MSaaS infrastructure capabilities for discovery, composition and execution of simulation services

Journal of Defense Modeling and Simulation: Applications, Methodology, Technology
XX(X):1–23

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Jo Erskine Hannay[1], Tom van den Berg[3], Scott Gallant[4], Kevin Gupton[5]

Abstract

Modeling and Simulation as a Service (MSaaS) embodies the idea that simulations should be composed quickly for the task at hand from loosely coupled shared components, *simulation services*, in a cloud-based environment. These simulations are then offered, as *composed simulation services*, to human and technical consumers. Instrumental to this, is functionality that lets a simulation operator discover and compose simulation services and execute the composition. We describe this functionality in terms of what we call MSaaS *infrastructure capabilities*. Following the idea of stepwise refinement, the discovery and composition of simulation services can be done at design-time using implementation-independent information about simulation services and at implementation time using implementation-specific information about simulation services. The execution environment can also be set up at design time and at implementation time. We therefore describe the MSaaS infrastructure capabilities in terms of how they are used on both implementation-independent and implementation-specific service information. By doing these elaborations, we intend to gain greater insight into how to perform simulation service discovery, composition and execution. We conclude that although much of the required functionality for an MSaaS infrastructure is available through existing platforms and frameworks, it is necessary to offer this functionality as services, alongside (composed) simulation services, to fulfill the MSaaS vision.

Keywords

Simulation service, Simulation as a Service, MSaaS infrastructure capability, Service abstraction

1 Introduction

Simulation support to operations, training and exercises holds much potential, making it possible to support and augment operational processes and enhance training with new aspects and with extended exposure[1]. Simulation support to defense activities is perceived to become progressively more important as multinational forces become more interconnected[2].

However, setting up and executing distributed simulations is a lengthy process with various obstacles depending on the complexities and characteristics of the systems involved. The process must often be repeated for each operation or exercise, as system versions and settings may have been updated or changed in the meantime. Connecting systems across networks also presents its own set of issues. All these challenges make it necessary to have skilled technicians in place at each site during a distributed simulation life cycle, adding to the already complicated logistics and sometimes lengthy planning for operations and exercises.

Modeling and Simulation as a Service (MSaaS) – and the NATO Allied Framework for MSaaS[3;4] in particular – presents a vision that setting up simulations for operations, exercises and training should be rapid and easy. The *service* concept embodies reusability by standardization of common functionality, and composability through loose coupling and standardized service descriptions.

The idea is illustrated in Figure 1, where suppliers share simulation services in a cloud environment. Simulation operators use a web-based portal to discover and compose simulation services into a simulation composition to be executed. Composed simulations can themselves be offered as services to be reused. The cloud environment facilitates

[1]The Norwegian Defence Research Establishment (FFI), [3]Netherlands Organisation for Applied Scientific Research (TNO), [4]Effective Applications Corporation, [5]Applied Research Laboratories, The University of Texas at Austin

Corresponding author:
Jo E. Hannay, Norwegian Computing Center, Pb. 114 Blindern, NO-0314, Norway.
Email: jo.hannay@nr.no, ph.: +47 22852574

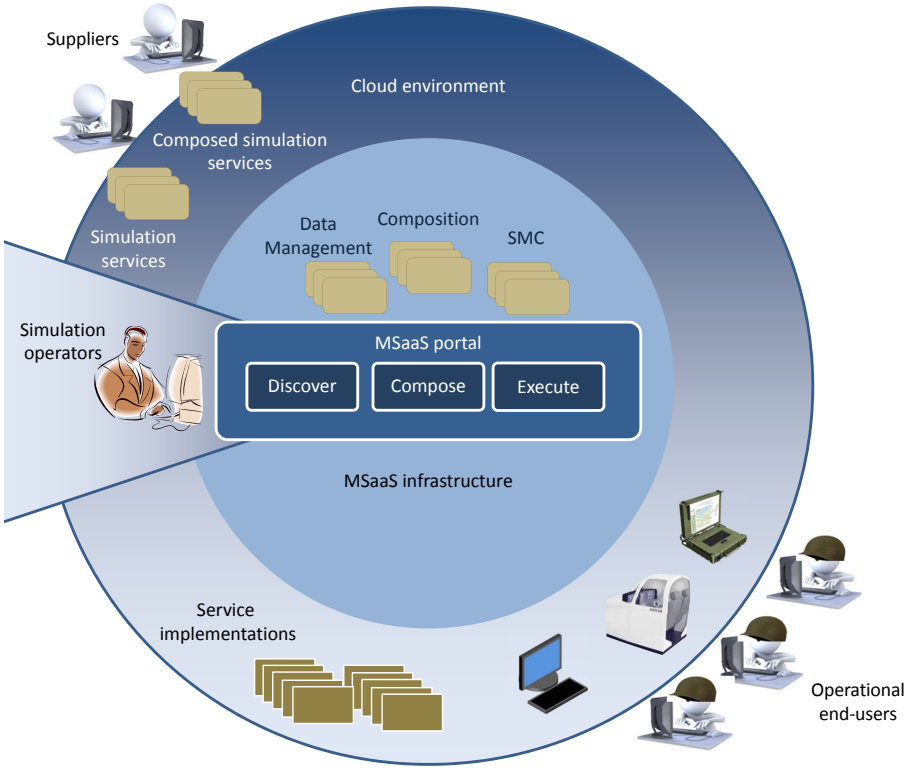

Figure 1. Allied Framework for MSaaS with MSaaS portal functionality (discover, compose execute) and MSaaS infrastructure capabilities for data management, composition and service management and control (SMC).

simulation access "on demand, anywhere". Indeed, cloud-based simulations and MSaaS are considered "grand challenges", entailing new requirements for simulation software, and the need for service descriptions, service discovery and service composition among other things[5].

The functionality in the portal to discover, compose and execute simulations is provided by a collection of MSaaS *infrastructure capabilities*, which are divided into capabilities for *data management*, *composition* and *service management and control* (SMC) (Figure 1). The main line of discussion is an elaboration on what these MSaaS infrastructure capabilities should be; the purpose being to understand better the essential mechanisms for handling simulations in a service-oriented environment. Our elaboration is grounded in earlier experiences with MSaaS.

In the MSaaS reference architecture[4], services are currently referred to as implementation independent. That is, services are identified by their implementation-independent service descriptions, and the reference architecture lists a number of pertinent services that are particular to modeling and simulation. When implementation-independent descriptions of services are standardized and expressed in a machine-readable format, tools can be built to support some degree of automatic discovery and composition. This supports the MSaaS vision of rapid simulation deployment, and further, the vision that simulation operators (Figure 1) may be non-technical trainers or other operational personnel in the future.

However, to be useful for developers in the world of service-oriented standards and simulation protocols, where each of these standards and protocols may be at different levels of implementation-specific abstraction, the MSaaS reference architecture needs to include corresponding levels of abstraction. Moreover, stepwise refinement principles, as expressed in the steps from conceptual modeling, through design, to implementation further motivate a service concept that holds multiple levels of abstraction.

Therefore, when elaborating the MSaaS infrastructure capabilities, we do so while considering how these capabilities operate on several levels of simulation service abstraction. This gives a better understanding of the service abstraction levels themselves and how infrastructure capabilities might be used in stepwise refinement through these levels of abstraction.

MSaaS relies on cloud infrastructures shared between nations and organizations in NATO and between civilian infrastructures. This means that simulation services, their compositions, as well as the infrastructure capabilities must

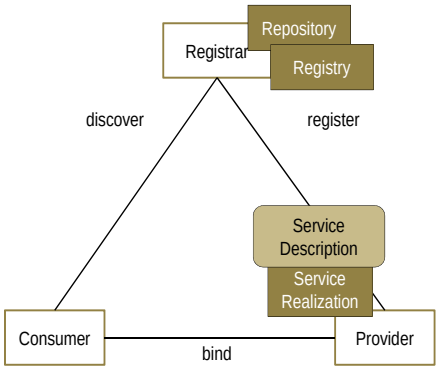

Figure 2. Key roles in service orientation (the SOA triangle adapted from Erl[17]).

be realized in software that is at appropriate levels of cloud application maturity; see for example the *Cloud Ready*, *Cloud Friendly*, *Cloud Resilient*, *Cloud Native* categorization of Kratzke[6;7]. However, our focus in this paper is on understanding infrastructure capabilities at the functional level (the *Service Composition* and *Application* layers in the Kratzke's reference model[6]). Identifying the appropriate levels of cloud application maturity for MSaaS is the next important step and is not discussed in this text.

Epistemologically, this work sorts under the building of what Gregor[8] calls *Analysis* type theories and *Design and Action* type theories. The former consists of conceptualizations of "what is"; where, in our case, the "what is" is not a physical entity, but is itself a conceptual entity; namely a reference architecture. The latter type of theory describes "how to do" things and includes design principles. Neither type of theory supports predictions expressed in the theory itself that can be refuted in the traditional manner. Instead, it can be argued that they imply meta-predictions by the assumptions that the conceptualizations and designs are beneficial to various ends.[9] That the conceptualization we present is beneficial can be verified analytically and empirically by how useful practitioners and researchers find it[10], how well-formed it is in terms of parsimony[11], how interesting it is[12] and by other quality aspects of theories.[13;14] This verification must be done over time by other researchers and practitioners, in concert with researchers who continuously evolve the conceptualization.[15;16]

In Section 2, we recapitulate and elaborate on the service concept of the MSaaS reference architecture, where services can be declared – using service descriptions – at several levels of abstraction; from implementation independent to implementation specific. We then introduce the MSaaS infrastructure capabilities in Section 3, and elaborate on the constituent data, composition and SMC capabilities in Sections 4, 5 and 6, relating to service abstraction levels. We conclude in Section 7.

2 The service concept of the MSaaS reference architecture

The concept of *service* embodies abstraction, loose coupling, reusability, composability and discovery[17]. The concept underlies old-style "SOA monolith" architecture, "microservice" architecture and "nanoservice" architecture (or "serverless architecture" for the notion of *Function as a Service*[6]); all of which are relevant for supporting MSaaS and the special demands of simulations and specialized simulation architectures.

2.1 Roles in service orientation

From service-oriented architecture (SOA), we emphasize three main roles: the *Service Provider*, the *Service Consumer* and the *Service Registrar*; see Figure 2. In this discussion, these roles are technical, rather than organizational. A service provider registers a service it wishes to provide to the community with a registrar. The registrar deposits the description in a repository and the concrete information for run-time binding in a registry. The consumer consults the registrar for descriptions from the repository to prepare for service consumption, and the consumer consults the registrar for binding information from the registry to an appropriate provider.

2.2 Service description and realization

In this discussion, a service consists of a

- *service description* for the benefit of consumers of the service, which consists of
  - an *interface*[18] with functional and operational signatures for syntactic interoperability[19],
  - a *contract*[18] with elaborations of what the functions and operations declared in the interface do in terms of functional and operational semantics, for a degree of semantic interoperability[19], as well as a specification of contractual non-functional requirements,
  - a *model* for *simulation services*[4] of that which is being simulated in the form of limited information (white-box view) on internal workings of the simulation functionality provided by the simulation service, necessary for determining what assumptions in the environment the simulation service uses, for pragmatic interoperability[19].
- a service realization, in the form of either requirements specifications, for the benefit of developers who will realize the service in software and code implementations for actual consumption at run time.

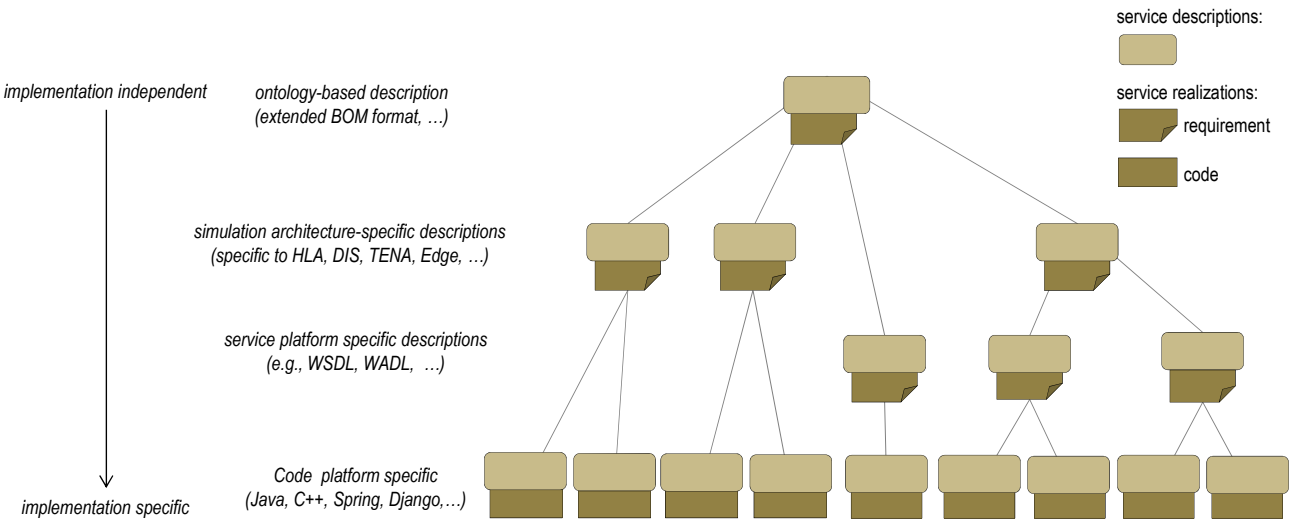

Figure 3. Levels of service abstraction for the simulation domain.

2.3 Service abstraction levels

In line with stepwise refinement and detailing[20], we promote the use of the service concept at various levels of abstraction. Figure 3 illustrates the idea for the simulation domain.

2.3.1 Implementation-independent level. At the highest level of abstraction, a service consists of a simulation architecture-independent and implementation-independent service description that can be used for design-time discovery and composition. At this level, service descriptions can be written using the terms of an ontology. An ontology provides a structured and machine-readable domain-specific but implementation-independent vocabulary for describing the elements of a domain and relationships between them. Relevant examples include the C2Sim ontology for C2Sim interoperability[21;22] and simulation ontologies such as the Trajectory Simulation Ontology[23]. One can combine ontologies to obtain the necessary vocabulary.

For simulation services, Base Object Models (BOMs)[24], when written in the vocabulary of an ontology, can be used at this level for service descriptions that contain aspects of interface, contract and model. BOM *entity descriptions* (expressible in Unified Modeling Language (UML) class diagrams) can be used for interfaces, *patterns of interplay* (expressible in UML sequence diagrams) provide the aspect of contract that describe intended use of functions and operations, and BOM state machines (expressible as UML state diagrams) provide a white-box view of the service that may express model aspects. When using ontological reasoning, further semantic information can be inferred[25–27].

Further, at the implementation-independent level of abstraction, a service consists of implementation-independent service realizations in the form of requirements specifications that developers should use in further detailing and refinement and, ultimately, implementation in concrete software.

2.3.2 Simulation architecture-specific level. At more implementation-specific levels of abstraction, a service consists of architecture-dependent service descriptions. For simulation services, an ontology-based service description might be refined to simulation-architecture dependent descriptions for, architectures such as the High Level Architecture (HLA)[28], Distributed Interactive Simulation (DIS)[29], Test and Training Enabling Architecture (TENA)[30] or for novel simulation architectures based on edge and fog computing (e.g., SpatialOS*). Service realizations in the form of requirements are then refined in terms of these architectures.

2.3.3 Service platform-specific level. Further, or other, levels of de-abstraction are possible. As one example, service descriptions may be refined into service platform formats, such as the Web Services Description Language (WSDL)[31] (interface) and WSDL-S[32] and SAWSDL[33] (contract) for Big Web Services (WS*)[34] that send messages over the Simple Object Access Protocol (SOAP)[35], Web Application Description Language (WADL)[36] (interface) and SA-REST (contract) for Representational State Transfer (REST) style technology[37], or emerging leaner

*https://improbable.io

Dette er en postprint-versjon/This is a postprint version.
Publisert versjon/Publised version: 10.1177/1548512919896855

formats. Service descriptions at this level should accommodate both stateful and stateless micro services, larger stateful SOA structures (*SOA monoliths*), and anything in between.

Service realizations in the form of requirements are then refined in terms of the chosen formats.

2.3.4 Implementation-specific level. At the implementation-specific level, service realizations take the form of code written in coding frameworks or in plain old Java and C++. Of particular relevance for cloud technology are virtualization and containerization. Containers package functionality ready to go, complete with necessary operating system-level virtualization and other dependencies in the package. Containers run within the context of a single operating system whose kernel is shared by all containers, dispensing with the need for an infrastructure (*hypervisor*) for sharing computing resources between multiple virtual machines running on a host. This enables lightweight packaging of deployable units of functionality, whether they be simulation nanoservices, microservices or SOA-monoliths[38].

At this level, there will be descriptions containing technical details required for containerization or deployment in virtual machines, such as the required operating system, libraries, memory, processing, disk and networking, etc.

2.3.5 Significance of levels. In this manner, a single service declared at an implementation-independent level has description and realization refinements in various protocols, service platform formats, and ultimately, in various coding frameworks. There are related ideas, with tools for transforming descriptions from one layer to the next[23].

To illustrate, some services will be simulation services "from the top", in that every refinement path goes through a simulation architecture-specific description; such as a service with only the two left-most refinement paths in Figure 3. Other services, may have both simulation architecture-specific refinements and non-simulation architecture-specific refinements; such as a service with the three left-most refinement paths in Figure 3. Examples are terrain analysis services (e.g., for route planning, line of sight and vantage point services[39]), automatic identification system (AIS) services[40] and weapons effects services, which can have simulation architecture-specific descriptions – and are thus, simulation services at this level of abstraction, as well as non-simulation architecture specific descriptions for use, e.g., in operations planning in a command and control (C2) system[41]. Yet other services (e.g., weather services and map services) would not have any simulation architecture-dependent service descriptions.

All three elements of a service description (interface, contract, model) can exist at the various levels of abstraction. For example, interfaces can be specified without regards to any programming language, and models range from conceptual models at the implementation-independent and simulation architecture-specific levels, to executable models at the implementation-specific level, in line with scenario abstraction levels[42].

Although there can be any number service abstraction levels, the MSaaS reference architecture[4] defines three levels of architecture that reflect the different levels of service abstraction described above: reference architecture level (implementation independent), solution/domain architecture level (simulation architecture specific), and implementation (implementation specific).

2.4 Service abstraction and the MSaaS engineering process

The MSaaS engineering process[43] is the MSaaS extension to the Distributed Simulation Engineering and Execution Process (DSEEP)[44] and its Multi-Architecture Overlay (DMAO)[45]. The system under development in this process is what is called a *simulation environment* (Figure 4).

A simulation environment consists of a number of simulation software components that adhere to one or several simulation architecture protocols (HLA, DIS, TENA or other protocols) in designated *enclaves*[45]. In each enclave, components are organized in different *topologies*[46]; e.g. an event-based topology (such as the HLA) and a data-centric simulation topology using a shared state database (such as SpatialOS). In turn, each topology might adhere to one or several appropriate service platform-specific styles.

Simulation components in each enclave relate to an enclave-specific *simulation data exchange model*[44], which specifies what data is shared between components in an enclave (e.g., a Protocol Data Unit (PDU) set for DIS, a Federation Object Model (FOM) for HLA, a Logical Range Object Model (LROM) for TENA). Together, the enclave-specific simulation data exchange models constitute the simulation data exchange model of the entire simulation environment.

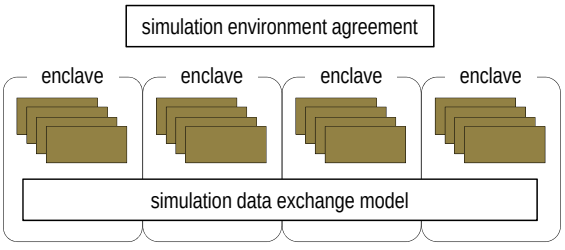

Figure 4. Simulation Environment.

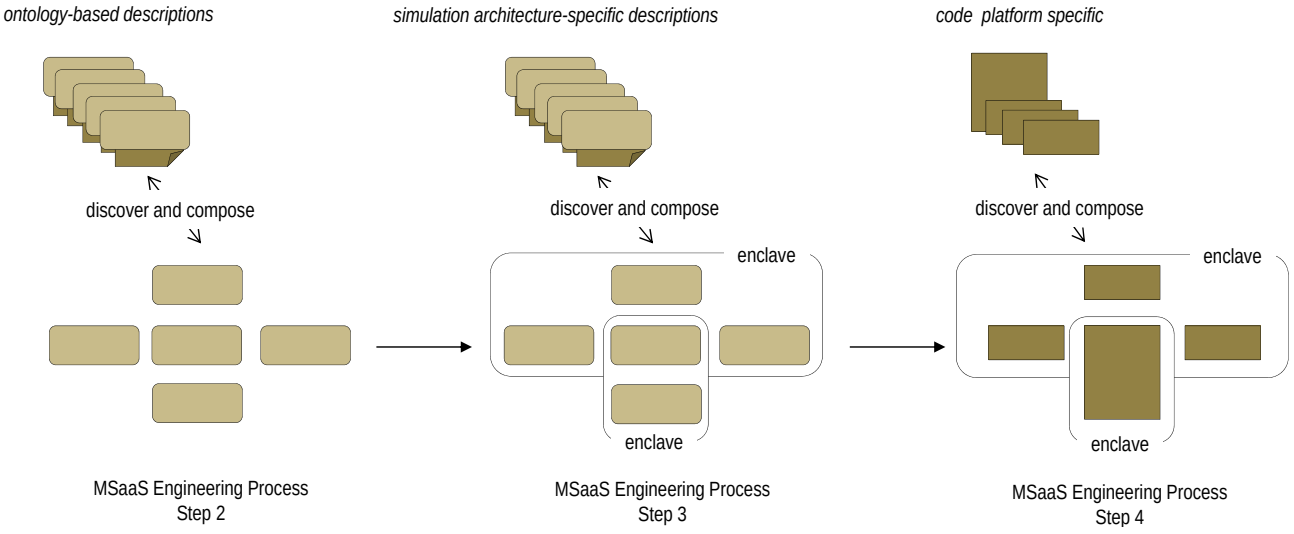

Figure 5. Simulation service abstraction levels and the MSaaS engineering process.

While the simulation data exchange model gives syntactic interoperability[19], the *simulation environment agreements*[44] specify the intended meaning of exchanged data and other semantic information.

The various steps of the MSaaS engineering process require one to work at different levels of abstraction (using corresponding architecture artifacts[47]). In the presence of service descriptions at relevant levels of abstraction, this can be formulated as follows; referring to Figure 5: At the conceptual modeling step (Step 2), one defines a *composed simulation service*[4] by searching, discovering and composing simulation service descriptions at the highest level of abstraction independent of simulation architecture or protocol. The simulation service descriptions must hold sufficient information to determine which services can be composed to satisfy the conceptual model and conceptual scenario. During the design step (Step 3), one refines the composed simulation service to a simulation-architecture specific design by using simulation architecture-specific descriptions of the services chosen in the previous stage. At this abstraction level, simulation services are arranged in abstract versions of the above-mentioned topologies in enclaves, using mediation services such as gateways to connect enclaves or to provide translation and transformation services within an enclave. During the development step (Step 4), the composed simulation service is refined into a simulation environment, implemented by choosing appropriate code realizations of the chosen services. Note that a simulation component may implement several services, as illustrated in Figure 5 by the large component implementing two services.

The vision of MSaaS is that the presence of simulation services with multiabstraction-level descriptions and code realizations will greatly speed up what presently are relatively time-consuming DSEEP steps. Of course, in the interim, services might have to be de-abstracted and/or implemented; in which case, the requirements specifications of the simulation services are there to help developers in that process. And even more typically today, services will be generated bottom-up from code, with service description reverse-engineered (hopefully). Nevertheless, the vision is that the end state, after more or less chaotic development, has yielded ready-to-use services as sketched in Figure 3.

Whenever an appropriate service description or service implementation does not exist, the MSaaS engineering process states that actual development of a service must be undertaken; complete with service description and service implementation(s).

2.5 Strict service interpretation

A service is *always* a packaging of a service description and one or several service realizations. Conversely, a service realization (requirements specification or code) on its own is *never* a service.

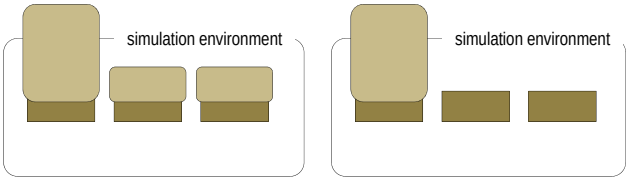

Figure 6. Simulation as a Service. Composed simulation service (left), simulation environment exposed as a service (right).

The dynamic discovery mechanism through the Service Registrar is essential to the loose coupling of service orientation. Without the discovery mechanism, services must be known statically, thereby precluding the idea that services may be created and hosted independently of specific consumers. In fact, it has been argued that service orientation in practice often does not include the discovery mechanism and only involves the lower part of the SOA triangle[48;49]. For example, at the implementation-specific end of the scale, current practices around popular technologies associated with service orientation, such as WebSocket[50], Advanced Message Queue Protocol (AMQP)[51], JavaScript Object Notation (JSON)[52], REST, etc. usually avoid or omit service descriptions. It is also common to write WSDLs when using WS* technology but with no registrar.

Without the discipline of using service descriptions and service registrars, it is also tempting to revert to tighter coupling in other areas of the consumer-provider relationship. For the vision of MSaaS to be realized, it is essential that service registrars are used and that service descriptions exist in *sufficient amount* at *appropriate levels of abstraction*.

Finally, other technologies than those mentioned above commonly associated with service orientation can also be used for writing service descriptions. Thus, the use of, e.g., WS*, REST, etc. is neither sufficient, nor necessary, for realizing service-oriented architecture.

2.6 Simulation as a Service

The notion of composed simulation service (Section 2.4) embodies the idea that entire simulations composed as in Section 2.4 can be exposed as a service. Following Section 2.5, the composed simulation service itself must have a service description (Section 2). In practice, this may be done by one or more of simulation services exposing certain functionality through its service description (Figure 6 – leftmost). As an example, consider interoperating C2 systems with simulations for the purpose of simulating operations during training and exercises[53–56] or for wargaming plans[41;57;58]. A simulation service offers functionality to give force structures, and orders and report structures to the simulation in terms of the Military Scenario Definition Language (MSDL)[59] and the Coalition Battle Management Language (C-BML)[60], and also offers functionality to receive reports from the simulation.

This notion of *simulation as a service* then means that the functionality is not tailored to a specific C2 system and that the functionality is declared in a service description that is discoverable by any potential consumers of that functionality. The simulation environment as a whole is the provider of the service, where the service is declared in the service description(s) of the designated simulation services that expose the relevant simulation functionality as a service.

It is also meaningful to speak of simulation as a service even when a simulation is not composed of simulation services, but are simply made up of conventional simulation components. This would be the case if one (or several) of these components has what amounts to a service description that exposes the simulation as a service in the same manner as above (Figure 6 – rightmost). We will not discuss this mode of simulation as a service, but focus on composed simulation services.

3 MSaaS infrastructure capabilities

The MSaaS infrastructure capabilities we present in the following represent a systematization of concepts from ongoing deliberations on MSaaS; in particular from various MSaaS architecture work[4;61–64], from MSaaS prototype and container-technology studies[38;65–69] and from work done in the Executable Architecture Systems Engineering (EASE) research activity[70].

Apart from a basis for further conceptual development in terms of understanding how MSaaS must work in various styles of simulation architecture, our suggestions are intended as a guide for developing a comprehensive infrastructure for MSaaS. If developed incrementally, viable parts of the infrastructure can be tested and validated to guide further increments. These would then be the first steps in a more concerted research effort.

We now introduce the MSaaS infrastructure capabilities that give the functionality for discovering and composing simulation services and executing the resulting composed simulation services. These capabilities consist of *user-facing applications* and *back-end services* in the sense of the NATO Consultation, Command and Control (C3) Taxonomy[71]; both of which are loosely coupled pieces of functionality (front-end or back-end) as expressed in the service concept of Section 2.

At present, very few MSaaS infrastructure capabilities exist as loosely coupled applications and services, even though a lot of functionality that can be used to implement an MSaaS infrastructure does exist in traditional forms. Therefore, Sections 4–6 will present a selection of infrastructure functionality that we argue should be offered as MSaaS infrastructure capabilities in the future.

Figure 7 shows the MSaaS infrastructure capabilities, and their relationships, that we discuss in the following sections. MSaaS infrastructure capabilities consist of data management capabilities, composition capabilities and service management and control (SMC) capabilities.

4 Data management

The MSaaS engineering process Steps 2–4 (Figure 5) entails significant data management activities. MSaaS presents engineering challenges and opportunities that data management services can help mitigate. Data management services, or simply "data services", contribute to enablement and automation of simulation life cycles, simplification of simulation engineering and execution, and delivery of simulation services to geographically distributed points of need.

The selection and composition of simulation services in, e.g., Step 2 of the MSaaS engineering process requires the availability of different kinds of data, such as service descriptions at an implementation-independent level, stakeholder needs and objectives, authoritative reference information, and a conceptual model and conceptual scenarios.

4.1 Simulation data management applications

Data management user applications are the front-end user interfaces that simulation operators (Figure 1) use to inspect conceptual and executable model knowledge bases (Section 2.3.5), discover simulation services for composition, and manage data artifacts generated and maintained throughout the MSaaS engineering life cycle.

4.1.1 Simulation life cycle data management application. This user application is for creating and revising data artifacts through the engineering life cycle, including the conceptual and executable models for the composed simulation service (and implementing simulation environment) under development, data models, simulation-environment design, and managing post-execution data and analyses.

4.1.2 Simulation service discovery application. Existing either as a stand-alone application or included as part of other life cycle tools, this user application allows simulation operators to discover simulation services for composition at design time. Discovery is based on service descriptions, which must contain various metadata facets at appropriate levels of abstraction (Figure 3), including (conceptual) model entities, properties, and interactions/effects.

4.1.3 Simulation conceptual modeling application. This user application is for discovering, retrieving, creating, and managing conceptual and executable model data. This application must handle behavior and effects representations (perhaps as process UML activity, sequence, and state diagrams) and also the management of all manner of *enumerations*, such as equipment types, compositions of parts, environmental feature types, and miscellaneous reference category codes (perhaps as UML class diagrams and ontologies). This application is related to the simulation integration application (Section 5.1.2) via the composition services, so that the simulation operator can assess whether the (partial) models in the various simulation service description align with the target conceptual and

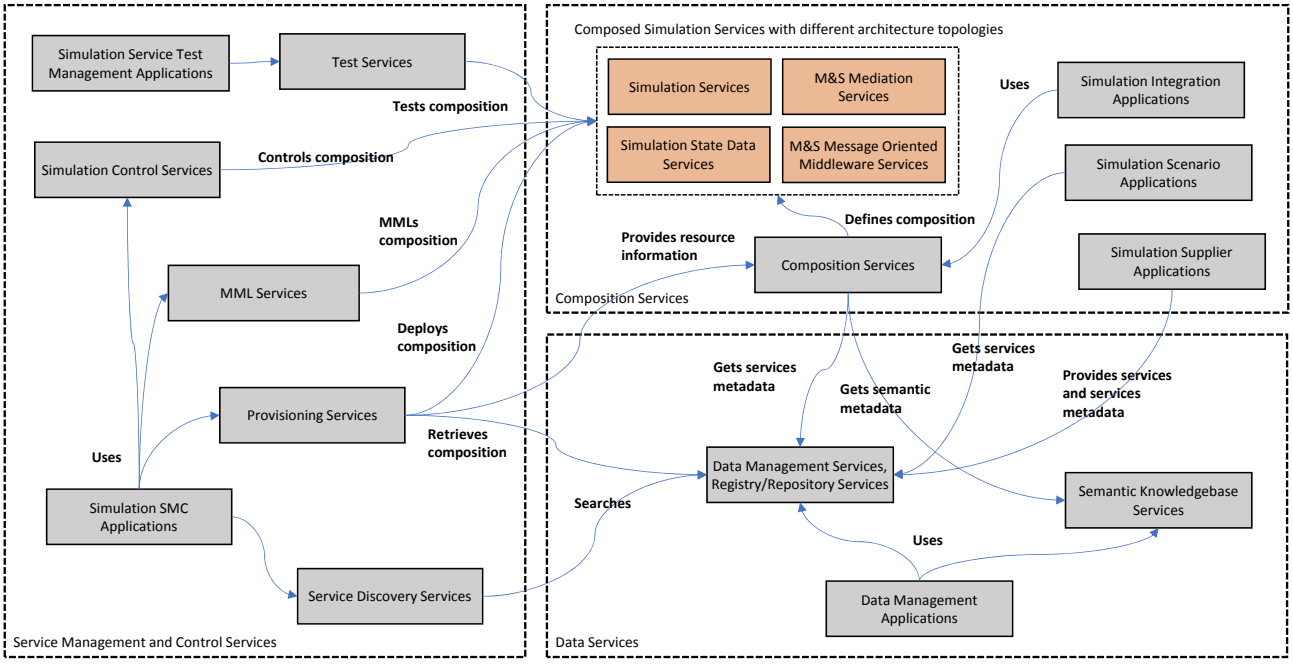

Figure 7. Relationships between MSaaS infrastructure capabilities.

executable models of the composed simulation service (and implementing simulation environment) under development.

4.2 Simulation data management services

What follows is an initial set of data services and interaction patterns that are pertinent for the discovery, composition and execution of (composed) simulation services (Figure 5). These data services are derived from the MSaaS engineering process and based on lessons learned in the engineering of several simulation environments. The data services are organized in two categories:

- Data management across the MSaaS engineering process: To automate and enable the process depends on managing inputs and outputs of the MSaaS engineering process steps. In Figure 7, this service category is shown as Data Management Services, including Registry/Repository Services.
- Semantic linking of operational (stakeholder) needs to simulation solutions: To integrate simulation services and the data fusion required by simulation services depend on the selection and alignment of simulation services to stakeholder needs and objectives. Semantic knowledge bases can further enable the MSaaS engineering life cycle data management; in particular simulation service composition by providing domain-specific information that enables decision making. Semantic knowledge bases relate entities (e.g., vehicles), composed parts (e.g., sensors, weapons and functional parts), capabilities, consumption of resources (e.g., fuel, water, food, ammunition and energy), and interactions with other entities (e.g., trailers pulled by vehicles, tanks' ability to damage a building and vehicle traction on different terrain). In Figure 7, this service category is shown as Semantic Knowledgebase Services.

Note that these data services operate and provide data artifacts for all the service abstraction levels of Section 2.3. In particular, the data services must manage the various service description components (interface, contract and model) (Section 2.2) at relevant levels of abstraction according to the steps in the MSaaS engineering process. When doing this, one must also retrieve the corresponding conceptual and executable scenario and model[42] specifications for the composed simulation service (and implementing simulation environment) under development.

4.2.1 Data management across the MSaaS engineering process. Every step of the MSaaS engineering process highlights opportunities for data management, considering the inputs and outputs, life cycle-related data stored and retrieved, and external data (not directly pertinent to the life cycle) to be referenced. In practical simulation life cycles (e.g., the multinational Viking exercise life cycle arranged by the Swedish Armed Forces or simulation-based operational test planning processes), process steps are specialized and often less sequential, but similar information management activities occur. With particular regard to the composition of simulation services, the MSaaS engineering process Steps 2–4 require management of conceptual analysis artifacts guiding the selection of simulation services for composition as well as the composed simulation service design artifacts:

- Discovery—including search, publish/subscribe notifications based on interest—of (composed) simulation services by service descriptions (Section 2.2).
- Retrieval and delivery of information artifacts essential to creating composed simulation services (and implementing simulation environments) such as conceptual and executable model and scenario specifications, software libraries, descriptive metadata, and initialization/configuration parameters.
- Publication of new or modified models and composed simulation services and their implementations in terms of simulation environment designs, including version and revision controls.

Of course these abstract services are not unique to MSaaS – they are ubiquitous constructs, frequently standardized by communities and organizations to facilitate interoperability. For MSaaS, we seek to apply these services to improve simulation-data management, facilitate simulation life cycle automation, and increase the usability and availability of composed simulation-services and their implementations as simulation environments. To that end, we highlight the following service patterns as pertinent to MSaaS:

- *Value-add pipelines, including the provenance trace of information artifacts produced through each engineering step*. Information products—such as planning documents, scenarios, federation agreements, and simulation-specific environmental/terrain datasets—may be produced in a sequential, multi-step manner. Further, the steps may be performed in parallel by different stakeholders participating in the simulation life cycle. Output from one step becomes input to the next step, and tracking the trail of inputs and performers (humans and algorithms) can be used to cue subsequent performers that their inputs are ready, can contribute to the validity of the simulation, and can indicate the re-use potential (or not) of an output.
- *Version and revision control of information products produced iteratively*. Planning documents, scenarios,

conceptual models, and environment design are examples of information artifacts that are often produced iteratively by one or more stakeholders to a simulation. One might consider version control in this context as a value-add pipeline that loops and labels products successively as "draft", "in progress", "ready for review", or "final".

- *Reference data in the simulation life cycle*. Presumably every step of an simulation life cycle adds value, but some steps may be manual—requiring human action, decision, or intervention—while some steps may be automated in part or in whole. To achieve more automation, supporting services may utilize external reference data. Supporting services may query for and retrieve reference data—such as military force structures, entity performance parameters, terrain data sets, country codes, or even elements from previous simulations—in order to prompt users with choices, prompt users with recommendations, or even fully automate the step based on established logic.

- *Extraction and transformation*. Steps in a simulation life cycle may require extraction of data from an information artifact or transformation of data from an information artifact. These common integration patterns that in many cases are readily automatable, particularly for syntactic transformation; semantic transformation can sometimes involving a semantic knowledge base, by way of a reference data set or a subject matter expert.

4.2.2 Semantic linking of operational need to simulation solutions. The simulation life cycle depends largely on our ability to integrate software and data to implement a conceptual model and fulfill the stakeholder objectives. And our ability to perform these integrations and alignments depends on the thoughtful, repetitive alignment of domain concepts to solutions for simulating those domain concepts. Consider the thousands of entity types, associated parts and attributes, states and interactions, and environmental phenomena that constitute complex simulation environments; these elements trace from early objectives through numerous authoritative data sources, into conceptual models and scenarios, to be realized by composition of simulation services and subsequent execution and analysis. Simulation data management services can help to mitigate the subject-matter knowledge transfer shortfalls in complex simulation environment engineering.

Ontological analysis and conceptual modeling are widely appreciated in the modeling of simulation environment requirements, but the products of these activities play a role in automating more of the simulation life cycle. The following categories stand out as potential knowledge sets that may be exploited by the simulation life cycle.

- Entity-type catalogs—Entity types such as vehicles, aircraft, lifeforms, munitions, and environmental features are frequently the fundamental conceptual building blocks of simulation environment design. Entity types involve a wealth of information that could be captured in more objective, reusable form rather than the conventional methods of embedding intrinsic knowledge in software source code or in simple spreadsheets of enumerations. Entity types are
  - often defined with numerous identifiers and aliases; are defined within multiple taxonomies
  - often composed of numerous parts such as mechanical elements, weapons, sensors, and other equipment
  - related to many categories of characteristic and performance data
  - associated with behaviors, states, potential interactions, and observables.

- Named-entity catalogs—Simulation life cycles within any domain may encounter the same entities often and can benefit from managing information about those entities. The country of France, the White House, the Grand Canyon, the mayor of London, 10th Mountain Division, and Theresa May are all examples of named entities, about whom numerous reference datasets may have data one might need to fuse and integrate for a simulation purpose, or for which a visual or behavior model might exist. Named entities are often declared to be of some entity type, as defined in entity-type catalogs.

- Event & behavior models—Building upon the entity-type elements described so far, the actions, events, processes, states, interactions, and other types of relationships make up more of the conceptual model or ontology of a domain. These elements are often captured in terms of engineering models for design purposes and captured as information exchange models (e.g., HLA FOMs), data models, or embedded in source code during implementation.

- Miscellaneous reference data and code lists – Still more reference data is used during simulation environment design, data which can be managed as knowledge bases and utilized by services enabling the simulation life cycle. Country codes, language codes, religion codes, and task lists are some

examples of domain concepts that are essential to characterizing the entities in a conceptual model and to metadata describing simulation services or datasets.

All of these knowledge sets play a role in the simulation life cycle and in enabling the discovery and composition of simulation services. Accessing and utilizing multiple data sources is cumbersome and unwieldy unless the various identifiers, aliases, and taxonomies used by those data sources are available and sufficiently integrated. The same can be said of fusing and integrating data from multiple processes and sources, as part of an simulation life cycle. Similarly, the functional needs of a composed simulation service (and implementing simulation environment) under development, as expressed by entity types and named entities in the conceptual and executable models and scenarios, cannot be linked to simulation services, 3D visual models, or other available simulation assets for composition in an automated way, unless the identifier for real-world entities are relatable to the simulation assets via metadata.

Relevant service patterns for service composition:

- *Heterogeneous service integration*. Not all data sources, scenario tools, and other data services use the same identifiers, codes, and definitions for the force structures, equipment and materiel, and geospatial features; integrating data across data producing and consuming services can be a manual effort unless identifier mappings are managed and exploited for automation. Knowledge bases of synonyms, aliases, and identifier mappings can be used to search multiple heterogeneous external and internal data stores without the consuming service (or human user) from having to tailor queries to different schemes. Similarly, misaligned input and output formats and semantics among data tools and simulation services can be mitigated with knowledge bases of definitions and mapping.

- *Design decisions enabled by domain ontology*. Knowledge about the relationships among data sets can enable applications to prompt users with smart defaults or informed options. Relating tasks or behaviors to relevant simulation services or simulators can simplify decision making an engineer or operator must make when composing or employing simulation services. Common scenarios or past simulation records can be related to the needs of an engineer or operator by task performed, operational environment conditions, military unit type, or equipment types, for example. Knowledge bases in the form of ontologies enable relating military unit types to equipment, to vehicles, to sensors, to weapons, to munitions, and more, for use by simulation planning tools, simulation environment design tools, scenario tools, and more.

- *Creating composed simulation services using conceptual models as metadata*. By annotating simulation services by entity type (e.g., M1A1 tank) and interaction elements from conceptual models (e.g., damage states or probability hit/kill), the effort to catalog, search for, and assess simulation services for composition can be further simplified and automated. Semantic knowledge bases of entity types, states, and interactions provide the foundational vocabulary for tagging[5] and describing simulation services in terms of conceptual models.

- *Simulation asset management*. Similar to tagging simulation services by conceptual model, management of other simulation assets can improved in a similar manner. Scenarios can be tagged by purpose, 3D models can tagged by the entity type portrayed, renderers (displayed from a sensor) and renderings (the heat signature of a vehicle through a non-visual sensor) can all be cataloged by entity types, behaviors, and observables as defined by a conceptual models in a knowledge base.

The data services categories above are enablers of greater automation and availability of simulations through MSaaS. Integration of simulation planning tools, conceptual analysis tools, composition tools, simulation services, authoritative data sources, and post-simulation analysis tools depends on data management and semantic linking of concepts and services across the simulation life cycle.

5 Composition

Composition occurs at several levels of abstraction; in particular in steps 2–4 of the MSaaS engineering process (Figure 5). Developing a composed simulation service (or *composition* for short) therefore requires an understanding of the involved simulation services, the functionality each simulation service provides, and the way the simulation services interoperate at various levels of abstraction (Figure 3): implementation-independent, simulation architecture-specific and implementation specific. Several Composition services would therefore use service descriptions at these levels of abstraction to determine which simulation services to use and how to configure them.

When composing at the simulation architecture-specific level, more detailed model information is needed, but also more technical information related to the simulation architecture itself in order to determine if services will (technically) fit in the chosen simulation architecture.

When composing at the implementation-specific level, composition services will rely on implementation-specific information in simulation service descriptions that include details about how the simulation service integrates with other services, such as protocols, object models, and pragmatic information such as information publications frequencies. This also information regarding how services can be configured and executed and information on the (technical) orchestration of simulation services.

Composition services also need information about the available computing resources in the MSaaS infrastructure, whether they be in a cloud environment, local set of servers, specific local personal computers or mobile devices. This information should be provided by the SMC services (Figure 7). Moreover, compositions may be deployed in various ways; e.g., in terms of stand-alone, single data center or multiple data centers [62].

Most existing distributed simulation environments typically have many engineers on staff who must individually manage each simulation component. This takes a lot of time and resources and is more susceptible to errors than an automated process. By using tools to understand technical, functional, and scenario details about the available simulation services, the resulting simulation environment can be deployed, configured, initialized, and executed with a lesser effort and time. The goal of having designated composition services is to reduce the time, errors, and amount of resources required for composing simulation services within a simulation life cycle.

5.1 Simulation composition applications

These applications are the front-end user interfaces simulation operators (Figure 1) would use to compose simulation services.

5.1.1 Simulation supplier applications. This is the interface used by service suppliers to provide their service and all the necessary information about that service during composition activities. Once simulation suppliers provide the information about their services, the application should allow the suppliers or the simulation operators to provide all the necessary metadata required to compose services correctly based on the goals for the composed simulation service (Section 5.1.2). This metadata includes the simulation service descriptions to be used for composition at the implementation-independent level and for the benefit of composition services.

The supplier should be able to upload their service description and service realization in software in multiple ways as well as provide appropriate registry locations. Various service realization formats should be permissible to allow for supplier flexibility: The supplier should be able to provide (through a link or direct upload) an executable, a container, or virtual machine. In the cases that the software needs to be compiled once configuration information is provided, the supplier should provide a script to do so rather than relying on the MSaaS components to manage compilation.

5.1.2 Simulation integration applications. These user applications allows an integration engineer (a simulation operator with a more technical focus) to manage simulation services and their metadata at various levels of resolution in service descriptions. The metadata managed here includes information from all four levels of abstraction (Figure 3). The integration engineer can manage that metadata within the context of the organization's taxonomy and ontologies. In some cases, new services introduced into the environment by suppliers may require adjustment to the overarching taxonomy of capabilities and domain ontologies. For example, when a higher resolution service is introduced, the capability that it provides may need more detail within the appropriate ontology. The integration engineer should be able to visualize and adjust the operational ontology and link services to the elements within that ontology.

The integration engineers will also need to verify accuracy of all other metadata and be the users responsible for accuracy of the data and how the services are linked within the overarching MSaaS system.

Finally the simulation integration engineer user application should allow the integration engineer to manage what simulation operators see (Section 5.1.3), so they can navigate through the MSaaS cloud environment to find the simulation services they require. The level of detail seen by simulation operators is important to make the system easy to use. Simulation operators should only see simulation services at a level that makes sense to them and provides descriptive details about what they can execute at the level of abstraction the simulation operator is currently working at. For example, when simulation operators are trying to find simulation services to provide ground units, they should not have to work through details about the resolution of the armor on those ground units until required at lower levels of abstraction.

5.1.3 Simulation scenario applications. Determining the concrete simulation scenario (in terms of which entities, events, terrain, etc. are involved) has implications on the entire simulation environment. In some cases, it may be appropriate that simulation engineers create and manage the scenarios and then provide access to them via the simulation operators. In other cases, a simulation operator may need the ability to set up the scenarios, in which case, these user applications should provide an easy to use scenario creation/editing interface. Every scenario has engineering-level decisions to make and these decisions are not trivial requiring decisions about computing resources

and service responsibility. For example, there may be areas for high resolution versus low resolution focus within the spatial or capability specific regions. When scenarios will be managed by simulation operators, then the user interface for the simulation integration engineers (Section 5.1.2) will need to include workflow and user interfaces to see the scenarios, and build out all the low level functional and technical details for that scenario to execute prior to the actual execution.

Note that this discussion deals with handling scenarios, while the activities of *generating* scenarios as described in the SISO guidelines for scenario generation[42] sorts under what the MSaaS reference architecture[4] calls "Modelling Services/Applications" which are outside this article's scope. For future deliberations, one should keep in mind that scenarios are tightly coupled to the applications that generate them, as each application might operate their own scenario format, data requirements and modeling capabilities. There is a need for standardized scenario formats. In the mean time, composition must accommodate several possible formats.

5.2 Simulation composition services

Simulation composition services are the back-end to the simulation composition applications. These services must use service descriptions for available simulation services to determine which services should be used, and how they should be configured to provide the appropriate functionality and execute the desired simulation. The composition services then provide the composition to the SMC services (Section 6) with all the metadata necessary to deploy and execute the composed simulation service.

Simulation composition services have a formidable job. They have to use all three elements of service descriptions at various levels of abstraction (Section 2.3.5) to align the following cross-cutting interoperability concerns between simulation services:

Technical interoperability concern refers to syntactic, semantic and pragmatic interoperability (Section 2.2) at the implementation-specific level (Figure 3). This involves the ability for services to communicate over the network, using a common protocol and Application Programmers Interface (API), and with the same syntax (including encoding/decoding of information). This also includes communication agreements such as message frequencies, dead-reckoning agreements, network optimization strategies such as Data Distribution Management (DDM)[28], and any details that are specific to the technical implementation of the simulation environment.

Functional interoperability concern refers to syntactic, semantic and pragmatic interoperability at the simulation architecture-specific and implementation-independent levels (Figure 3). This involves to assess that the candidate simulation services have the appropriate functional capabilities; that is, that what they represent within the simulation environment (the model part of the service description) aligns well. This concerns the forces being simulated, the fidelity (accuracy), the resolution (level of detail), and the interactions between those entities are suitable.

Scenario interoperability concern refers to configuring syntactic, semantic and pragmatic interoperability to a specific scenario for a given simulation life cycle at the simulation architecture-specific and implementation-dependent levels. This involves the assessing that simulation services are able to cooperate to represent a specific scenario within the simulation: Simulation services need to be synchronized on what the simulated entities will be doing and aligned on the data sets being used for those entities. Model responsibilities need to be delegated across services for reasons including scaling, visualization, user interaction (e.g. human players controlling units on specific workstations), entity capabilities (i.e. tank modeling services owning tanks while aviation services owning aircraft). The data sets could also be varied across different scenarios using different classification of data, different performance data of the entities, or automated behaviors.

In order to compose simulation services at the various abstraction levels, the MSaaS engineering process (Figure 5) must ensure that the chosen simulation services are able to interoperate together to achieve the desired federated capabilities. At the implementation-dependent level, this boils down to simulation components interoperating in a simulation environment. In the following we describe the functionality of composition services according to what abstraction level of simulation service description they operate on, taking into account the three cross-cutting interoperability aspects above. This sheds light on what information is needed in simulation service descriptions at the various abstraction levels.

5.2.1 Architecture-agnostic composition services. The simulation service descriptions (Section 2.2) must hold enough information to determine which services can be composed[72] to meet the desired conceptual scenario[42] and to determine if their abstract interfaces, patterns of interaction and semantics are compatible. For example a service providing a sensor capability, will be defined in a way to show its reliance on other data such as ground truth information about entities, which are then provided by a different service. These two services can be provisionally

composed based on their service descriptions until further details are examined. This composition is then further refined at the simulation architecture-specific level.

5.2.2 Architecture-aware composition services. Composition services will here need to determine if the services chosen above can work together at the architecture-specific level, given architecture-specific information. This metadata must include the simulation middleware protocols used, the object models used, and any pragmatic agreements such as dead reckoning or interest management. Other information that is related to the architecture, for example, communication methods, must be available for the services.

5.2.3 Implementation-aware composition services. Implementation-specific composition considers the lowest level details of the service functionality including modeling details such as the resolution and fidelity details of the entities and relationships being represented. When two executable models are integrated, it is important that the interfaces are appropriate to the desired level of resolution (detail) in order to avoid data mismatches, translation errors, and poor assumptions leading to a lack of interoperability. This level of composition requires details about the service at the lowest levels. The suppliers are in the best position to provide this information, but they must also align that information with the implementation-independent ontology-based description (Figure 3).

5.2.4 Further architecture and implementation-dependent metadata. Service descriptions at lower levels of abstraction must be quite detailed. Necessary deployment metadata includes all the information required by SMC to deploy and execute the service. This information includes operating system, computing footprint, licensing details, and security constraints[73]. Configuration metadata includes all the information regarding how a service can be configured including both *what* is being configured and *how* those items are being configured. Example configuration mechanisms include environment variables, command line parameters, configuration file changes, or even user interface actions which can be automated with tools like Sikuli[†].

Both deployment and configuration aspects span across all simulation components including middleware software (e.g. HLA RTI), gateways, management tools, and after action review (AAR) tools.

The aspects that can be configured span across technical, functional, and scenario details. There can be overlap across those three areas. The scenario responsibilities of a service have a direct impact on the technical configuration requirements of the service. For example, the computational resources would be higher if the number of entities represented within the scenario is higher. If a service can execute at varying levels of resolution, then the computing footprint could differ depending on the simulation's functional representation requirements. As the scenario size changes, the modeling responsibilities of each service could change requiring different levels of computing power and memory. The information is quite often in sets or ranges. For example, a service could run on multiple operating systems, could use a range of memory sizes or processing power, and those details could depend on the capabilities required or the size of the scenario.

Services do not have to have separate service implementations for each simulation protocol, operating system, code platform, etc. Instead, services can have can have multiple modes (that need to be configured) including:

- Multiple protocols for communication (e.g. HLA, DIS, TENA, DDS, etc.)
- Multiple operating system choices
- Multiple sets of data (force structures, entity representation variants, data classification, etc.)
- Multiple modes (e.g. representing the entire entity versus allowing external services to represent portions of an entity
- Multiple user interface options including running headless, having a ground truth view, or being a simulation user station (e.g. virtual interface)

Determining the services required to provide a simulation capability is based on the functional capabilities of the components and the hierarchy and relationships of the functionality.

Breaking down a high level functional capability into its lower level atomic parts and then representing the relationships between those atomic parts allows for components to be mapped to that functional decomposition and then chosen as the simulation environment goals are identified in the form of high level functional capabilities.

When a simulation user identifies the functional capabilities required, they should not presented with low level considerations unless they choose to go to that depth. The user will most likely want to focus on high level capabilities, such as picking the force structure, the terrain location, and some high level actions. There may be many choices for which components can be used, each with their own advantages and disadvantages. The user may care about some of the functional and scenario options of the services. This implies that the composition service needs to provide a user interface based on the functional selection of services including any options that the user may want to

[†] http://doc.sikuli.org/

choose including data, model pedigree, modeling resolution of certain aspects of the simulation, and other details. In that case, the available component choices should be available to be displayed to the user based on the metadata available.

The composition services have the role of providing service metadata to the SMC services in order to setup and manage the composed simulation service. The information required by SMC includes all the details required to understand service capabilities, requirements, and execution details.

The descriptive elements for service metadata include:

- Repository location: where to find the service application (virtual machine, container, executable, and/or configuration files).
- Version of the service: multiple versions could exist within the repository, and each could be used at different times depending on the capabilities (technical, functional, or scenario).
- Dependencies: there are many dependencies a service could have including operating system, libraries, network configuration (e.g. ports open), and anything required to be there for the service to execute properly.
- Execution initiation information: how to start the service including parameters such as environment variables, command line parameter, or a script to execute that handles all of the above mentioned configuration items such as:
  - Scenario configuration: force structure, modeling responsibilities, and anything else that denotes the service's responsibility during the simulation and what it should execute regarding the scenario.
  - Technical configuration: the technical implementation of the service could potentially be different for varying simulations. The scale of the service's responsibilities could differ between scenarios. The protocols that the service uses may differ across simulations that use different architectures. Any technical information that the service requires to run could differ across simulations and must be configured accordingly.
- Tags and/or labels for features and capabilities of the service to be used to find the appropriate service for each simulation.
- Data artifact information: data generated and logged/stored from each simulation should be retrievable after the simulation. The data could be in any form (e.g. database file, set of files, database located locally or distributed, graphics created, etc.). The composition service should record what data will be generated (metadata), the data type (file, set of files, etc), its location, and in some cases a script to execute in order to obtain data
- Streaming mechanism: services may include a graphical display that needs to be streamed to users during the simulation, or recorded for viewing after the simulation. Streaming of the display can be done generically from a virtual machine configuration using a streaming program such as VideoLAN Streaming Solution[‡] or Apache Guacamole[§].

Service implementations within a distributed simulation environment should not be limited to be one type of software executable. Services could range in implementation to being a binary executable with accompanying configuration files that needs to be deployed to a container that can be deployed using a container management tool like Kubernetes[¶], a virtual machine that needs to be delivered to and executed within a hypervisor or virtual machine monitor. The service implementation details across this range of execution variations will need to be accounted for in the metadata provided to SMC. Obviously, simplifications can be made when using one type of software executable, like containers.

SMC will be responsible for the deployment, configuration, execution, monitoring, and management of the services using the application details captured within the composition services. As a part of the systems engineering and design of a simulation environment, it should be decided which execution types (virtual machines, containers, binaries, etc.) can be included and therefore need to be accounted for within the composition services.

6 Service management and control (SMC)

The NATO C3 Taxonomy defines *Service Management and Control* (SMC) as a collection of capabilities to coherently manage components in a federated service-enabled information technology infrastructure. In this definition, SMC concerns policies, human processes and procedures, as well as computer and information systems (CIS) capabilities to manage components. This is in line with IT Service Management (ITSM), a term used for frameworks such as COBIT (Control Objectives for Information and related Technology) and ITIL (Information Technology Infrastructure Library).

[‡]https://www.videolan.org/vlc/streaming.html

[§]https://guacamole.apache.org/

[¶]https://kubernetes.io/

For MSaaS, the policy and procedure side of SMC is to some extent addressed by the MSaaS Governance Policies[74], the MSaaS Operational Concept[75], and the MSaaS engineering process[43]. Our focus here is on CIS capabilities.

In the following, we elaborate on SMC to manage simulation services and composed simulation services in an MSaaS infrastructure. The focus will be on the *Execute* capabilities of the MSaaS portal (Figure 1); that is, the means to start and stop composed simulation services (compositions), to monitor compositions, and to test if compositions function as required. The capabilities are described at an implementation-independent level as SMC Applications and SMC Services, and several concrete examples of such services are provided.

6.1 Simulation SMC user applications

Simulation SMC user applications are the front-end user interface of the Simulation Operators (Figure 1) to the back-end SMC Services. The main SMC capabilities provided are *Simulation Service Test Management*, and *Simulation SMC*; recall Figure 7.

6.1.1 Simulation Service Test Management user applications. These user applications enable the Simulation Operator to ensure that the various (simulation) services function properly; that is, with compliance to agreed service interfaces and contracts. The correct functioning of a (composed) simulation service is tested before it is entered in the service registry, but testing typically occurs throughout the service lifetime; for example, for particular training exercises.

Service testing can be performed at various steps of the MSaaS engineering process (Figure 5) and corresponding levels of service abstraction (Figure 3).

At the implementation-independent level, (composed) simulation services can be tested using, for example, a framework that provides methods, techniques and tool support for verifying composability at a syntactic and semantic level of simulation interoperability[72]. The proposed framework describes a verification process for model composition and uses BOMs with several semantic extensions.

At the simulation architecture-specific level there are several applications available for testing. Examples of such applications are the NATO Integration Verification and Certification Tool (IVCT)[‖] and the Joint Exercise Control Suite (JECS)[**]. The NATO IVCT is an application to test the interoperability capabilities of HLA simulation components and to support the integration of distributed simulations. The application consists of several software components, including a web-based front-end with several back-end components (i.e., test services; see Section 6.2.4) such as a test case execution engine and a set of executable test cases for (NATO Education and Training Network) NETN FOM modules[76]. The test case engine provides an API for adding in additional test cases. The tool is open source and was provided to NATO as an Initial Operational Capability (IOC) for HLA certification in 2017. JECS supports the testing of, among others, Joint Live Virtual and Constructive (JLVC) training events. JECS includes several tools such as the Joint Simulation Protocol Analyzer (JSPA) and Joint Master Enumeration Manager (JMEM) and the Joint Analysis Workstation (JAWS). JSPA manages data flows among simulation components to support trouble shooting during exercise test and execution. JMEM checks platform names, weapon effects and sensor codes (i.e., entity type values) used by the simulation components to support consistent communication exchanges among the various components. JAWS provides a replay of interactions among simulation components.

To enable a greater degree of automated service testing, simulation services need to be consistently described (Section 2.2), preferably in a machine readable format. For example, the BOM format[24] mentioned earlier caters for several aspects of service descriptions; in particular patterns of interplay with other simulation services. The information in the BOM is input to the construction of test models for testing simulation services at a simulation-architecture specific level[77]. Although the construction of test models is, in practice, a largely manual step, more automation may be possible in the future through Model Based Testing Tools.

Simulation Service Test Management user applications are described as a separate capability, and current implementations are often dedicated applications for particular simulation technology. However, our vision is that these applications should become more integrated with the MSaaS portal and the Simulation SMC user applications, described in the next subsection.

6.1.2 Simulation SMC user applications. These user applications enable the simulation operator to view available composed simulation services (i.e. compositions) that meet certain objectives; to select an appropriate composition for execution; to reserve resources for the execution of a composition; to provide input parameters and start a selected composition; and to monitor, manage and control previously started compositions. Each composition may be implemented with different simulation technologies and use different simulation architecture topologies.

When the simulation operator determines the composition he wants to execute, some follow-up choices should be presented to inform the application of when and how

[‖] https://www.mscoe.org/nato-hla-certification-ivct/

[**] JECS is a trademark of Knight Federal Solutions

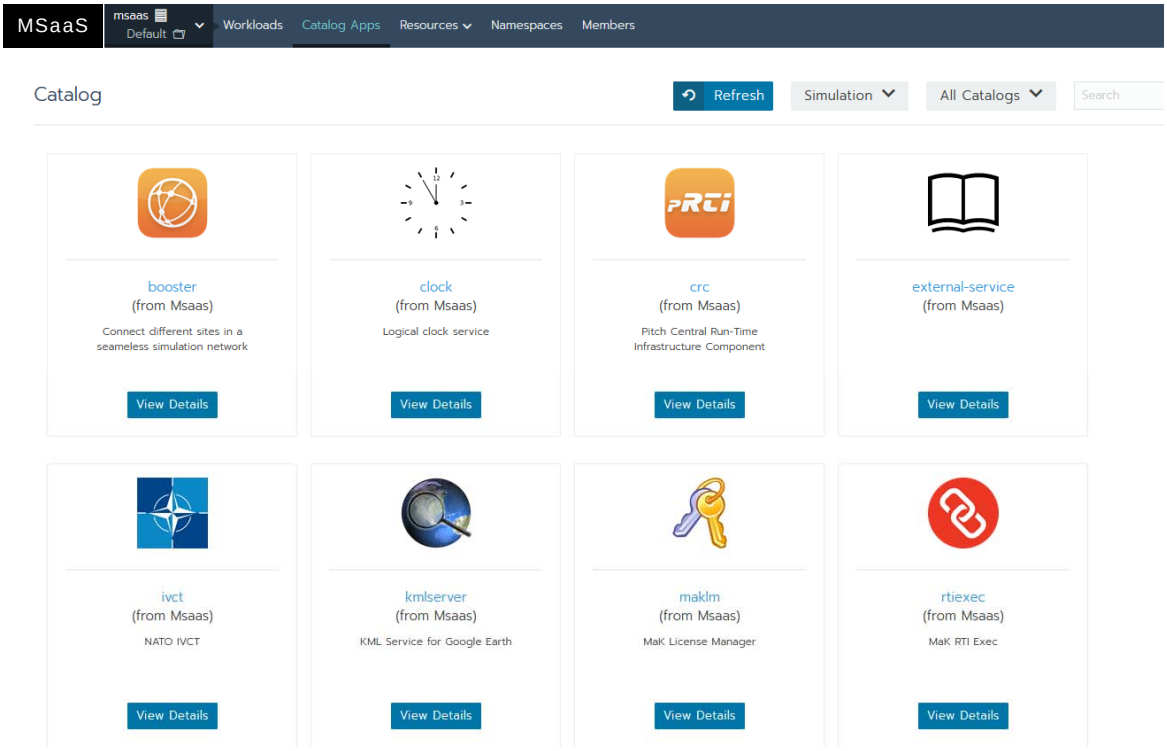

Figure 8. Kubernetes search and discovery user interface for Helm Chart repositories.

the composition should be deployed and executed. The simulation operator should be able to schedule the execution for any time he chooses, albeit there could be some resource contention issues. When resources are reserved, they should become unavailable for other users, and that information should be displayed to the simulation operator. Other information that a simulation operator should be able to provide, is a description for other users to see for the purposes of reuse, privacy information (who should have access to the composition and resulting data), and use case/organization-specific information. For example, when used for training, the user application may provide information on unit availability and a workflow for scenario generation/changes.

These applications provide, among other, functionality to start and stop (composed) simulation services, discover available services, provide reports on service and resource usage, provide dashboards and statistics charts, manage exceptions and manage service metadata. The applications interact with the underlying technical services to provision (virtualized) resources and to obtain the required performance data, services metadata, composition data, and so on. In large part, this category of application exposes to simulation operators the service execution capability of the envisioned MSaaS portal.

It has already been noted[38] that when using container technology, the required functionality for Simulation SMC user applications can largely be provided by current container orchestration environments such as Kubernetes. Kubernetes is an open source product for automating deployment, scaling and management of containerized applications; in our case, (composed) simulation services. The Kubernetes user interface is a web based front-end for back-end SMC functionality. The Simulation Service Test Management user applications described above may be deployed using the same container orchestration environment used for the regular (composed) simulation services. For example, the NATO IVCT is available in the form of Docker[††] container images and can be deployed in such a container orchestration environment, thereby becoming part of the MSaaS portal together with the simulation services. This way a (composed) simulation service can be tested in the same environment as where it is used. Compositions can be started and stopped via a dashboard; for example, Monocular[‡‡]; see Figure 8 where each icon on the dashboard represents a composition in the form of a Kubernetes Helm Chart[§§], stored in a (Git) repository.

[††] https://www.docker.com/
[‡‡] https://github.com/helm/monocular
[§§] https://helm.sh/

Dette er en postprint-versjon/This is a postprint version.
Publisert versjon/Publised version: 10.1177/1548512919896855

6.2 Simulation SMC Services

Simulation SMC Services are the back-end services of the front-end Simulation SMC Applications (Figure 7). We distinguish between SMC services that need to be aware of the simulation architecture of a particular composition, and services that function regardless of the selected simulation architecture. We start with architecture-agnostic services.

6.2.1 Architecture-agnostic simulation monitoring, metering and logging services. In terms of Section 2.3 applied to infrastructure services, this entails that the implementation-independent service descriptions of these services do not need a simulation architecture-dependent refinement and do not follow the path of implementation-specificity of simulation services. However, they would have other architecture- or implementation-specific service description refinements on their path to implementations. We use three terms to characterize this kind of service:

- *Monitoring*. Monitoring Services monitor service communication based on service calls and message exchanges to identify performance issues and determine current availability. Examples are the rate at which DIS entity state PDUs are exchanged, the occurrence of specific events such as weapon fire and munition detonation events in an entity

Table 1. Potential implementations of some Monitoring, Metering and Logging Services

Monitoring	Datadog `https://hub.docker.com/r/datadog/agent`
Monitoring	Graphite `https://github.com/graphite-project/graphite-web`
Monitoring	Scout `https://scoutapp.com`
Monitoring	Prometheus `https://hub.docker.com/r/prom/Prometheus`
Monitoring	Grafana `https://hub.docker.com/r/grafana/Grafana`
Monitoring	EFK = Elasticsearch + FluentBit + Kibana `https://docs.fluentd.org/v0.12/articles/docker-logging-efk-compose`
Monitoring	ELK = Elasticsearch + Logstash + Kibana `https://hub.docker.com/r/sebp/elk`
Monitoring	Graylog `https://hub.docker.com/r/graylog/graylog`
Monitoring	Grafana `https://hub.docker.com/r/grafana/grafana`
Metering	Collectd `https://collectd.org`
Metering	Telegraf `https://hub.docker.com/\_/telegraf`
Metering	cAdvisor `https://github.com/google/cadvisor`
Metering	Influxdb `https://hub.docker.com/_/influxdb`
Metering	Sensu `https://sensu.io`
Metering	Sematext `https://sematext.com`
Logging	Syslog (standard Linux tool)
Logging	Fluentd `https://hub.docker.com/r/fluent/fluentd`
Logging	Filebeat `https://hub.docker.com/r/elastic/filebeat`
Logging	Fluent-bit `https://hub.docker.com/r/fluent/fluent-bit`
Logging	Metricbeat `https://hub.docker.com/r/elastic/metricbeat`

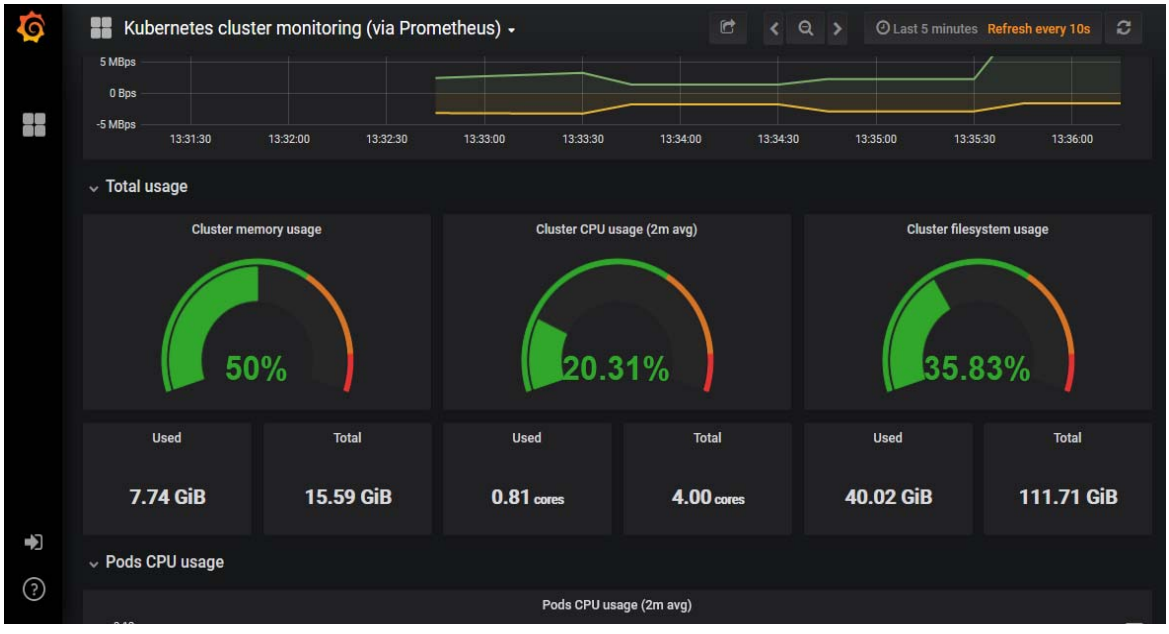

Figure 9. Kubernetes cluster, node and service monitoring using Prometheus and Grafana.

Dette er en postprint-versjon/This is a postprint version.
Publisert versjon/Publised version: 10.1177/1548512919896855

level engagement simulation, or events that indicate simulation execution state changes.

- *Metering*. Metering Services measure levels of resource utilization by individual simulation services such as the number of HLA-RTI calls and callbacks, number of DIS messages received or sent, number of get/set operations on a simulation state database, number of simulation entities managed/owned by a service, etc. The measured values can be used to determine if sufficient resources are allocated to run the service.

- *Logging*. Logging Services provide functionality for capturing, filtering and writing the data collected through monitoring and metering. The resulting logs can be used for auditing purposes, troubleshooting, performance optimizations, etc. The data should be combined with metrics from Core Services[71], such as memory and CPU usage.

Often these capabilities are mixed and service implementations may provide several or all of the services; see Table 1 for examples. Most of these tools provide both containerized and non-containerized installation options for different platforms, and are suitable for use in an MSaaS infrastructure. Figure 9 shows an example of a Prometheus and Grafana[¶¶] dashboard for Kubernetes cluster monitoring. Via this dashboard the whole cluster, nodes, or individual services can be monitored (e.g. memory usage, disk space usage, network usage). This is an example of an architecture-agnostic service, since it is not aware of the simulation architecture used in the compositions(s) that run in the cluster. For more information about what is happening inside a composition we need simulation architecture-aware SMC Services.

6.2.2 Architecture-aware simulation monitoring, metering and logging services. These services mimic the similarly-named architecture-agnostic services, but are aware of the simulation architecture used in a composition in order to collect simulation data from the composition. Architecture awareness is achieved through the provisioning of specific adapters that can hook into a composition and collect the data. Examples of simulation architecture-aware logging services are HLA or DIS data recording services. Also several of the tools listed in Table 1 provide the possibility to add simulation architecture specific adapters.

In some instances the monitoring, metering and logging services may be part of the composition itself. This entails that the service description abstraction levels of these simulation SMC services follow the service description abstraction levels of the composed simulation services on which they operate. Following Section 2.3 also for infrastructure services, the simulation SMC services would have service descriptions at the implementation-agnostic level, but since they need to be simulation architecture aware, they would also have simulation architecture-dependent descriptions.

6.2.3 Architecture-aware simulation control services. These services provide the capability to control the simulation execution state of a (composed) simulation service. Similarly to the Monitoring, Metering and Logging services, the Simulation Control Services may be simulation architecture-aware and possibly part of the simulation composition, rather than an architecture-independent SMC Service in the MSaaS infrastructure. Simulation Control Services also provide the capability to orchestrate the initiation and termination of the simulation services within a composition. A few examples of Simulation Control Services can be found in literature[70;78–80].

6.2.4 Architecture-aware simulation test services. These services are the back-end technical services of the Simulation Service Test Management User Applications (Figure 7). These services are simulation architecture-aware, and they test if compositions comply with agreed service interfaces and contracts. Monitoring, Metering and Logging Services may be needed to assist the Test Services in collecting data; for example metrics on HLA-RTI interface usage. Simulation Control Services can also be used to orchestrate the initiation and termination of Test Services.

In case of the NATO IVCT, the Test Services are provided by a single back-end (containerized) component, the IVCT *Test Case Runner*. The IVCT Test Case Runner executes the configured test cases against the system under test. In the context of MSaaS, this can be a non-cloud based simulation component or environment, or a (composed) simulation service located in the MSaaS infrastructure. Test cases are packaged in Docker container images and provided to the Test Case Runner for execution. The Test Case Runner is operated via a web-based front-end (Section 6.1.1).

6.2.5 Provisioning Services. Provisioning Services manage the instantiation, runtime management and disposal of dynamically scalable and virtualized infrastructure resources. Provisioning Services use the results of the Composition Services to create the declared resources as defined in the composition description. To do this, the description of a composition must contain sufficient technical information for the provisioning and deployment of resources (see Section 5 for more information). In the example of a container orchestration environment such as Kubernetes or Docker, the virtualized infrastructure resources are containers, overlay networks and data

[¶¶] https://prometheus.io/docs/visualization/grafana/

volumes, and the provisioning information is provided as a Helm Chart in Kubernetes or Docker Compose file in Docker. Provisioning services are simulation architecture agnostic.

6.2.6 Service Discovery Services. These services provide information about (composed) simulation services currently executing, and information about composed simulation services available for execution in the MSaaS infrastructure. This deals with run-time discovery as opposed to design-time discovery (Section 4.1.2). To continue the example with Kubernetes, the status of executing compositions can be inspected via the Kubernetes Dashboard, and available compositions can be queried via Monocular, a web-based application that enables the search and discovery of Helm Charts from multiple Helm Chart repositories. These services are simulation architecture agnostic.

6.2.7 Other applications and services relevant to SMC. The sections above elaborated on a number of simulation SMC Applications and Services relevant to MSaaS. An MSaaS infrastructure will generally rely on further non simulation-particular capabilities to those discussed here. Examples would be Storage Services, Database Services, Infrastructure Providing Services; many of which are at the level of infrastructure as a service (IaaS) and platform as a service (PaaS).

7 Final remarks

Earlier discussions[81] have remarked that the notion of "reference architecture" should be reserved for blueprint architectures that contain more detail than what is perhaps commonly the case for many so-called reference architectures. The efforts in the preceding discussion should contribute to making the MSaaS reference architecture[4] more specific and useful to developers, service providers and service consumers than it is in its current form, which is more in the form of an overarching architecture[81].

By elaborating on the essential MSaaS infrastructure capabilities; that is, simulation data management capabilities, simulation composition capabilities and simulation service management and control capabilities, we hope to shed light on how simulation services and composed simulation services can be discovered, composed and executed in practice, using implementation-independent simulation service descriptions at design time and implementation-specific service description at implementation time.

The *functionality* needed for the MSaaS infrastructure is, in many cases, provided by existing platforms and frameworks. However, as the elaborations on MSaaS infrastructure capabilities in this article indicate, it is necessary to offer that functionality as services to fulfill the MSaaS vision and to provide the MSaaS portal functionality on various platforms. Infrastructure functionality as services entails that one uses the same principles of service description abstraction levels as we put forth for simulation services.

We also found that MSaaS infrastructure services in some instances are simulation services and become part of a composed simulation service. Then, these infrastructure services would have the same service description abstraction levels that (composed) simulation services have. However, in other instances, MSaaS infrastructure services are supporting services that do not adhere to simulation architecture. In the latter case, they would have a different service description abstraction structure from that of simulation services. Further, such infrastructure services might not need to be designed to be composed rapidly and readily to the same extent as simulation services. Indeed, it may not even be necessary to formulate that functionality in terms of services. However, to conform with other frameworks, such as the NATO Consultation, Command and Control (C3) Taxonomy, the MSaaS reference architecture does formulate this more stable functionality as services. Understanding the distinction between these different types of infrastructure service should help to prioritize what infrastructure functionality to provide as services first.

Future work will elaborate further on MSaaS infrastructure capabilities. In particular, it is essential to understand how infrastructure services differ in content and roles as expressed in topologies for different simulation architectures. Indeed, the contents and roles of MSaaS infrastructure services might to some degree define the various simulation architectures. This is especially interesting when investigating newer edge and fog architectures applied to simulations.

Acknowledgements

The authors are grateful to members of the NATO Task Groups MSG-136 and MSG-164 on Modeling and Simulation as a Service (MSaaS) for valuable insight and discussions.

References

1. NATO Modelling and Simulation Group. *NATO Modelling and Simulation Master Plan (version 2.0)*, 2012. Doc. no. AC/323/NMSG(2012)-015.
2. NATO Allied Command Transformation. *Federated Mission Networking (FMN)*, 2015. Accessed November 2015.
3. van den Berg TW, Huiskamp W, Siegfried R et al. Modelling and Simulation as a Service: Rapid deployment of interoperable and credible simulation environments – an overview of NATO MSG-136. In *Proc. 2017 Fall Simulation Innovation Workshop*. 17F-SIW-018.
4. Hannay JE and van den Berg TW. The NATO MSG-136 Reference Architecture for M&S as a Service. In

Proc. NATO Modelling and Simulation Group Symp. on M&S Technologies and Standards for Enabling Alliance Interoperability and Pervasive M&S Applications (STO-MP-MSG-149).
5. Taylor SJE, Khan A, Morse KL et al. Grand challenges for modeling and simulation: simulation everywhere—from cyberinfrastructure to clouds to citizens. *Simulation* 2015; 91(7): 648–665.
6. Kratzke N. A brief history of cloud application architectures. *Appl Sci* 2018; 8(1368).
7. Kratzke N and Quint PC. Understanding cloud-native applications after 10 years of cloud computing – A systematic mapping study. *J Systems and Software* 2017; 126: 1–16.
8. Gregor S. The nature of theory in information systems. *MIS Quarterly* 2006; 30(3): 611–642.
9. Hannay JE, Sjøberg DIK and Dybå T. A systematic review of theory use in software engineering experiments. *IEEE Trans Software Eng* 2007; 33: 87–107.
10. Lewin K. The research center for group dynamics at Massachusetts Institute of Technology. *Sociometry* 1945; 8: 126–135.
11. Whetten DA. What constitutes a theoretical contribution. *Academy of Management Review* 1989; 14(4): 490–495.
12. Davis MS. That's interesting! towards a phenomenology of sociology and a sociology of phenomenology. *Philosophy of the Social Sciences* 1971; 1: 309–344.
13. Bacharach SB. Organizational theories: Some criteria for evaluation. *Academy of Management Review* 1989; 14(4): 496–515.
14. Weick KE. What theory is *not*, theorizing *is*. *Administrative Science Quarterly* 1995; 40: 385–390.
15. Jarvis P. *The Practitioner-Researcher*. Jossey-Bass Publishers, 1999.
16. Gorschek T, Garre P, Larsson S et al. A model for technology transfer in practice. *IEEE Software* 2006; 23: 88–95.
17. Erl T. *SOA principles of Service Design*. Prentice Hall, 2007.
18. The Open Group. *SOA Ontology, Version 2.0 Open Group Standard*, 2014. Doc. no. C144.
19. Tolk A. Standards for distributed simulation. In Tolk A (ed.) *Engineering Principles of Combat Modeling and Distributed Simulation*, chapter 12. Wiley, 2012. pp. 209–241.
20. Hannay JE, Brathen K and Mevassvik OM. Agile requirements handling in a service-oriented taxonomy of capabilities. *Requirements Engineering* 2017; 22(2): 289–314.
21. Singapogu SS, Gupton K and Schade U. The role of ontology in C2SIM. In *Proc. 21st International Command and Control Research and Technology Symposium (ICCRTS 2016)*.
22. Simulation Interoperability Standards Organization. *The Command and Control Systems – Simulation Systems Interoperation (C2SIM) Product Development Group (PDG) and Product Support Group (PSG)*, 2014. Accessed June 2015.
23. Durak U, Oğuztüzün H, Köksal Algin C et al. Towards interoperable and composable trajectory simulations: an ontology-based approach. *J Simulations* 2011; 5(3): 217–229.
24. Simulation Interoperability Standards Organization. *SISO-STD-003-2006 – Standard for Base Object Model (BOM) Template Specification*, 2006.
25. Mojtahed V, Svee EO and Zdravkovic J. Semantic enhancements when designing a BOM-based conceptual model repository. In *Proc. 2010 European Simulation Interoperability Workshop (SIW)*. Simulation Interoperability Standards Organization.
26. Mojtahed V, Andersson B, Kabilan V et al. BOM++, a semantically enriched BOM. In *Proc. 2008 Spring Simulation Interoperability Workshop (SIW)*. Simulation Interoperability Standards Organization.
27. Moradi F, Ayani R, Mokarizadeh S et al. A rule-based approach to syntactic and semantic composition of BOMs. In *Proc. 11th IEEE Int'l Symp. Distributed Simulation and Real-Time Applications (DS-RT 2007)*. IEEE Computer Society, pp. 145–155.
28. IEEE Standards Association. *1516-2010 – IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)*, 2010.
29. IEEE Standards Association. *1278.2–2015 – IEEE Standard for Distributed Interactive Simulation (DIS) – Communication Services and Profiles*, 2015.
30. Test Resource and Management Center. *TENA The Test and Training Enabling Architecture – Architecture Reference Document, Version 2002*, 2002.
31. World Wide Web Consortium. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, 2007.
32. World Wide Web Consortium. *Web Services Semantics (WSDL-S) Version 1.0*, 2005.
33. World Wide Web Consortium. *Semantic Annotations for WSDL and XML Schema*, 2007.
34. World Wide Web Consortium. *Web Services Architecture – W3C Working Group Note*, 2004.
35. World Wide Web Consortium. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, 2007.
36. World Wide Web Consortium. *Web Application Description Language (WADL) Submission*, 2009.
37. Fielding RT and Taylor RN. Principled design of the modern web architecture. *ACM Transactions on Internet Technology* 2002; 2(2): 115–150.
38. van den Berg TW and Cramp A. Container orchestration environments for M&S. In *Proc. 2017 Fall Simulation Innovation Workshop*. 17F-SIW-006.
39. Tolt G, Hedström J, Bruvoll S et al. Multi-aspect path planning for enhanced ground combat simulation. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. pp. 1–8. DOI:10.1109/SSCI.2017.8280886.
40. de Reus N, van den Berg TW, Janssen H et al. Lessons learned from leveraging Simulation as a Service in Viking18.

In *Proc. Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2018*. National Training and Simulation Association.

41. Bruvoll S, Hannay JE, Svendsen GK et al. Simulation-supported wargaming for analysis of plans. In *Proc. NATO Modelling and Simulation Group Symp. on M&S Support to Operational Tasks Including War Gaming, Logistics, Cyber Defence (STO-MP-MSG-133)*.
42. Simulation Interoperability Standards Organization. *SISO-GUIDE-006-2018 – Guideline on Scenario Development for Simulation Environments*, 2018.
43. NATO Science and Technology Organization. MSaaS Refernece Architecture: Volume 3—MSaaS Engineering Process. Technical Report STO-TR-MSG-136-Part-VI, 2018.
44. IEEE Standards Association. *1730-2010 – IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP)*, 2010.
45. IEEE Standards Association. *1730.1-2013 – IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process Multi-Architecture Overlay (DMAO)*, 2013.
46. van Steen M and Tanenbaum AS. *Distributed Systems*. 3rd ed. distributed-systems.net, 2017.
47. van den Berg TW and Luz R. Simulation environment architecture development using the DoDAF. In *Proc. 2015 Fall Simulation Innovation Workshop*. 15F-SIW-019.
48. Michlmayr A, Rosenberg F, Platzer C et al. Towards recovering the broken SOA triangle—A software engineering perspective. In *Proc. 2nd Int'l Workshop Service Oriented Software Engineering (IW-SOSWE'07)*. ACM, pp. 22–28.
49. Michlmayr A, Rosenberg F, Leitner P et al. End-to-end support for QoS-aware service selection, binding, and mediation in VRESCo. *IEEE Transactions on Services Computing* 2010; 3(3): 193–205.
50. Fette I and Melnikov A. *The WebSocket Protocol—Internet Engineering Task Force (IETF) Request for Comments: 6455*, 2011.
51. Organization for the Advancement of Structured Information Standards. *Advanced Message Queuing Protocol (AMQP) Version 1.0*, 2012.
52. Ecma International – European association for standardizing information and communication systems. *ECMA-404 – The JSON Data Interchange Format*, 2013.
53. Pullen JM, Corner D, Brook A et al. MSDL and C-BML working together for NATO MSG-085. In *Proc. 2012 Spring Simulation Interoperability Workshop (SIW)*. 12S-SIW-045, Simulation Interoperability Standards Organization.
54. Allen GW and Schroeder L. Utilization of Service Oriented Architecture (SOA)-based commercial standards to address Live, Virtual, Constructive (LVC) interoperability challenges. In *Proc. Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2011*. National Training and Simulation Association.
55. Coolahan JE and Allen GW. LVC Architecture Roadmap Implementation—results of the first two years. In *Proc. 2011 Fall Simulation Interoperability Workshop (SIW)*. 11F-SIW-025, Simulation Interoperability Standards Organization.
56. Allen GW, Lutz R and Richbourg R. Live, virtual, constructive, architecture roadmap implementation and net-centric environment implications. *ITEA Journal* 2010; 31(3): 355–364.
57. Asprusten M and Hannay JE. Simulation-supported wargaming using M&S as a Service (MSaaS). In *Proc. NATO Modelling and Simulation Group Symp. on Multi-National Pooling and Sharing of Simulation Resources under the M&S as a Service Paradigm (STO-MP-MSG-159)*.
58. Çayırcı E and Özçakır L. Modeling and simulation support to the defense planning process. *J Defense Modeling and Simulation: Applications, Methodology, Technology* 2016; .
59. Simulation Interoperability Standards Organization. *SISO-STD-007-2008 – Standard for Military Scenario Definition Language (MSDL)*, 2008.
60. Simulation Interoperability Standards Organization. *SISO-STD-011-2014 – Standard for Coalition Battle Management Language (C-BML) Phase 1, Version 1.0*, 2014.
61. Ford K, Lloyd J and Smith N. NATO Aligned UK Approach to Modelling and Simulation as a Service. In *Proc. NATO Modelling and Simulation Group Symp. on M&S Technologies and Standards for Enabling Alliance Interoperability and Pervasive M&S Applications (STO-MP-MSG-149)*.
62. Çayırcı E. Modeling and simulation as a cloud service: A survey. In *Proc. 2013 IEEE Winter Simulation Conf.* IEEE, pp. 389–400.
63. Çayırcı E and Rong C. Intercloud for simulation federations. In *Proc. 2nd IEEE Int'l Workshop Cloud Computing Interoperability and Services (Intercloud 2011)*. IEEE.
64. Çayırcı E, Karapinar H and Özçakır L. Joint military space operations simulation as a service. In *2017 Winter Simulation Conference (WSC)*. pp. 4129–4140.
65. Lloyd J, Ford K and Skinner S. Common non run-time simulation services – lessons from UK MOD research. In *Proc. Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2016*. National Training and Simulation Association.
66. van den Berg TW, Siegel B and Cramp A. Containerization of High Level Architecture-based simulations: A case study. *J Defense Modeling and Simulation: Applications, Methodology, Technology* 2017; 14(2): 115–138.
67. van den Berg TW, Siegel B and Cramp A. Guidelines and best practices for using Docker in support of HLA federations. In *Proc. 2016 Fall Simulation Innovation Workshop*. 16F-SIW-031.
68. Cramp A, van den Berg T and Huiskamp W. Service orientation for the design of HLA federations. In *Proc. 2014 Fall Simulation Innovation Workshop (SIW)*. 14F-SIW-048,

Simulation Interoperability Standards Organization.
69. Kallfass D, Bertschik M, Vrieler S et al. Proof of concept demonstrator of MSG-136 for using and providing simulation as a service within NATO environments. In *Proc. NATO Modelling and Simulation Group Symp. on M&S Technologies and Standards for Enabling Alliance Interoperability and Pervasive M&S Applications (STO-MP-MSG-149)*.
70. Gallant S, Metevier CJ and Gaughan C. Systems engineering an executable architecture for M&S. *M&S Journal* 2014; Spring: 16–24.
71. NATO Communications and Information Agency. *The C3 Taxonomy*, 2016. Accessed June 2018.
72. Mahmood I. *A Verification Framework for Component Based Modeling and Simulation, putting the pieces together*. PhD Thesis, KTH Royal Institute of Technology, 2013.
73. Çayırcı E, Rong C, Huiskamp W et al. Snow Leopard cloud: A multi-national education training and experimentation cloud and its security challenges. In *Proc. Cloud Computing 2009*. Springer, pp. 57–68.
74. NATO Standardization Office. AMSP-02 Allied Framework for Modelling and Simulation as a Service (MSaaS) Governance Policies, edition (a), version 1.0. Technical Report STANREC 4794, 2018.
75. NATO Science and Technology Organization. Operational Concept Document (OCD) for the Allied Framework for M&S as a Service. Technical Report STO-TR-MSG-136-OCD, NATO MSG, 2018.
76. NATO Research and Technology Organisation. NATO Education and Training Network. Technical Report RTO-TR-MSG-068, 2012.
77. van den Berg TW. Interoperability testing tools and techniques. In *Proc. 2018 Fall Simulation Innovation Workshop*. 18F-SIW-007.
78. van den Berg TW, Jansen REJ and Ufer H. Design patterns for and automation of federation state control. In *Proc. 2009 Spring Simulation Interoperability Workshop*. 09S-SIW-009.
79. Gaughan C, Metevier CJ, Athmer K et al. Bringing next generation simulation into the land of practicality. In *Proc. 2013 Fall Simulation Interoperability Workshop*. 13F-SIW-017.
80. Snively K, Leslie R and Gaughan C. Runtime execution management of distributed simulations. In *Proc. 2013 Fall Simulation Interoperability Workshop*. 13F-SIW-019.
81. Hannay JE, Brathen K and Mevassvik OM. A hybrid architecture framework for simulations in a service-oriented environment. *Systems Engineering* 2017; 20(3): 235–256.