Aerospace Technology Congress, 8-9 October 2019, Stockholm, Sweden
Swedish Society of Aeronautics and Astronautics (FTF)

An Ontological Approach to System-of-Systems Engineering in Product Development

Ludvig Knöös Franzén , Ingo Staack , Christopher Jouannet*, and Petter Krus

Department of Management and Engineering (IEI), Linköping University, Linköping, Sweden
*Overall Design and System Integration, Saab Aeronautics, Linköping, Sweden
E-mail: ludvig.knoos.franzen@liu.se

Abstract

This paper presents an approach to system-of-systems engineering for product development with the use of ontology. A proposed method for building as well as using ontology to generate and explore system-of-systems design spaces based on identified system-of-system needs is presented. The method is largely built to cover the first levels of related work, where a process for system of systems in the context of product development is introduced. Within this work, it is shown that scenarios for a system-of-systems can be used to identify needs and subsequently the system-of-systems capabilities that fulfils them. The allocation of capabilities to possible constituent systems is used to show the available design space. The proposed method of this paper therefore addresses these initial challenges and provides a framework for approaching the system-of-systems design space creation using ontology. A case study is used to test the method on a fictitious search and rescue scenario based on available resources and information from the Swedish Maritime Administration. The case study shows that a representation of a system-of-systems scenario can be created in an ontology using the method. The ontology provides a representation of the involved entities from the fictitious scenario and their existing relationships. Defined ontology classes containing conditions are used to represent the identified needs for the system-of-systems. The invocation of a description logic reasoner is subsequently used to classify and create an inferred ontology where the available system-of-systems solutions are represented as sub-classes and individuals of the defined classes representing the needs. Finally, several classes representing different possible system-of-systems needs are used to explore the available design space and to identify the most persistent solutions of the case study.

Keywords: Systems Engineering, System-of-Systems, Ontology, Description Logic Reasoning, Design Space Creation

1 Introduction

Interest in *system-of-systems engineering* (SoSE) for aeronautical product development has seen steady growth in recent years, and aerospace systems are becoming more and more interconnected with their operational environments [1]. This, together with rapid advancements in technology, generates a desire for systems to collaborate to achieve capabilities that are not reachable by the individual systems alone [2]. In an ever-changing world, traditional approaches to product development for aerospace systems fall short when external factors such as politics, economics, regulations, technologies and doctrines affect the initially specified requirements for the systems. This problem becomes more pronounced when both the long product development time and the expected lifetime for aerospace systems are taken into account. A *system-of-systems* (SoS) perspective for product development of aerospace systems puts a focus on SoS needs and the required SoS capabilities that fulfil them. The corresponding requirements for *Constituent Systems* (CS) are subsequently generated depending on the intended SoS architecture [1]. This SoS conceptualization of aerospace product development consequently takes the process to a more abstract level, where capabilities that are not achievable by the individual systems can be generated through system collaboration. A definition of SoS presented by [3] states that a SoS is separated from a typical complex system by five characteristic properties. These are operational independence of components, managerial independence of components, geographical distribution of components, evolutionary development of components and that the system experiences emergent behaviour. This definition of SoS is used throughout this paper. A proposed holistic product development in a SoS context was presented in [1], where the process of development was divided into five main levels of interest. The proposed SoS design process can be seen in Fig. 1.

The first level of interest presented in Fig. 1 describes how the needs and boundary conditions of the SoS can generate SoS capabilities from possible scenarios. It is stated in [1]

DOI
10.3384/ecp19162004

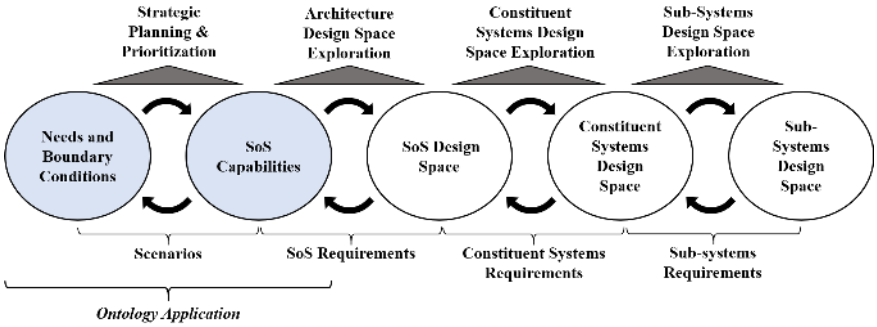

Figure 1: A representation of the holistic SoS design process from [1]. This paper covers the content under Ontology Application.

that analyses can be performed on the process by varying the initial conditions and boundaries to see how required capabilities respond to changes in the SoS needs. This consequently generates a design space of available capabilities that fulfil the SoS needs. The identified SoS capabilities from needs and scenario analyses can then be distributed to constituent systems that make up the SoS design space and architecture. The last two levels of the process describe how CS and their *sub-systems* (SS) should be chosen depending on the requirements generated in the previous levels. It is believed that ontologies are needed to connect the SoS process levels in a coherent way using common language and semantics.

The work presented in this paper therefore aims to provide a method for approaching the first two levels of the SoS-process and generating a SoS design space as presented in [1] using ontology. Furthermore, the influence of changing SoS needs on the available capabilities and SoS design space are also subject to investigation. This paper thereby contributes to the realization of the first two levels of interests shown in Fig. 1. A simple *search and rescue* (SAR) mission based on the Swedish Maritime Administration [4] is used as an implementation example to test the proposed method.

2 Frame of reference

The goal of the presented work is to provide a method for breaking down SoS needs into suitable capabilities and CS that together specify an available SoS design space. This chapter presents work that has been carried out in areas related to this paper and other approaches intended for the problem outlined in the introduction.

2.1 Capability- and system-of-systems engineering

The definition of a system according to the *International Council On Systems Engineering* (INCOSE) specifies that "*A system is a construct or collection of different elements that together produce results not obtainable by the elements alone*". [5]. The term SoS is defined by [6] as "*An interoperating collection of component systems that produce results unachievable by the individual systems alone*". As previously mentioned, [3] specifies that a Complex System or SoS is distinguished from a "conventional" system's definition by five different characteristics properties. [7] presents a collection of views on SoS including a definition of SoS types. These include virtual, collaborative, acknowledged and directed SoS. It is furthermore argued that SoS are rarely developed as SoS. The process referred to as SoSE is rather initialized once an assessment of the SoS performance and capabilities begins. SoSE consequently involves the planning, analysis, organization and integration of CS. Capabilities performed by the CS are combined together with SoSE into SoS capabilities not achievable by the individual systems [2, 8]. This focus on capabilities for SoS can be referred to as capability engineering [9]. The process of capability engineering involves the identification of desired capabilities to be performed by the SoS, as well as investigating the possible options for attaining these capabilities [10]. Furthermore, [10] presents a method for supporting SoSE in translations of SoS capabilities into requirements. *Unified Modelling Language* (UML) object models are here used to model and increase the understanding of the involved systems with their respective functions, thus enabling explorations and trades to achieve desired capabilities. The *Systems Modelling Language* (SysML) has also been used to model SoS in various studies such as [11] and [12].

Enterprise architecture frameworks for defence industries such as the *US Department of Defence Architecture Framework* (DoDAF), the *UK Ministry of Defence Architecture Framework* (MoDAF) and the *NATO Architecture Framework* (NAF) have been used to model SoS in a *model based systems engineering* (MBSE) focused approach [13]. These architecture frameworks are intended to capture the operational, system, service, maintenance and information views of the SoS, among others. Other main actors within SoS and identified methods for approaching complex systems and SoS have been presented in [1]. [1] furthermore describes a holistic product development approach for SoS as mentioned in the introduction. The first of the proposed SoS-process levels shown in Fig. 1 describes the SoS needs and boundary conditions. The understanding of SoS needs and boundary conditions directly influences the strategic planning and prioritization, which together with the intended scenario defines the suitable SoS capabilities.

2.2 Ontology and ontological engineering for system of systems

A different approach to SoSE involves using ontologies. An ontology is an "*explicit specification of a conceptualization*" [14]. This is further explained as a formal and explicit representation of a given domain that involves knowledge of the involved entities and the relationships that exist between them [15]. Ontologies have seen a steady increase in usage for areas such as *systems engineering* (SE), SoSE and capability engineering [16–18]. An ontology can be used to enhance the interoperability aspect of a system or SoS [16], and it has been shown in [19] that scalability is also increased. Another approach to further enhance interoperability but on an ontology level is suggested in [20], where it is explained that a domain-neutral top-level ontology structure can support both the creation of new domain ontologies and the re-usability of existing ones. Top-level ontology examples can be found in [21]. Domain-specific ontologies are ontologies intended to describe individual systems or domains of interest. Such domain ontologies can for example be found in [22], which describes an ontology for aircraft design, or [23] where an ontology for information systems interoperability is presented.

An ontology can be implemented in different ontology languages. Current standards include languages such as the *Web Ontology Language* (OWL) and the *Resource Description Language* (RDF). OWL is based on RDF, but with the advantages of being better equipped for description logics and constraints checking [24]. An ontology made in OWL is composed of individuals, classes and their properties, which together are used to describe concepts of the intended domain modelled in the ontology. OWL supports the use of description logic reasoners, enabling the creation of more complex concepts out of simpler ones. Description logic reasoners can also be used to check for inconsistencies in the implemented ontology [25]. Different types of reasoners support different features [26]. Automatic reasoning over large ontologies requires large computational resources as indicated in [27, 28]. The scalability for automatic reasoning is thus hindered by computational resources. Ontologies that utilize heuristics can contribute to the efficiency of reasoning in domains containing incomplete data and consequently a large number of axioms [29]. There are also various optimization techniques which can contribute to the efficiency of reasoning [28].

Ontology-based approaches for modelling SoS are complementary to UML and SysML due to their ability to describe chosen domains from different terminologies and perspectives [18, 19]. Studies from Georgia Tech's *Aerospace Design Laboratory* (ASDL) have shown that ontologies can be used to model and prune the design space of cyber-physical systems in the context of conceptual design [30]. It has also been shown that matrix-based approaches such as an *Interactive Reconfigurable Matrix of Alternatives* (IRMA) can be used to illustrate and create the available design space from the knowledge captured in the ontology [31].

The presented frame of reference for this paper has identified methods and approaches for the modelling and usage of SoS in the context of aerospace product development. Figure 2 shows a representation of where the proposed scope of this research is situated. Based on the gathered information, a method of modelling an ontology intended for design space explorations on SoS is proposed in the next chapter.

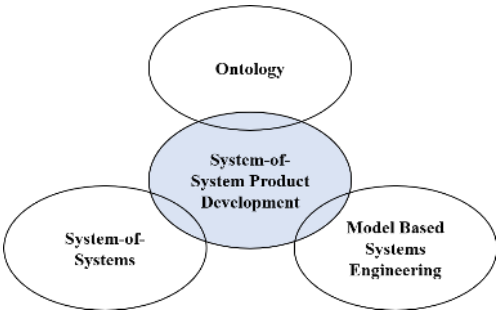

Figure 2: The positioning of the performed work compared to other areas and disciplines

3 Method

This chapter introduces a method for translating SoS needs into capabilities, and subsequently a way of generating the available design space for a SoS. It builds upon the holistic engineering approach for SoS introduced in [1], which can be seen in Fig. 1. As mentioned earlier, this holistic engineering approach suggests that the development is divided into five levels of interest, where each of the levels are recurrent and interrelated with each other. The presented levels are associated with a respective design space of solutions that should be derived by successive investigation and exploration of the previous levels.

3.1 Design space for system-of-systems capabilities

The first level of the presented process involves the description of the intended scenarios that the SoS is believed to be situated in during its lifetime. This description includes definitions for high-level frames of interest such as geopolitics, economics, customer needs, technology, laws and regulations, which together specify the overall boundary conditions and needs for the SoS. These initial conditions and needs must be varied based on different possible scenarios in order to achieve a holistic perspective on the intended SoS [1]. The influence and uncertainty of the high-level frames of interest are thereby evaluated. Suitable capabilities that can fulfil the needs should then be identified in order to generate the SoS capability design space. The capability design space is also subject to changing boundary conditions and needs, and a similar approach by varying scenarios should be used to increase the understanding of the available design space. These variations on initial conditions are used to explore the design space, consequently identifying the persistent solutions least affected by changes. These "resistant" capabilities represented in the design space are deemed to be suitable strategical choices based on the trade of boundary conditions and needs.

The distribution of the chosen capabilities can later be explored in a variety of different SoS architectures where CS and SS are assigned with the required capabilities. The process explained above is illustrated in Fig. 3.

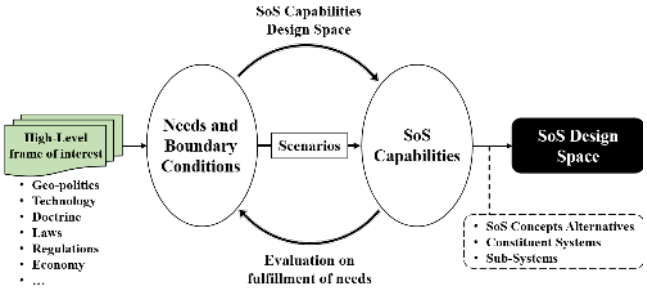

Figure 3: A detailed view of the needs to SoS capability process.

The method proposed in this paper suggests that ontology should be used to represent the main components presented in Fig. 3. The ontology provides a mapping of the involved actors and high-level frames of interest as well as the existing relationships between them. The following section describes a suggested approach for modelling an ontology intended for SoS design space generation and exploration.

3.2 Ontology design and usage

An ontology can be created in several ways, and it is even argued that there is no one correct way of modelling a domain since it depends on the application in mind [15]. The ontology creation method introduced in this paper is largely based on guidelines presented in [15] and [20], and is intended to be used with OWL. It follows eight successive steps which are illustrated in Fig. 4.

3.2.1 Step 1

The aim of the initial step is to determine the scope of the content and domain that is to be represented. Typically, this would correspond to the intended usage for the SoS to be developed, for example SAR.

Step (A) in Fig. 4 is an intermediate step which introduces the option of using a top-level ontology structure. There are several benefits with top-level ontologies that are explained in [32]. These include improved interoperability between domain ontologies.

3.2.2 Step 2

Holistic analyses of possible scenarios for the intended SoS domain and scope are used to identify entities to be implemented in the ontology. Such entities include laws, regulations, weather conditions, available assets, capabilities, needs and more. It is important to specify terms and vocabularies carefully to enable coherent formalism so that ambiguity in definitions is avoided.

Ontologies can suffer from a reinventing-the-wheel syndrome where several ontologies are created for the same domain [32]. It is therefore desirable to consider using existing ontologies for the SoS design space generation instead of creating a redundant one [15]. If a decision is made to utilize existing external ontologies, these should be modified to fit the intended formalism of the ontology under development.

3.2.3 Step 3

The third step in the ontology development process involves the definition and creation of all classes based on the entities identified in step 2. This is typically done in a hierarchy structure with classes and sub-classes. The identified SoS needs of the analysed scenarios should also be defined as classes. Different approaches can be used when developing the class hierarchy as described in [15]. It is important that unequal classes are defined as disjoint for the later invocation of the description logic reasoner.

Step (B) is optional and can be used if a top-level ontology structure is to be utilized. Top-level ontologies include predefined classes that provide an overarching framework for organizing the knowledge of different domain ontologies [32]. The classes and eventual domain ontologies that represent the intended SoS should consequently be allocated to the existing framework of the chosen top-level ontology.

3.2.4 Step 4

This step is used to describe the internal structure, existing relationships and properties of the previously created classes of the ontology. The associations between the classes should come from the scenario analyses in step 2, where the relationships between the different entities are identified. Classes should also include properties that describe them. These properties can include values in the form of numbers and lists. Cardinality specifications are used to describe the number of relationships a class can have in a min., max. or exactly logic. Further explanations and details of available properties and cardinalities in OWL can be found in [15] and [25]. Finally, classes can be defined as either *primitive* or *defined*. The differences between these are so-called *necessary* and *sufficient conditions*. A *primitive class* only includes *necessary conditions* while a *defined class* includes at least one *necessary* and *sufficient condition*. The use of *defined classes* allows for the automated classification and computation of class relationships in the ontology by a description logic reasoner. This is particularly useful when building large ontologies [25]. The classes representing the SoS needs and boundaries should be specified as *defined classes* for the reasoner to associate available solutions to the *necessary* and *sufficient conditions* specified in them. This process is further explained in chapter 4.1.

3.2.5 Step 5

Instances should represent the lowest granularity of the modelled ontology domain. This is described in [15] as "*Indi-*

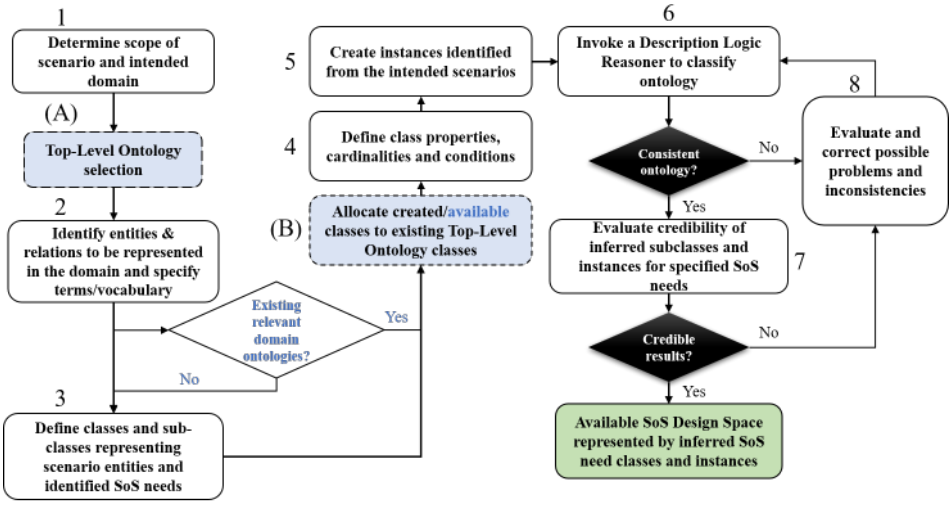

Figure 4: Ontology development process in the context of SoS design space generation.

vidual instances are the most specific concepts represented in a knowledge base". They represent the lowest level of detail for the intended domain and should be related to suitable corresponding classes of the ontology.

3.2.6 Step 6

The sixth step is used to invoke a description logic reasoner to check the consistency of the implemented ontology. The reasoner will classify the ontology and build an inferred ontology based on the previously defined relationships, properties and conditions of the different represented classes and instances. This process will associate the previously implemented classes representing the SoS needs of the ontology with all classes and individuals that have fulfilled the specified conditions. It will consequently show the available SoS design space as sub-classes and individuals to the inferred need classes. If the reasoner classifies the ontology as inconsistent, step 8 needs to be performed.

3.2.7 Step 7

It is recommended that the credibility of the inferred ontology and populated classes representing the needs should be evaluated. The ontology may prove to be consistent but include unreasonable or unwanted inferred relationships of the reasoner. If such unwanted relationships or results exist, step 8 needs to be carried out.

3.2.8 Step 8

This final step in the process presented in Fig. 4 is only to be performed if inconsistencies or unreasonable results or relationships exist in the inferred ontology. Possible corrections should be performed in the definition of classes and their relationships, properties and conditions. OWL supports the usage of various ontology debuggers that can prove to be useful for finding the source of any inconsistencies [33]. Unreasonable results can be further investigated by description logic querying, where the inferred ontology is used to answer "questions" about specific classes or instances [34].

The end result of the process shown in Fig. 4 is an inferred ontology where the defined need classes represent the available SoS design space. This process both creates and reduces the available SoS design space based on the description of needs identified from the possible scenarios for the intended SoS.

4 Implementation of case study

In order to test the proposed method of creating a SoS design space from specified needs with an ontology, a simple case study based on the operations and resources of the *Swedish Maritime Administration* (SMA) was performed. According to statistics, SAR at sea has accounted for approximately 60% of the operations performed by the SMA during the past three years and is a good example of a SoS where changing circumstances can affect the available solutions.

The scenario considered for the case study is fictitious and solely based on the available assets, regulations and capabilities of the SMA. It is also worth mentioning that the purpose of the case study is not to provide a complete ontology for Swedish SAR but instead to test the validity and usefulness of the proposed method in this context. Hence, the implemented scenario is kept simple in order to show the possibilities of the proposed method even at low levels of scenario detail. The use of a top-level ontology was also excluded for the purpose of this case study.

Following the process shown in Fig. 4, a SAR scenario was determined as the domain to be modelled in the ontology. This fictitious SAR scenario was created based on available information such as the assets, resources and regulations of the SMA [35] and is illustrated by a simple sketch in Fig. 5.

The scenario shown in Fig. 5 illustrates that there is a subject situated in the sea near Gothenburg, Sweden. The subject can make a distress call and indicate a position. Information

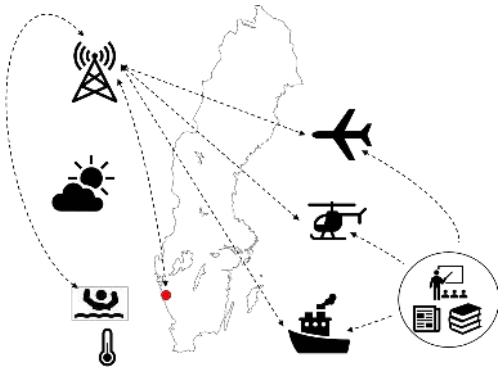

Figure 5: A fictitious scenario based on available assets and resources of the SMA.

about the distress call is forwarded to the *Joint Rescue Coordination Centre* (JRCC) located in Gothenburg. The JRCC can dispatch suitable and available assets for the SAR of the subject. The available assets considered in this scenario are Bombardier Dash 8 Q300 aeroplanes, AW 139 helicopters and two fictitious kinds of sea vessels: fast and slow. The environment is associated with weather conditions, temperatures and time of year and day.

4.1 Ontology modelling

The process of identifying all entities and relationships to be modelled in the ontology involved detailed scenario analyses where available capabilities and assets were broken down into functions and means. Information flows between entities such as communication links were determined as well as the relationships between assets and their corresponding subcomponents. After identifying entities and relationships to be represented in the ontology, the class and sub-class definition was initiated. The case study was implemented in the Protégé ontology editing software, which is based on OWL [36]. The class structure of the study was created using a top-down development process where the most general classes, such as System and Environment, were created first [15]. The class definition was done in a "is_a" manner, which meant that all sub-classes of an intended class inherited its properties. A depiction of the case study class hierarchy can be seen in Fig. 6. Classes not equal to each other are defined as disjoint.

The identified needs of the SoS are represented as the solution classes in Fig. 6 and are explained in more detail further down. The available assets of the case study are represented under the system classes which are defined by the various sub-system classes of the hierarchy structure. Figure 7 shows an example of how the *HelicopterSystem* class is defined.

The properties, cardinalities and conditions describing the various classes in the hierarchy were based on specifications of SMA assets, resources and available capabilities. The fictional sea vessel's properties were estimated based on high and low speed sea vessels for SAR. Mapping between capabilities, functions, systems and other classes was carried out to create the relational properties between all entities represented in the ontology hierarchy. Assets and resources

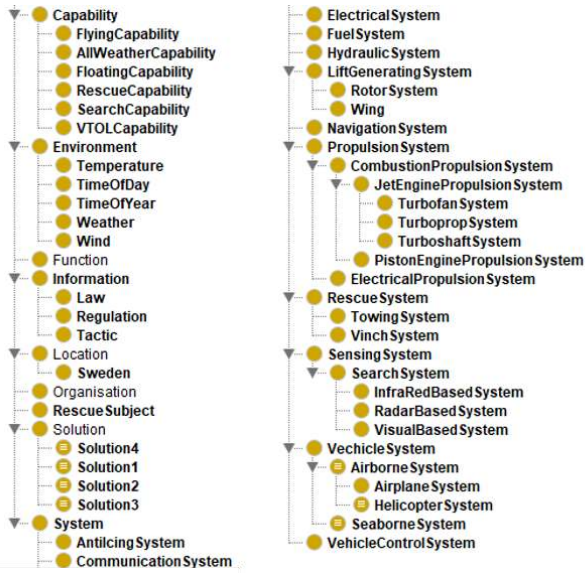

Figure 6: The ontology class hierarchy for the intended scenario.

capable of performing the functions or capabilities from the former breakdowns were related to their respective function. Once a satisfactory mapping of the relationships between classes was achieved, the step of implementing the lowest levels, or instances, of the ontology was performed. The instances for this case study represented specific individuals such as an AgustaWestland 139 helicopter which is part of the SMA assets [35]. These individuals were assigned data properties describing their individual performances. Figure 7 shows the definition of a *HelicopterSystem* and the AW139 instance within the case study ontology.

It should be noted that the instance values described in Fig. 7 do not include any units. The data properties describing the relationships are instead assigned comments describing the intended units. The *hasOperationalRange* data property, for example, has the unit of kilometres in this case study. The instances of the sub-systems are defined for the sake of simplicity as various types indicating that resources can be composed of different alternatives such as different types of radar.

Finally, the solution classes describing the intended SoS needs were specified and given *necessary* and *sufficient conditions*. This was done so that the *Solution* classes were defined as equal to specified needs of the SAR scenario. The procedure was carried out by describing the required capabilities, functions and performances in the *necessary* and *sufficient conditions* of the *Solution* classes. An example of such a class can be seen in Fig. 8. The conditions described in the *Solution* classes of this case study are, as with the SAR scenario, fictitious and are only used to show the intended process of the method. It can be seen in Fig. 8 that the class describes a need for both a *search* and a *rescue capability*. It furthermore specifies that the rescue vessel needs to be able to carry at least 10 persons and have a length of 15 (meters) or above.

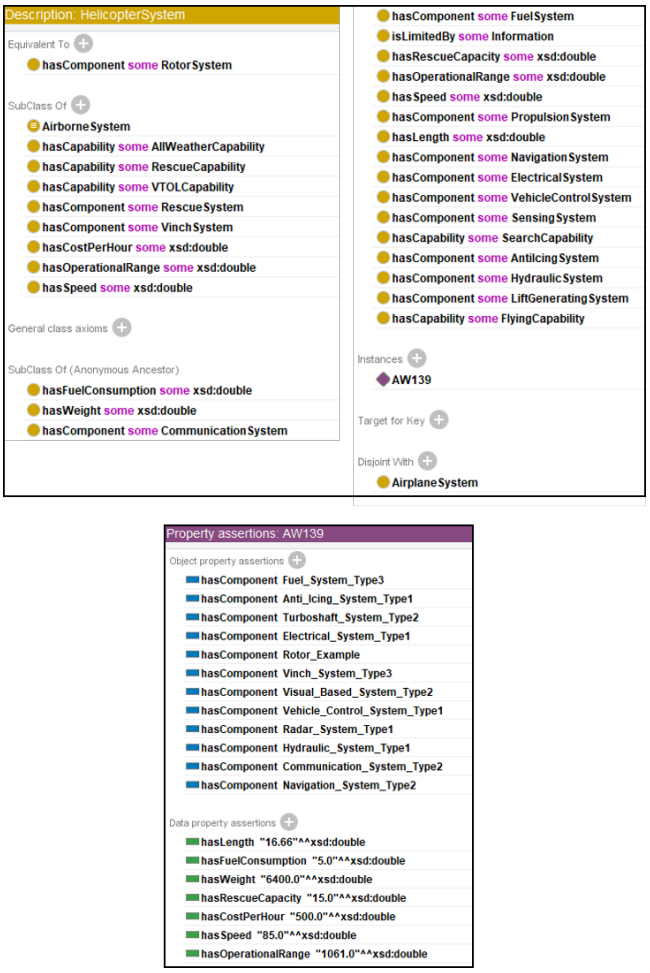

Figure 7: The definition of the HelicopterSystem class (top) and the AW139 instance (bottom), where both are defined by respective object and data properties.

4.2 Ontology usage

In order to classify the ontology, check consistency and obtain the available sub-classes and instances of the *Solution* classes, a description logic reasoner needed to be used. Protégé has a set of default reasoners as well as additional reasoners available through plug-ins. The default *Pellet* reasoner was considered suitable for this case study and was used throughout the implementation process. (For a detailed comparison of available reasoners in Protégé, see [26].) The reasoner was invoked to classify the ontology and consequently generate the logically inferred hierarchy. This process generated the available SoS design spaces as sub-classes and individuals of the defined *Solution* classes that could fulfil the *necessary* and *sufficient conditions*. These results are shown in Figs. 9 and 10.

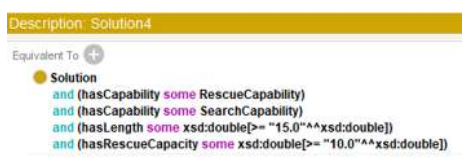

Figure 8: The necessary and sufficient conditions specified for the Solution4 ontology class.

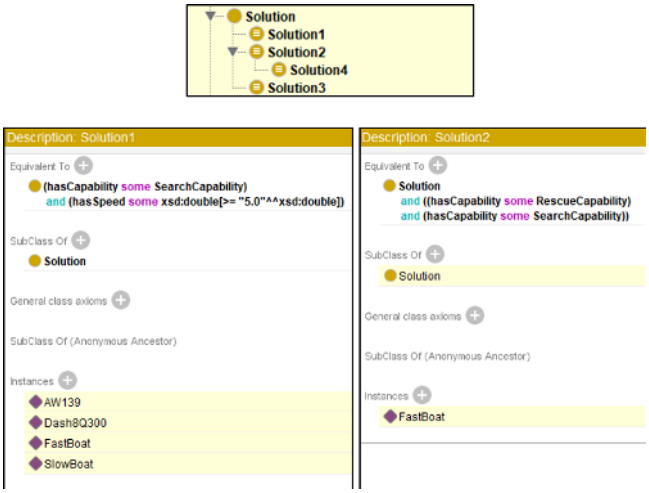

Figure 9: The inferred solution class structure (top) and the description of Solution classes 1 and 2 (bottom). Solution1 describes a need for a search capability and a minimum speed, while Solution2 requires both search and rescue capabilities.

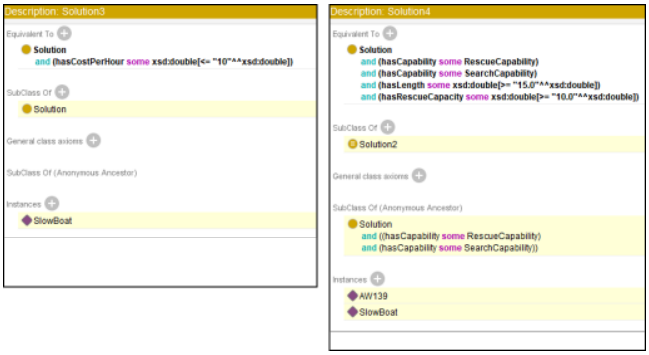

Figure 10: The description of Solution classes 3 (left) and 4 (right). Solution3 specifies a need to keep the cost under a specified cost per hour.

It can be seen in Fig. 10 that the *Solution4* class from Fig. 8 has been populated with the *AW139* and *SlowBoat* individuals due to their fulfilment of the *necessary* and *sufficient conditions* specified in the class description. It should be noted that *Solution4* has been inferred as a sub-class of *Solution2* in Fig. 9. This is due to the solutions described in *Solution4* also fulfilling the conditions in *Solution2*. It is possible to carry out design space explorations by defining several solution classes based on different possible SAR scenarios and needs as illustrated in Figs. 9 and 10. Figure 11 shows the available design space for all *Solution* classes. The *Solution2* class describes needs for both search and rescue capabilities. All individuals in the *Solution4* class fulfil the conditions and are subsequently part of the solution for the needs described in *Solution2*.

By comparing all inferred design spaces, it can be seen that the individuals with the highest reoccurrence represent

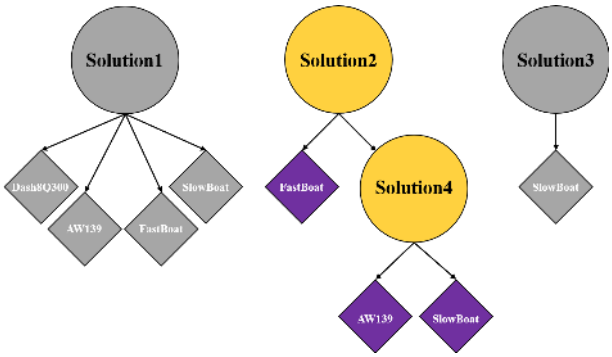

Figure 11: The inferred SoS design spaces where the Solution2 class is shown in the middle with corresponding individuals and Solution4 as a sub-class.

the most persistent solutions. The consequent SoS design space generated with the method is available in the OWL file of the inferred ontology and can be exported for further investigations and analyses. Such analyses can for example include determining the most suitable number of assets described by the inferred individuals based on *Measure of Effectiveness* (MoE), which is not provided by this method.

Finally, this case study has shown that the proposed method can be used to generate SoS design spaces for SAR and specified needs from a SoS scenario.

5 Discussion

The method proposed in this paper is intended to be used for any SoS design space creation and delimitation. The implementation of the case study was based on a fictitious existing SoS. This can be referred to as a near-term SoS where available resources and how best to use them are investigated. The method and consequent use of ontology allows for the evaluation of new resources in combination with existing ones. New resources and capabilities can be introduced to the ontology structure and can be related to the existing entities. The use of a reasoner can once again prove useful for automatically expanding the ontology with new resources depending on their description. A new entity with a capability to fly can, for example, automatically be placed as a sub-class of *AirborneSystem* if an *AirborneSystem* has been defined with *necessary* and *sufficient conditions* specifying that an airborne system has the capability to fly. This ability makes ontologies highly scalable and flexible for introducing new entities to the modelled domain and building up taxonomies. However, this comes at the cost of increased computational time for automatic reasoning as described in section 2.2.

A combination of existing and new SoS solutions can be referred to as a mixed SoS. It is also possible to use the proposed method for analyses of long-term SoS where no existing resources are available. This typically corresponds to creating a SoS from scratch where only vague definitions of possible needs and capabilities are defined and related. Assets and resources can be guessed and evaluated based on the scenario and epoch analysis of the intended SoS.

As mentioned in the frame of reference, there are supplementary methods for ontologies available for modelling and performing analyses on SoS. Enterprise architecture frameworks have been successfully used to describe SoS as well as intended scenarios in a SysML or UML language [13]. Relational databases have a similar structure to ontology, and it is possible to describe a SoS with all entities and the relationships that exists between them here as well. However, the advantage with ontologies is that they work under the previously mentioned open world assumption which, together with an ability to perform description logic reasoning, can be used to infer implicit knowledge of the intended domain. This implicit knowledge can reveal emergent behaviours of the SoS which are very important and desirable to understand at an early stage of development [17]. Nevertheless, ontology has some limits when it comes to inference by a reasoner. Such limits are shown in detail in [34], where inference is used to a high degree on an ontology. The presented method is, as previously mentioned, built upon existing methods for creating an ontology. There are many available methods and pieces of software for creating an ontology. Protégé was chosen as the implementation software due to its compatibility with OWL and RDF, as well as the support for description logics. The proposed method of this paper is however neutral in terms of the choice of software and could be used provided that the software supports a description logic reasoner.

An important delimitation of the presented case study is the exclusion of a top-level ontology structure. This was mainly excluded due to the fact that no existing ontology was to be used for the implemented example. The use of a top-level ontology would also imply a large increment in the class hierarchy structure and size, which would make the ontology less comprehensible to the human eye. Furthermore, the ontology for the case study was, as previously mentioned, implemented to show the possibilities for creating and exploring the SoS design space and not to provide an extensive ontology for SAR operations. However, the inclusion of a top-level ontology can be utilized at a later stage if desired due to the scalability and flexibility of ontology structures. This would require some restructuring of the class hierarchy to fit the top-level ontology properly, but would also enable re-usability and combinability with other ontologies that utilize the same top-level design. The scope of the modelled SAR case study was kept at a simple level to visualize the goal of the proposed process and method. It could however easily be expanded to include more details about the domain and available system components. The example components shown in Fig. 7 could be replaced with actual sub-system representations that have their own sub-classes and instances, leading to a question about fidelity levels in the ontology. The fidelity or detail level of the ontology is determined at the very beginning of the proposed method in Fig. 4. This is however also expandable at a later

stage due to the flexibility of ontologies. The results obtained in Figs. 9 and 10 show that the SoS needs can be represented as demands for the necessary performances and capabilities. The usefulness of this strategy becomes increasingly distinct in ontologies featuring a larger number of resources and possible solutions. The reasoner becomes a valuable asset for inferring knowledge and creating the available design space in larger ontologies.

The results in Figs. 9 and 10 show that the available solutions are instances of the *Solution* classes. It is possible to use the method in order to generate sub-classes representing the available design space as a complement to a set of individuals. The open world assumption that OWL ontologies work under can, however, complicate class inferences. A closed world assumption assumes that data which does not exist is false, while an open world assumption sees non-existent data as simply unknown. Ontologies consequently do not make any assumptions about incomplete data. This means that ontology classes can include more entities that are simply not yet known unless the class has been explicitly stated not to do so using closure axioms [25]. The open world assumption is also a reason why ontologies are so flexible and easy to expand. Instances are, however, easier to work with since they represent the lowest level of the intended ontology and are less affected by the open world assumption.

Finally, the implemented case study is still under development at the time of writing this paper. More SMA information and resources must be implemented in order to generate a larger pool of available solutions for the SoS design spaces. More detailed capability and functional breakdowns are subject to implementation to obtain a more comprehensive picture of the existing relationships in the domain. Additionally, more extensive design space explorations will be performed by defining more solution classes based on scenario and epoch analyses of the case study. As mentioned in section 3.1, varying the initial conditions to increase the understanding of the design space is an important future addition to the case study.

6 Conclusions and future work

The proposed method introduced in this paper has shown that a design space of possible SoS solutions can be generated using ontology. An implementation of a *search and rescue* (SAR) case study was carried out to test the method and show the process of building an ontology for SoS design space generation. Several ontology classes describing different needs were implemented and populated with suitable instances representing solutions by a description logic reasoner. This corresponded to a small design space exploration which was used to identify the most common solution elements of the SoS. The future work of this study involves the expansion of the case study and extraction of the SoS design space generated in the inferred ontology file. Similar approaches to those presented in [31] can be used to transfer the information from the ontology to matrix-based approaches. This transfer allows for more advanced numerical calculations and optimizations needed for the remaining levels of interest proposed in [1]. Furthermore, the design space generation of this paper has only shown available types of solutions and no number of required assets. Future work includes the determination of the most suitable SoS architectures with regard to number and collaboration of assets. The work so far has, however, shown that ontologies provide a resilient way of exploring and generating SoS design spaces based on specified needs. It contribute to the ways of approaching the complex challenges of today's product development.

References

[1] Ingo Staack, Kristian Amadori, and Christopher Jouannet. "A Holistic Engineering Approach for Aeronautical Product Development". *Proceedings of the Congress of the International Council of the Aeronautical Sciences*, pages 1–15, Belo Horizonte, 2018.

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

[3] Mark W. Maier. "Architecting Principles for Systems-of-Systems". *Systems Engineering*, 1(4):267–284, 1998.

[4] Sjöfartsverket, "Sjö- och flygräddning", 2019. [online]. available: https://www.sjofartsverket.se/sv/sjofart/sjo–och-flygraddning/. [accessed 2019-07-02].

[5] INCOSE, "About Systems Engineering", 2019. [Online]. Available: https://www.incose.org/about-systems-engineering. [accessed 2019-07-02].

[6] David D Walden, Garry J Roedler, Kevin J Forsberg, Douglas R Hamelin, and Thomas M Shortell. "Systems Engineering Handbook". Wiley, San Diego, 4 edition, 2015.

[7] Jo Ann Lane. "What is a system of systems and why should i care?". Technical report, USC-CSSE-2013-001, University of Southern California, 2013.

[8] Charles Keating, Ralph Rogers, Resit Unal, David Dryer, Andres Sousa-Poza, Robert Safford, William Peterson, and Ghaith Rabadi. "System of systems engineering". *EMJ - Engineering Management Journal*, 15(3):36–45, 2003.

[9] CAE, "Capability Engineering", 2019. [Online]. Available: https://www.cae.com/media/documents/Def ence_Security/Services_-_Documents/datasheet.capability.engineering.pdf. [accessed 2019-07-02].

[10] Jo Ann Lane. "System of systems capability to requirements engineering". *Proceedings of the 9th International Conference on System of Systems Engineering: The Socio-Technical Perspective, SoSE 2014*, pages 91–96, 2014.

[11] Jo Ann Lane and Tim Bohn. "Using SysML to Evolve Systems of Systems". Technical report, INCOSE, 2010.

[12] Marco Mori, Andrea Ceccarelli, Paolo Lollini, Bernhard Frömel, Francesco Brancati, and Andrea Bondavalli. "Systems-of-systems modeling using a comprehensive viewpoint-based SysML profile". *Journal of Software: Evolution and Process*, 30(3):1–20, 2018.

[13] Ron Williamson. "INCOSE (MBSE) Model Based System Engineering (SoS) System of Systems/Enterprise Activity Introduction", 2012.

[14] Thomas R Gruber. "A translation approach to portable ontology specifications". *Knowledge Acquisition*, 5(2):199–220, 1993.

[15] Natalya F Noy and Deborah L McGuinness. "Ontology Development 101: A Guide to Creating Your First Ontology". Technical report, Stanford, 2014.

[16] Leo Van Ruijven. "Ontology for systems engineering: Model-based systems engineering". In *Proceedings - UKSim-AMSS 6th European Modelling Symposium, EMS 2012*, pages 371–376. IEEE, 2012.

[17] Gary Langford and Teresa Langford. "The making of a system of systems: Ontology reveals the true nature of emergence". In *2017 12th System of Systems Engineering Conference, SoSE 2017*, pages 1–5. IEEE, 2017.

[18] Huseyin Dogan, Michael J de C. Henshaw, and Julian Johnson. "7.5.1 An incremental hybridisation of heterogeneous case studies to develop an ontology for capability engineering". In *INCOSE International Symposium*, volume 22, pages 956–971. INCOSE, 2012.

[19] John S. Osmundson, Thomas V. Huynh, and Paul Shaw. "Developing Ontologies for Interoperability of Systems of Systems". In *Conference on Systems Engineering Research*, 2006.

[20] Barry Smith, Mauricio Almeida, Jonathan Bona, and Mathias Brochhausen. et al. "Basic Formal Ontology 2.0 SPECIFICATION AND USER'S GUIDE". Technical report, 2015.

[21] Viviana Mascardi, Valentina Cordì, Paolo Rosso, and Paolo Rosso Viviana Mascardi, Valentina Cordì. "A Comparison of Upper Ontologies". Technical report, 2007.

[22] Markus Ast, Martin Glas, Tobias Roehm, and Bauhaus Luftfahrt eV. "Creating an Ontology for Aircraft Design". In *Deutscher Luft- und Raumfahrtkongress 2013*, pages 1–11, 2013.

[23] Peter Morosoff, Ron Rudnicki, Jason Bryant, Robert Farrell, and Barry Smith. "Joint doctrine ontology: A benchmark for military information systems interoperability". In *CEUR Workshop Proceedings*, volume 1523, pages 2–9, 2015.

[24] Diana Kalibatiene and Olegas Vasilecas. "Survey on Ontology Languages". Lecture Notes in Business Information Processing, 2011.

[25] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe, Simon Jupp, Georgina Moulton, Nick Drummond, and Sebastian Brandt. "A Practical Guide To Building OWL Ontologies Using Protégè 4 and CO-ODE Tools Edition 1.3". Technical report, Manchester, 2011.

[26] Sunitha Abburu. "A Survey on Ontology Reasoners and Comparison". *International Journal of Computer Applications*, 57(17):33–39, 2012.

[27] Thomas Lampoltshammer and Stefanie Wiegand. "Improving the computational performance of ontology-based classification using graph databases". *Remote Sensing*, 7:9473–9491, 07 2015.

[28] Volker Haarslev and Ralf Möller. "On the scalability of description logic instance retrieval". *Journal of Automated Reasoning*, 41:99–142, 08 2008.

[29] Kristin Stock, Didier G. Leibovici, Luciene Delazari, and Roberto Santos. "Discovering order in chaos: Using a heuristic ontology to derive spatio-temporal sequences for cadastral data". *Spatial Cognition & Computation*, 15:115–141, 2015.

[30] Kevin Lynch, Randall Ramsey, George Ball, Matt Schmit, and Kyle Collins. "Conceptual design acceleration for cyber-physical systems". In *11th Annual IEEE International Systems Conference, SysCon 2017 - Proceedings*, 2017.

[31] Matt Schmit, Simon Briceno, Kyle Collins, Dimitri Mavris, Kevin Lynch, and George Ball. "Semantic design space refinement for model-based systems engineering". In *10th Annual International Systems Conference, SysCon 2016 - Proceedings*, 2016.

[32] Robert Arp, Barry Smith, and Andrew D Spear. "Building Ontologies With Basic Formal Ontology". Massachusetts Institute of Technology, Cambridge, 2015.

[33] Konstantin Schekotihin, Patrick Rodler, and Wolfgang Schmid. "OntoDebug: Interactive Ontology Debugging Plug-in for Protégé". Technical report, Alpen-Adria-Universität, Klagenfurt, 2018.

[34] Robert Stevens, Margaret Stevens, Nicolas Matentzoglu, and Simon Jupp. "Manchester Family History Advanced OWL Tutorial". Technical report, The University of Manchester, Manchester, 2015.

[35] Lars Widell and Peter Hellberg. "Svenskt Program för Sjö- och Flygräddning". In *Sjöfartsverket Dr-nr 1199-13-01359 (in swedish)*, 2013.

[36] Mark A. Musen. "The protégé project. a look back and and a look forward". *AI Matters. Association of Computing Machinery Specific Interest Group in Artificial Intelligence*, 1(4), 2015.