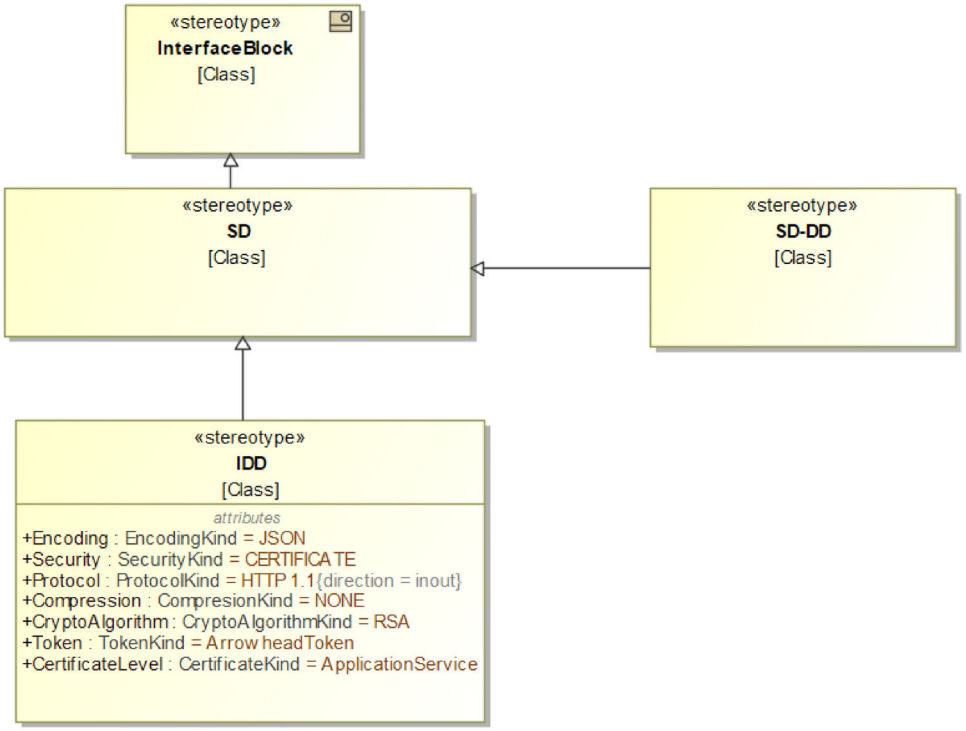

Fig. 1 Stereotypes for describing services

and be interpreted as actual interface endpoints belonging to single operations of an overall service definition.

3.2 Systems stereotypes

Figure 2 shows an overview of the rest of the concepts presented in this paper, namely stereotypes concerned with *systems* and compound entities based on them, *local clouds* as well as *systems of local clouds*.

First, note that the aforementioned concepts also constitute a three-layer composition hierarchy, as represented by the composition links (the black diamond-headed lines) between them from the top to the bottom. As for the horizontal classification, the organization of these concepts, again, follows our principle of providing black-/white-box pairs for each composition layer. As shown in the upper-left corner, each system stereotype is a specialization of `Block` coming from the SysML language, being its central, general stereotype.

In the remainder of this section, we describe the system stereotypes, i.e., the elements in the bottom layer of Fig. 2.

The `SysD` (*System Design*) stereotype represents a black-box view of systems, i.e., it can be considered a generic template for (the software behavior of) a given system type in the SoS that we want to model.

In turn, the `SysDD` (*System Design Description*) stereotype represents a white-box view of systems, resulting in higher precision regarding the details of the behavior. Note, however, that these system stereotypes can be used hierarchically; i.e., there are no strict constraints on how much information has to be provided in a single `SysDD`, and the amount of information can be refined in further specialization blocks.

A very important, even central, usage of `SysD` and `SysDD` blocks is to hold *ports* (which are, again, basic SysML elements), i.e., connections with the outer world. However, in SysML, the exact interpretation of a port is left to the modeler. The interpretation in our profile is that a so-called full port is typed over an `SD`, representing an overall service provided or consumed by that system, while the individual interfaces of that service are modeled by `IDD`-typed proxy ports contained in that full port.

3.3 Stereotypes for local clouds and systems of local clouds

In this section, we elaborate on the four stereotypes remaining in Fig. 2: the pair consisting of `LocalCloudDesign` and `LocalCloudDesignDescription` (middle layer), as well as the `System-of-LocalClouds-Design` stereotype, together with the `System-of-LocalClouds-DesignDescription` (upper layer).

The concept of a *local cloud* [36] has been proposed in the context of Arrowhead; however, it constitutes a general-purpose organizational principle for SOA-based SoS. A local cloud is an SoS instance, where the partaking systems have a certain level of trust toward each other, enabling a more direct orchestration mechanism between them, while at the same

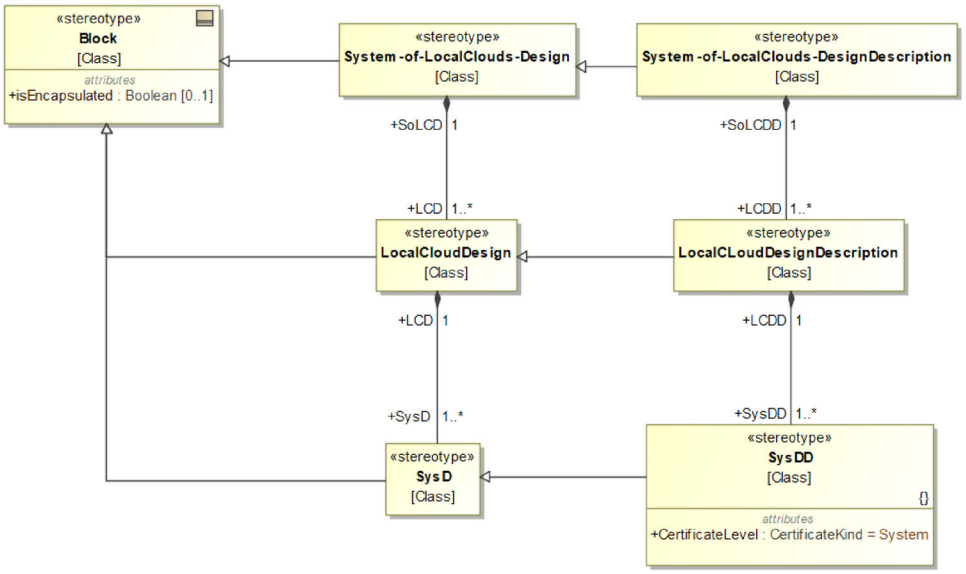

Fig. 2 Stereotypes for describing systems and (systems of) local clouds

time, the local cloud also implies a logical grouping of its constituents. (For a concrete example, cf. Sect. 6.) As expected, the `LocalCloudDesign` stereotype corresponds to other *Design* stereotypes and serves as their logical container, while the `LocalCloudDesignDescription` is a more detailed entity with a white-box view on design details, as represented by other *Design Description* stereotypes within.

Finally, the most large-scale entities in our modeling domain are the *Systems of Local Clouds*. Such dimensions and scenarios are the final goal of industry-scale IoT endeavors: the envisioned level of automation and autonomy is achievable only by a coordinated, yet adaptive interaction on a global scale. In our setting, this is expressed by the SoLC level: an SoLC is *also* an SoS instance, having a compound, more global nature. The communication between the clouds has to follow strict rules; therefore, an SoLC has a different operational nature than that of a single local cloud, despite the structural analogy. As expected, the two different stereotypes associated with systems of local clouds serve as high-level containers for the respective black-box and white-box designs of their constituents.

4 Arrowhead core system library

The profile described above has been used to create *library* models of a number of the core systems of the Eclipse Arrowhead framework.

An example of such a core system model is the Eclipse Arrowhead ServiceRegistry system. The model is shown in Fig. 3. The currently available core system models for enabling SOA solution architecting and implementation are as follows:

- ServiceRegistry system
- Orchestration system
- Authorization system
- SystemRegistry system
- DeviceRegistry system
- Onboarding system
- CertificateAuthority system
- DataManager system
- Translation system
- EventHandler system
- QoS (Quality of Service) system
- TimeManager
- GateKeepeer system (intercloud Lookup, Late binding and Authorization)
- Gateway system (intercloud secure service exchange)
- OPC-UA adaptor
- Z-wave adaptor
- MQTT broker
- LocalCloud
- System-of-LocalClouds
- Generic-application system
- Device; capable of hosting multiple systems with physical network port/s
- Network; defining the physical network layers used by Device network ports

The profile and the library enable us to design and model complex automation and digitalization solutions. A generic example of a modeled SoS use case is shown in Fig. 4. Here, a single local cloud is modeled with the Eclipse Arrowhead core system. A protected solution is created within one local cloud with SOA lookup, late binding and loose coupling properties, as implemented by the Arrowhead ServiceRegistry and Orchestration systems. Authentication and autho-

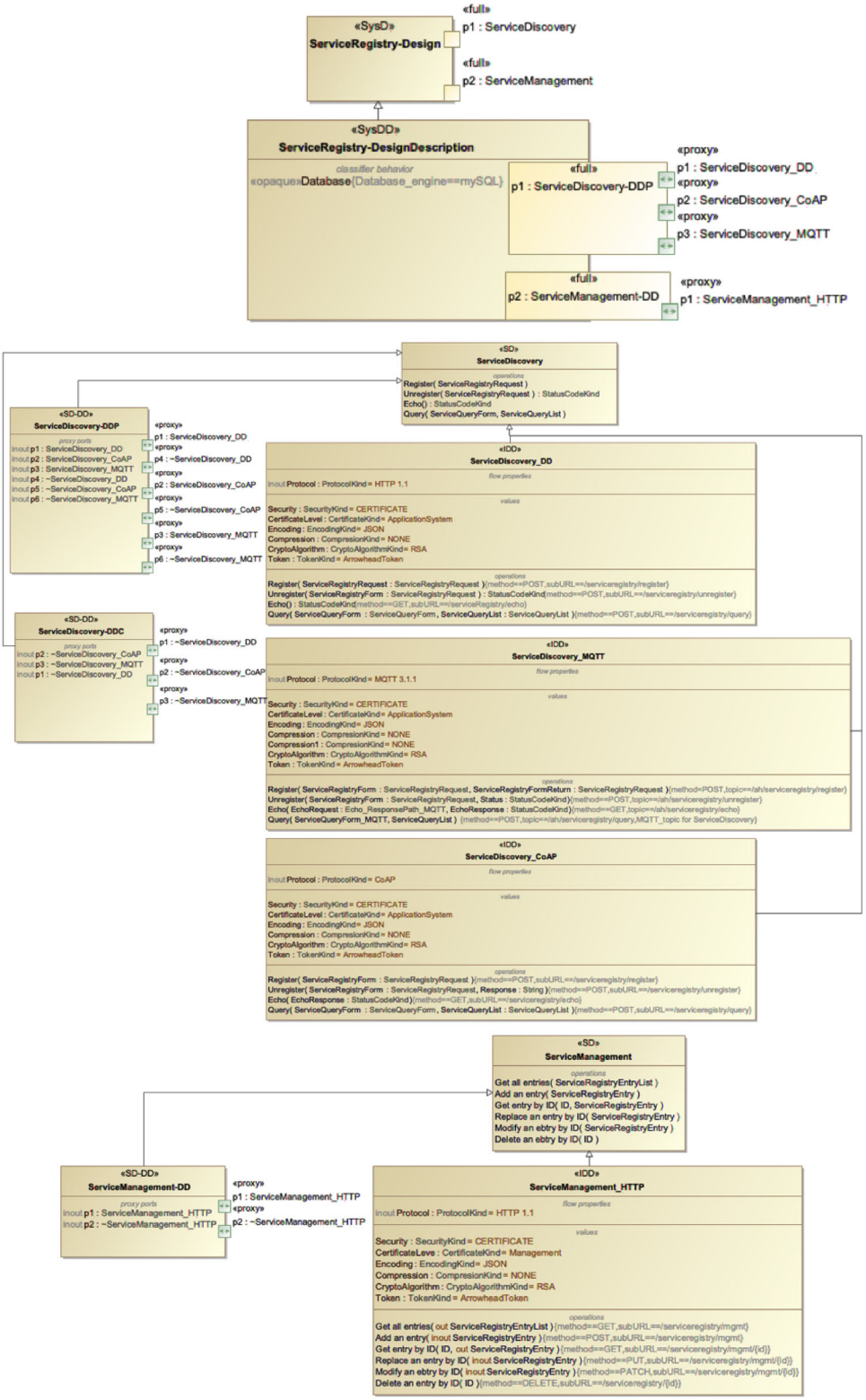

Fig. 3 SysML model of the eclipse arrowhead ServiceRegistry system

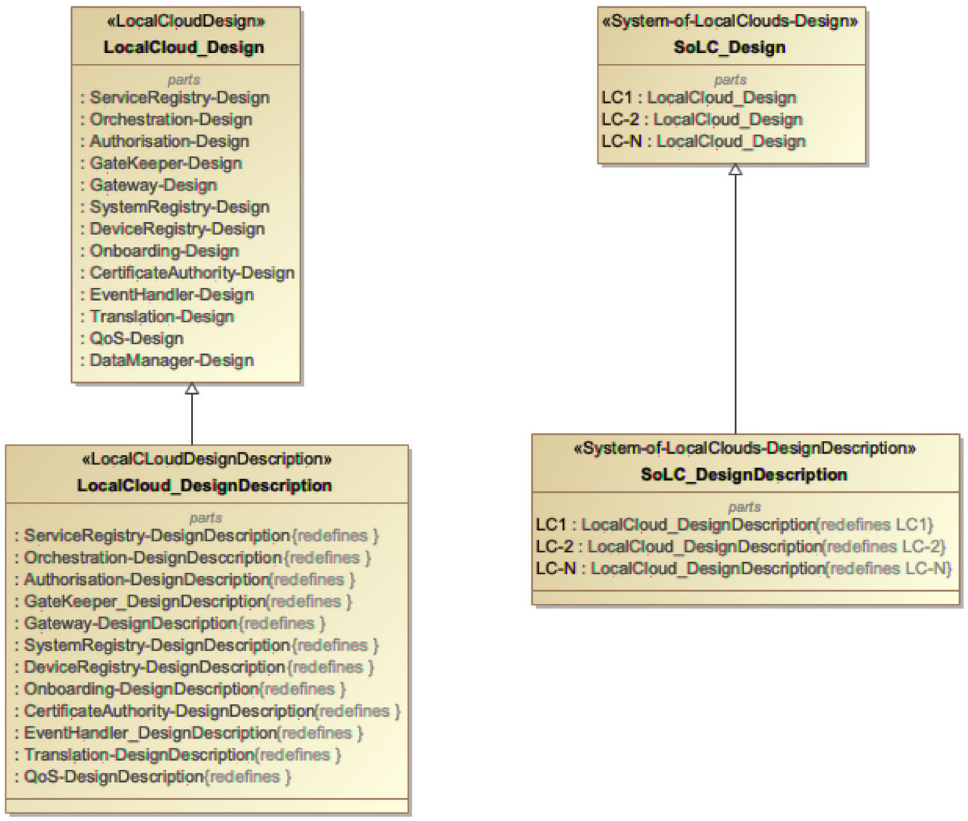

Fig. 4 Left: the SysML model of a basic SoS modeled as one local cloud including a number of the eclipse arrowhead core systems. Right: a system of local clouds (SoLC)

rization security is implemented through the Arrowhead Authorization system. The required automation functionality can then be modeled as a set of application systems and their interaction with the available core systems. Application system modeling is supported through a generic application system template. Solutions using multiple local clouds are also modeled and shown in Fig. 4.

The current version of the library is available from the GitHub repository of Eclipse Arrowhead https://www.github.com/eclipse-arrowhead

The validation of the profile and the library is based on a concrete industrial automation use case, described and modeled in Sect. 6.

5 Standardization: toward next-generation systems modeling

5.1 Introducing SysML v2

SysML v2 is the emerging next version of SysML. SysML was introduced in 2005, and it seems appropriate that a successor would appear to account for the technological developments in the past 15 years. The standardization of SysML is performed in the context of OMG (Object Management Group) that also provides a number of other model-oriented technologies, such as UML [33]. The process is open in the sense that there are no restrictions on becoming a member of the OMG, but the process is not public since until the technology is available, the documents may be kept for those members that participate in related consortia. For SysML v2, however, the large consortium (and there is only one consortium in this case) has decided to make the evolving pilot implementation with documentation open source. This has made the SysML v2 process more transparent to the general public [37].

The pilot implementation is important since it represents an opportunity to experiment and a solid basis for discussion. Most discussions occur in subgroups of the large consortium, which now includes more than a hundred individual participants and close to a hundred organizations.

The development of SysML v2 has been a well-managed process in which the requirements of the new version of the language were collected and systematized in a request-for-proposals. The satisfaction of these requirements is meticulously monitored by applying the pilot implementation.

However, there were few requirements in the field of SOA. To take advantage of the new SysML v2 to support Arrowhead, we would need proper constructs to describe SOA; therefore, we proposed to provide a validation use-case in the field of SOA based on the Norwegian industrial use case

in the project Productive4.0 [38], where communication is handled by Arrowhead. This scenario has turned out to be fruitful for several discussions since the example is manageable and originates from a real case in industry.

5.2 The descriptive needs of the arrowhead framework

Taking the SysML Arrowhead profile as starting point and the Norwegian Productive4.0 use-case as an example [39], we investigated how SOA could be described with the emerging SysML v2. We found some issues with describing SOA in general since there have not been many requirements originally recorded in that field, and we found that when drilling down into the details, there were other issues related more directly to Arrowhead.

- Representing a remote method call where the requiring party waits for the provided service to be performed and results to be returned.
- Sufficient information described for asynchronous signal sending and reception, potentially through a subscribe/publish approach.
- Visualization of SOA protocols with sequence diagrams where the detailed communication points appear.

These issues may at first glance appear obvious and to be topics that have been included in earlier versions of SysML version 1 and UML version 2, but this is only partially correct. SysML v2 is constructed from scratch on a foundation (KerML) that provides more precision with formal semantics. Thus, the language is not simply an enhancement of SysML v1 even though it should cover the same purposes and more. Therefore, our Arrowhead requirements must be documented with a proper rationale, such that Arrowhead and its use cases are credible grounds for discussion.

Moreover, even though operation invocation is included in UML 2, it has not been straightforward to apply operations. Built on UML 2, the SoaML profile never achieved a large audience. Therefore, our aim with SysML v2 is to cover SOA concepts in a straightforward manner and not fall into the pitfalls of the past attempts.

In September 2021 SysML v2 presented their "Revised Submission" for the OMG Technical meeting. Even though our Arrowhead needs have been on the agenda for over a year, the language still has no proper support for remote method calls. The asynchronous communication is taken well-care of, and the remote procedures can be modeled through signaling, but we want more native concepts. Hopefully, this will be doable before the final submission intended for early 2022. Basic sequence diagrams are in place, and the major improvement is that the sequences will be based on the same semantic base as the other language concepts of SysML v2. In UML 2.0 sequences, activities and states had different semantic definitions and consistency was hard to prove.

5.3 The arrowhead language extensions for SysML

SysML v2 may not receive the same kind of extension constructs as we know from UML profiles, as given in Sect. 3, but rather promises a stronger language extension mechanism inspired by how SysML v2 is built on the KerML kernel language. The big advantage is in the potential to establish a domain-specific language (DSL) based upon the SysML v2 constructs and tooling, in addition to defining the proprietary language constructs' semantics in terms of KerML and SysML libraries. We believe that Arrowhead is the perfect candidate to implement such a language extension mechanism, and we have already performed an experimental simulation of how this could look and be defined.

5.3.1 Language extension for arrowhead in SysML v1

In UML and SysML, we apply "profiles" that define extensions of the UML or SysML basic concepts. Typically, such "stereotypes" are given meaning through code generation by the community that supports and applies the defined profile. We have defined a SysML v1 profile for Arrowhead, as shown in Figs. 1 and 2. The profile defines the basic concepts of Arrowhead, and their semantics are given by the explanation found in the Arrowhead documentation. The implementation of the profile is given by the tool providers that support the profile.

Eclipse Arrowhead also contains Arrowhead systems and services that are already developed by the open-source community, and there is a SysML v1 description of these systems and services in the form of a model library based on the profile. Excerpts of our core library are shown in Fig. 3.

5.3.2 Language extension for arrowhead in SysML v2

In SysML v2, the precise semantics refer to a basic library of concepts of execution defined by the new language KerML. Thus, the whole language semantics are built on a core library, and extensions of the concepts are defined by specializing the core concepts and redefining and subsetting the properties. This technique is well-known in object orientation but has not previously been applied as systematically as in SysML v2 [40].

We also apply this technique to tailor Arrowhead to SysML v2, and the Arrowhead profile becomes an Arrowhead Profile Library, as shown in Fig. 5.

The Arrowhead Core library is built upon the profile library and corresponds very closely to SysML v1 description of the core systems. In Fig. 6, we focus on the Service Registry, as we did in Fig. 3. We have added some colors to

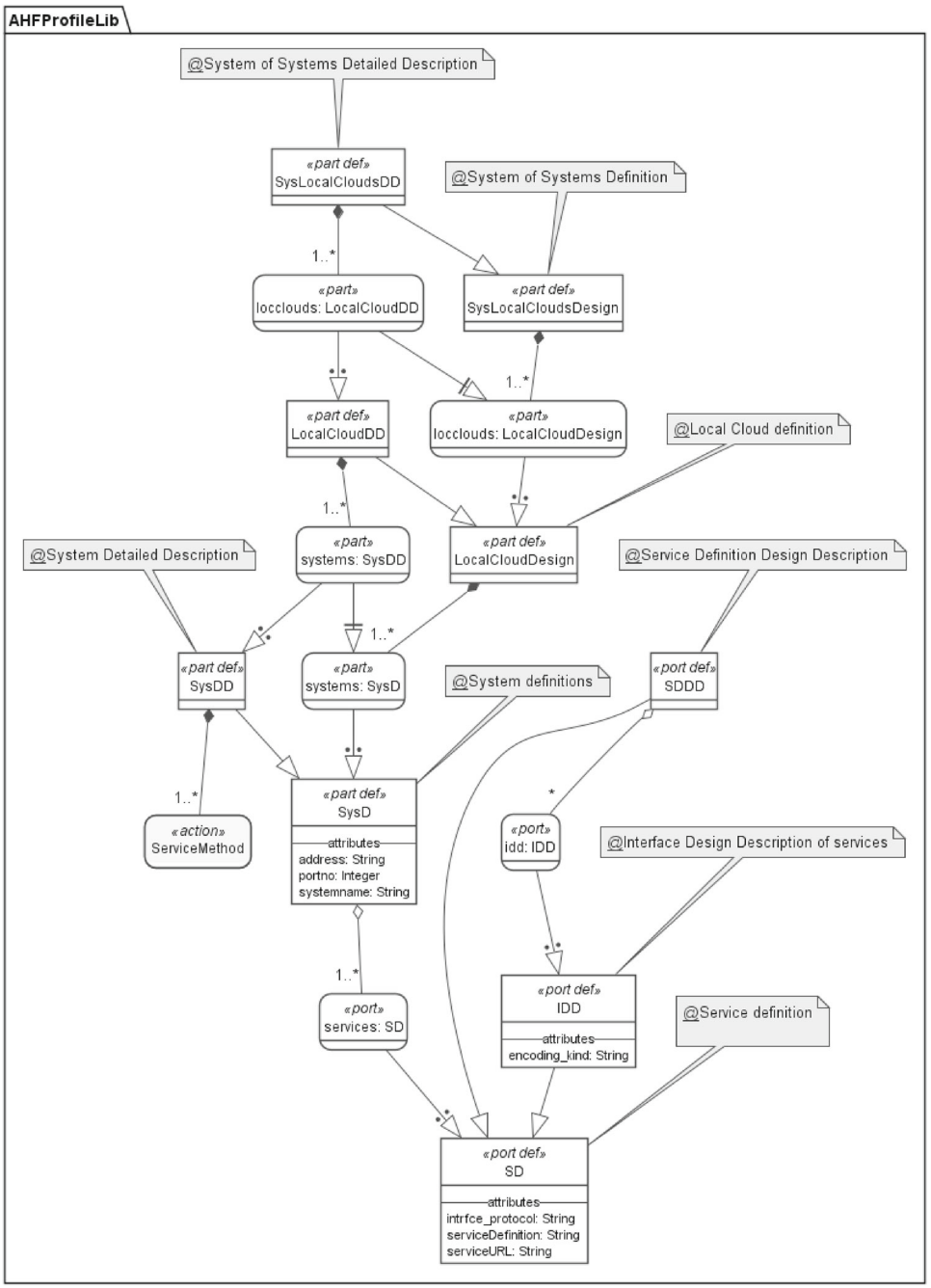

Fig. 5 SysML v2 arrowhead profile library

show how the Arrowhead core concepts are inherited from the profile library. The service_registry (light blue) is the Arrowhead system of type SysD concept of the profile library and subsetting the set of systems defined within the LocalCloudDesign.

The service_registry defines its service serviceDiscovery (green) as a port of type ServiceDiscovery, which is inherited from library concept SD, and is a subset of the services of the SysD concept of the profile library.

This shows the design level of the core system service_registry. Now we apply the same mechanisms of inheritance, subsetting, redefinition and typing to specialize the detailed description from the design descriptions.

The service_registryDD (darker blue) is subset from service_registry, and it redefines the serviceDiscovery port to be of the type ServiceDiscoveryDD that is inherited from ServiceDiscovery, and profile type SysDD. More detail is added by a couple of nested ports representing the different protocols that can be used to communicate with this service. These nested ports (yellow) subset the idd (typed by IDD) defined within SDDD.

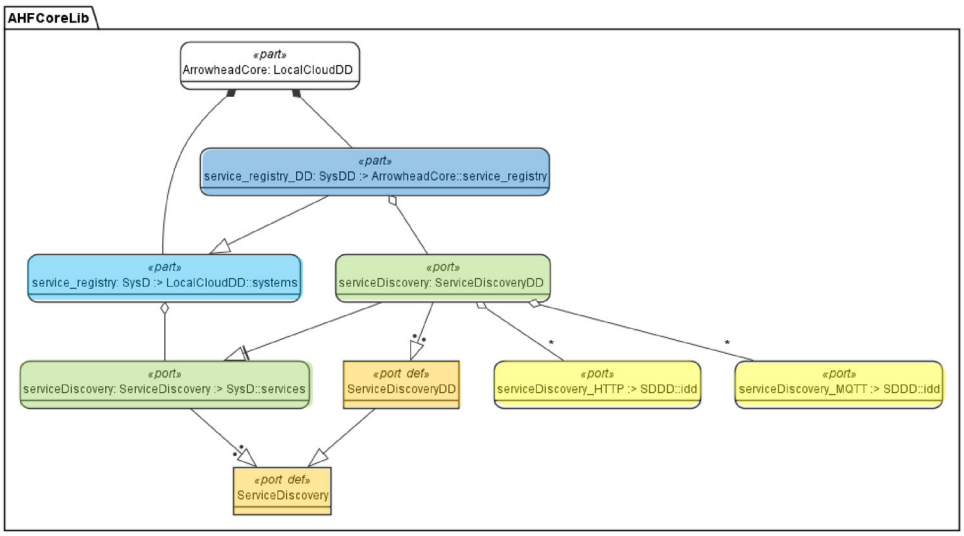

Fig. 6 SysML v2 arrowhead core library (excerpts)

5.3.3 An arrowhead domain-specific language based on SysML v2

Even though the principles of specialization in SysML v2 originates from object-orientation, their application may still be slightly complicated for the creator of Arrowhead systems-of-systems. This is why we apply the profile terms in SysML v1, but SysML v2 will have an even better way to define language extensions using the library concepts and the general specialization mechanisms.

In Fig. 7, we show an excerpt of the SysML v2 definition of the core system library corresponding to Fig. 6 plus a mock-up of what that definition could look like with the generalized language extension.

We shall go through the principles by referring to the example in detail. On the very first line, we have "#service def ServiceDiscovery" in our Arrowhead notation corresponding directly to "port def ServiceDiscovery :>SD" in the plain SysML v2. This example shows how we explain the #service concept by specializing the "SD" port definition in our Arrowhead library. This is the exact same approach as that used by the whole SysML v2 language to define its semantics and is therefore possible to accommodate without unwanted side effects. Similarly, subsetting is applied for usages, as can be seen further down in the line "#local-cloud ArrowheadCore" which is a usage of an Arrowhead local cloud containing a set of systems, and usages of systems are defined by "#system" keywords, and the inheriting of the SysD library concept is included in the definition of the system concept.

The SysML v2 concepts used for specialization and for all other needs can be used without restrictions.

This simple mechanism may seem like "syntactic sugar" and it can be viewed in that way, but one should not underestimate the usefulness of directly applying terms of the domain. We may also include constraints in the definitions of the #-keywords restricting the concepts even more. Moreover, ongoing work is investigating how this domain-specific notation can be visualized via domain-specific graphics similar to our Arrowhead notation shown in Fig. 8. In this way, the Arrowhead domain and use of service-oriented architecture becomes visually intuitive, corresponds directly to the Arrowhead methodology and is defined precisely through the same principles as the SysML v2 language itself.

We contribute to standardization to ensure that the modeling means of tomorrow fit our Arrowhead framework. Through active participation, some of our proposals will be included in SysML v2, but we cannot expect all our desires to be satisfied in the first version of SysML v2 for several reasons. SysML v2 has many requirements, and our focus is not the same as everyone else's. SysML v2 will be used for a variety of systems that will not apply SOA. Therefore, some Arrowhead concepts may not be possible to include, and they may also be difficult to express with the extension mechanisms indicated in this section.

6 Chemical factory validation use-case

In Productive4.0 [38], we used Eclipse Arrowhead as the vehicle for integrating the chemical use-case performed by the Norwegian consortium at Unger Fabrikker[3]. The chemical use case focused at first on reducing the time between products and thereby contributing to both less waste and higher productivity. The chemical production goes on continuously 24/7, but in this chemical plant, they make several specialized niche products and there is a need to roll over from one product into another one. Between products there

[3] https://www.unger.no

Fig. 7 Services in arrowhead DSL compared with plain SysML v2

```
#service def ServiceDiscovery;

#servicedetail def ServiceDiscoveryDD :> ServiceDiscovery;

#localcloud ArrowheadCore{
  #system service_registry{
    #service serviceDiscovery : ServiceDiscovery;
  }

  #systemdetail service_registry_DD :> service_registry{
    #servicedetail :>> serviceDiscovery:ServiceDiscoveryDD {
      #idd serviceDiscovery_HTTP;
      #idd serviceDiscovery_MQTT;
    }
  }
}
```

```
port def ServiceDiscovery :> SD;

port def ServiceDiscoveryDD :> ServiceDiscovery, SDDD;

part ArrowheadCore : LocalCloudDD{
  part service_registry : SysD :> systems {
    port serviceDiscovery : ServiceDiscovery :> services;
  }

  part service_registry_DD:SysDD :> service_registry{
    port :>> serviceDiscovery:ServiceDiscoveryDD {
      port serviceDiscovery_HTTP :> idd;
      port serviceDiscovery_MQTT :> idd;
    }
  }
}
```

is a phase where no actual product is produced and such production needs to be handled as waste. By applying machine-learning and a new non-intrusive NIR (Near Infra Red) sensor, we were able to make predictions such that the process could be tuned quicker and waste reduced.

During the project we also introduced two other use cases associated with the chemical production. On use case concentrated on improving the supply deliveries of lye and sulfur by providing the suppliers with more online information about the states of the tanks and what was expected to be used. From this, the suppliers could themselves decide when they wanted to deliver the raw material and thus improving their own logistics as well as taking some work off the purchasing staff at the chemical plant.

Finally, we also did some experiments on the product deliveries to investigate what could be the source of accidental lumping of a powder product when transported over longer distances in Europe by truck. We instrumented the big bags of powder with sensors for acceleration, temperature, humidity and location and could follow and store these data online.

Supporting all of these three use cases of the chemical plant was a data communication structure that involved several different platforms, different hardware, and the need for physical and logical firewalls with security. The sensor data from the NIR sensor was collected by the APIS system from the partner and sensor vendor Prediktor[4] and the data from the sensors in the big bags of powder was collected via the TellUCloud gateways and software by the partner TellU[5] before this was also placed in the APIS system before communicated to the chemical plant hub.

The unifying framework for this communication between these different platforms was the Eclipse Arrowhead. We added adaptors and analysis applications to the framework, and also took advantage of the built-in registration, orchestration and security properties. For our presentation in this paper, we have in in Fig. 8 simplified the system to highlight the unifying principles of the Eclipse Arrowhead.The real use-case is of course more elaborate [39], but the simplification demonstrate the main issues that we cover in this paper. We show the asynchronous communication between the APIS Producer and the APIS Consumer throughout the MQTT broker. We define how the data from the TellUConsumer is communicated to the APIS Producer. The complexities in the real data and the definition of the adaptors have been omitted here since this is more associated with the proprietary use-case than the open-source Eclipse Arrowhead.

A piece of the SysML v1.6 model of the use-case is shown in Fig. 9.

Excerpts of the corresponding SysML v2 model from the current pilot implementation is shown in Fig. 10 in its textual form. Due to current limitations of the pilot implementation, all communication is defined asynchronously.

SOA is not only a composite structure architecture with communication connections but also the behavior in terms of the protocol of messages transferred between the parts. In Fig. 11, we see the normal flow of messages in the simplified chemical use-case. Here, we show synchronous communication between TellUConsumer and APISProducer applying remote method calls as we want to achieve in the future. On the right side of the sequence diagram, we see the communication between the APISProducer and the APISConsumer using asynchronous signals using the publish/subscribe protocol MQTT via MQTTserver. All together the information passes from left to right, and this is repeated in a loop. The lifelines show the participants in the communication with details on their ports and nested ports defining the details of the data transfers. Please refer also to Fig. 10 for details of the communication and see that it is consistent with the sequence diagram for the asynchronous communication. Basic sequence diagrams are also now included in SysML v2.

7 Discussion: alignment with SOA and SoS principles

Given the basic properties of SOA/micro-service architecture and SoS provided in Sects. 2.1 and 2.2, in this section, we analyze which of those properties are captured by our pro-

[4] https://www.prediktor.com

[5] https://tellu.no/en/

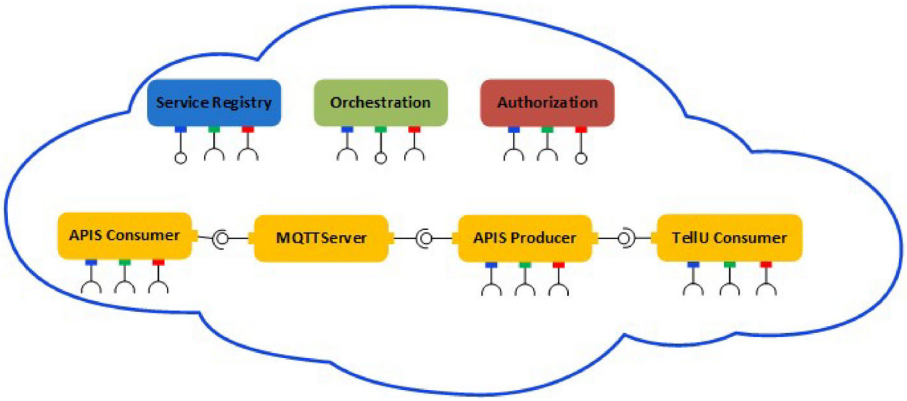

Fig. 8 Simplified chemical use-case

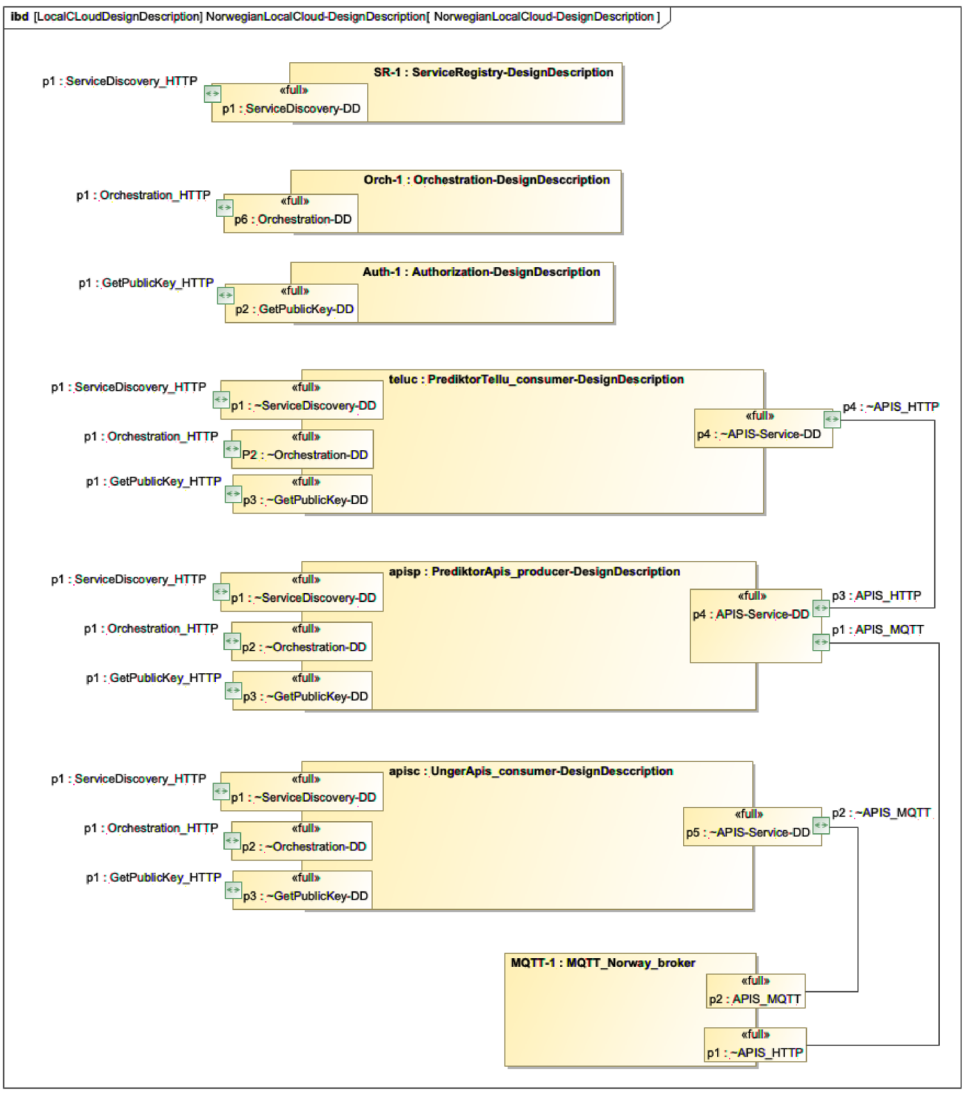

Fig. 9 Chemical use-case modeled according to SysML 1.6

```
part AHFNorway_LocalCloudDD :> ArrowheadCore {
    part TellUConsumer : SysDD subsets systems{
        port serviceDiscovery:~ServiceDiscoveryDD subsets services;
        port apisp:APIS_DD subsets services;

        attribute :>> systemname = "UngerApisClient";
        attribute :>> address = "Unger_network_ip";
        attribute :>> portno = 0;

        state TellUbehavior{
            entry send CallGiveItems("All the items") to apisp.APIS_HTTP;
            then Wait;
            state Wait;
                accept rs:ResultGiveItems
                // Here do whatever about the result rs.ret
            then Wait;
        }
    }

    part APISProducer : SysDD subsets systems{
        port serviceDiscovery:~ServiceDiscoveryDD :> services
        port tellu:~APIS_DD :> services; // providing the APISService
        port apisc:APIS_DD :> services; // talking to APISConsumer

        :>> systemname = "PrediktorApisServer";
        :>> address = "Prediktor_network_ip";
        :>> portno = 6565;
        attribute x:Boolean;

        action giveItems :> ServiceMethod (in itms:String, out ack:Boolean)
         {  /* Forward itms and return an ack */
            first start;
            then send Return_AllItems(itms) to apisc.APIS_MQTT;
            success = true;
            bind ack = success;
         }

        state APISPbehavior{
            entry send Publish("Return_AllItems") to apisc.APIS_MQTT;
            then WaitOnData;

            state WaitOnData;
                accept cl:CallGiveItems via tellu.APIS_HTTP.cll
                do action {
                    first start;
                    then action giveItems(in itms=cl.itms,out ack=x);
                    then send ResultGiveItems(x) to tellu.APIS_HTTP;
                }
            then WaitOnData;
        }
    }

    part APISConsumer : SysDD subsets systems{
        port serviceDiscovery:~ServiceDiscovery :> services;
        port apisp:~APIS_DD :> services;
        :>> systemname = "TellUClient";
        :>> address = "Prediktor_network_ip";
        :>> portno = 1;

        // Now sending signal to the remote behavior through the port functionality
        state MQTT_APISP {
            entry send Subscribe("Return_AllItems") to apisp.APIS_MQTT;
            then Idle;
            state Idle;
                accept Return_AllItems via apisp.APIS_MQTT.retall
                // Get the stuff and do something with them
                then Idle;
        }
    }
```

Fig. 10 Excerpts of the chemical use-case model in SysML v2 textual

posed SysML modeling approach, as summarized in Table 1.

The SysML modeling of SOA and micro-service-based architecture can fulfill several and, eventually, most of the fundamental SoS properties discussed in the literature. A minor use-case validation of this has been performed in Sect. 6. The detailed outcome is further commented on in Table 1.

8 Conclusion and future work

With the ever-growing presence of IoT-based solutions in the industry, there is an increasing need for well-founded system design solutions to address novel challenges. Between rigorous systems modeling on the one hand and the expected level of autonomy and self-organization on the other hand, SOAs have been investigated as a promising bridge concept.

In this paper, we have presented an MBSE approach to model SOA-based SoSs, relying upon SysML, the standard MBSE language, investigating both its established baseline version, v1.6, and its upcoming new major release, SysML v2. We have presented a SysML profile (v1.6), its application to model the Eclipse Arrowhead ecosystem, and our proposed concept for adapting our ideas into SysML v2, along with a comprehensive, real-life industrial case study. We have also discussed the alignment of our concepts with the generally accepted SOA and SoS principles from the respective communities.

As for future investigations, on the conceptual side, we aim to extend our conceptual framework with an adequate representation of SOA-IoT *deployments*, allowing for a tighter integration with real-life Arrowhead installments. On a more practical note, we aim to experiment with different, potentially cloud-based modeling environments to increase the accessibility of our approach.

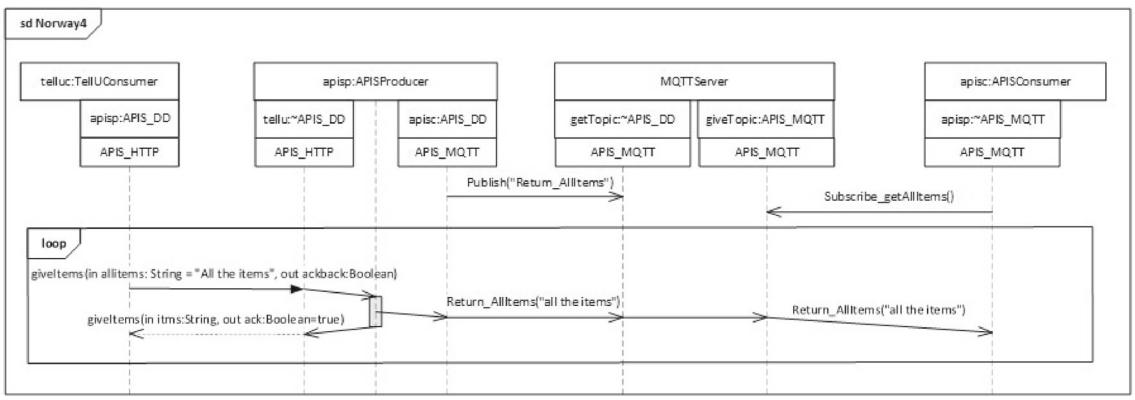

Fig. 11 Normal message flow of the chemical use-case

Table 1 Analysis of SOA and SoS properties modeled by SysML v1.6 and SysML v.2

SoS property	Mechanism and architecture core systems	SysML v1.6	SysML v2	Use-case validation
Look up	ServiceRegistry for finding available services	Profile & Library	DSL & Library & Core system usage[a]	Validated from ServiceRegistry database content after service registration process
Late binding	Orchestration for providing late binding in run-time	Profile & Library	DSL & Library & Core system usage	Validated by registration of the Arrowhead compliant services involved
Loose coupling	Once orchestrated service exchanges will run until further notice by either changed plans (re-orchestration) or changed authentication or authorization policies	Profile & Library	DSL & Library & Core system usage	Operation after orchestration until re-orchestrated
Stateful and/or stateless micro system	Architecture allows for both	Profile	DSL	Both stateful and stateless micro-systems involved in the use case
Operational autonomy	Once orchestrated service exchanges will run until further notice by either changed plans (re-orchestration) or changed authentication or authorization policies	Profile & Library	DSL & Library & Core system usage	Operation after orchestration until re-orchestrated
Belonging	ServiceRegistry, SystemRegistry, DeviceRegistry and OnboardingProcedure enables device, systems and services to be accepted "partners" of a SoS	Profile & Library	Not fully modeled yet	Only service registration was validated
Connectivity on-demand	See late binding above provided that the underlying physical layer interoperability can be provided	Profile & Library	Possible but not yet validated	Run-time orchestration of service exchanges between involved systems
Diversity/ heterogeneity	Integration of heterogeneous micro-systems based on heterogeneous hardware using different communication protocols	Profile & Library	Possible but not yet validated	Implementation on heterogeneous devices using two communication protocols
Emergent behavior	Not yet addressed	Not yet modeled	Not yet modeled	N/A
Managerial independence	Not yet addressed	Not yet modeled	Not yet modeled	N/A
Evolutionary development	Re-orchestration based on preplanned or autonomously updated orchestration rules only partly addressed yet, prototype system in evaluation	Not yet modeled	Not yet modeled	N/A
Geographical distribution	Deployment to multiple devices, local clouds and system of local clouds spanning any geographical distance	Profile & Library	Possible but not yet validated	Deployment of involved systems at various machines and locations
Secure and safe	Authorization (Authentication, Authorization), Onboarding (Authentication and authorization of devices and systems to enter a local cloud), CertificateAuthority (local certificate distribution), DataManager (Audit) systems plus prototype systems addressing Security monitoring and mitigation, Security standard compliance & Safety monitoring and mitigation	Profile & Library of released systems	Not yet modeled	Authentication & Authorization down to individual services

[a] SysML v2 is more focused on describing usages of concepts than is SysML v1

Acknowledgements The authors would like to thank Ákos Horváth, Zoltán Micskei and András Vörös for their work in developing the first version of the SysML profile.

This research is funded by ECSEL, the *Electronic Components and Systems for European Leadership Joint Undertaking* under grant agreement No 826452 (Arrowhead Tools), supported by the European Union Horizon 2020 Research and Innovation Programme and by the member states.

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

Funding Open access funding provided by Lulea University of Technology.

References

1. Maier MW (1996) Architecting principles for systems-of-systems. INCOSE Int Symp 6(1):565–573. https://doi.org/10.1002/j.2334-5837.1996.tb02054.x
2. Maier MW (1998) Architecting principles for systems-of-systems. Systems Eng 1(4):267–284 https://doi.org/10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D
3. Boardman J, Sauser B (2006) System of systems - the meaning of of. In: 2006 IEEE/SMC international conference on system of systems engineering. p 6 https://doi.org/10.1109/SYSOSE.2006.1652284
4. Micouin P (2014) Model based systems engineering: fundamentals and methods. John Wiley and Sons, New Jersey
5. Friedenthal S, Moore A, Steiner R (2014) A practical guide to SysML: the systems modeling language. Morgan Kaufmann
6. Omg systems modeling language (omg sysml), version 1.6 (2019), https://www.omg.org/spec/SysML/1.6/
7. Erl T (2005) Service-oriented architecture: concepts. Prentice Hall, Technology and Design
8. IBM Cloud Education: SOA (service-oriented architecture), https://www.ibm.com/cloud/learn/soa
9. Bashioum C, Behera P, Breininger K, et al (2006) Reference model for service oriented architecture. OASIS Standard soa-rm, Organization for the Advancement of Structured Information (OASIS), https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html, version 1.0, accessed 2021-10-05
10. Edwards WK (1999) Core Jini with Book. Prentice Hall Professional Technical Reference
11. https://www.fiware.org
12. https://www.eclipse.org/basyx/
13. Delsing J (2017) IoT autoamtion - arrowhead framework. CRC Press, Boca Raton
14. https://projects.eclipse.org/projects/iot.arrowhead
15. Paniagua C, Delsing J (2020) Industrial frameworks for internet of things: a survey. *IEEE Syst J* pp 1–11 https://doi.org/10.1109/JSYST.2020.2993323
16. https://en.wikipedia.org/wiki/Service-oriented_architecture
17. Scholten B (2007) The road to integration: A guide to applying the ISA-95 standard in manufacturing. Isa
18. Adolphs P, Bedenbender H, Dirzus D, Ehlich M, Epple U, Hankel M (2015) Referenzarchitekturmodell industrie 4.0 (rami 4.0)
19. https://www.iiconsortium.org/IIRA.htm
20. Sauser B, Boardman J, Verma D (2010) Systomics: Toward a biology of system of systems. IEEE Trans Syst Man Cybern Part A Syst Hum 40(4):803–814. https://doi.org/10.1109/TSMCA.2010.2048024
21. Iso 811346-12:2018, https://www.iso.org/standard/63886.html
22. ISO-18828 - industrial automation systems and integration — standardized procedures for production systems engineering. Standard document Part 1-5, ISO (2018-2019)
23. Wikipedia contributors:(2020)ANSI/ISA-95 — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=ANSI/ISA-95&oldid=977352897 , [Online; accessed 24-January-2021]
24. Karnouskos S, Colombo AW, Jammes F, Delsing J, Bangemann T (2010) Towards an architecture for service-oriented process monitoring and control. In: IECON 2010 - 36th annual conference on IEEE industrial electronics society. pp 1385–1391 https://doi.org/10.1109/IECON.2010.5675482
25. Karnouskos S, Colombo AW (2011) Architecting the next generation of service-based scada/dcs system of systems. In: IECON 2011 - 37th annual conference of the IEEE industrial electronics society. pp 359–364 https://doi.org/10.1109/IECON.2011.6119279
26. Colombo AW, Bangemann T, Karnouskos S (2013) A system of systems view on collaborative industrial automation. In: 2013 IEEE international conference on industrial technology (ICIT). pp 1968–1975 https://doi.org/10.1109/ICIT.2013.6505980
27. OMG (2009) Service oriented architecture modeling language (soaml) - specification for the uml profile and metamodel for services (upms). Tech. Rep. OMG Document Number: ptc/2009-12-09, OMG http://www.omg.org/spec/SoaML/20091101
28. Papazoglou MP, van den Heuvel WJ (2006) Service-oriented design and development methodology. Int J Web Eng Technol 2(4):412–442
29. Papazoglou MP (2013) Web Services and SOA: principles and technology. No. ISBN 9780273732167, Pearson Education Canada, 2nd edn
30. Arsanjani A, Allam A (2006) Service-oriented modeling and architecture for realization of an soa. In: 2006 IEEE international conference on services computing (SCC'06). pp 521–521 https://doi.org/10.1109/SCC.2006.93
31. Bell M (2008) Introduction to service-oriented modeling - service-oriented modeling: service analysis, design, and architecture. No. ISBN 978-0-470-14111-3., Wiley and Sons, New Jersey
32. Urgese G, Azzoni P, Deventer JV, Delsing J, Macii E (2020) An engineering process model for managing a digitalised life-cycle of products in the industry 4.0. In: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium. pp 1–6 https://doi.org/10.1109/NOMS47738.2020.9110365
33. Omg unified modeling language (uml), version 2.5.1 (2017), https://www.omg.org/spec/UML/
34. Fuentes-Fernández L, Vallecillo-Moreno A (2004) An introduction to UML profiles. UML Model Eng 2:6–13
35. Blomstedt F, Ferreira LL, Klisics M, Chrysoulas C, de Soria IM, Morin B, Zabasta A, Eliasson J, Johansson M, Varga P (2014) The arrowhead approach for soa application development and documentation. In: IECON 2014 - 40th Annual conference of the IEEE industrial electronics society. pp 2631–2637 https://doi.org/10.1109/IECON.2014.7048877

36. Delsing J, Eliasson J, van Deventer J, Derhamy H, Varga P (2016) Enabling iot automation using local clouds. In: 2016 IEEE 3rd world forum on internet of things (WF-IoT). pp 502–507 https://doi.org/10.1109/WF-IoT.2016.7845474
37. https://github.com/Systems-Modeling/SysML-v2-Release
38. https://productive40.eu
39. https://productive40.eu/2019/02/25/prediction-online-analyzing-sensors-norwegians-taking-control/
40. Seidewitz E (2007) On a metasemantic protocol for modeling language extension. In: Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD. pp 465–472. INSTICC, SciTePress (2020). https://doi.org/10.5220/0009181604650472