Service Orientation for the Design of HLA Federations

Anthony Cramp, Tom van den Berg, Wim Huiskamp
TNO
Oude Waalsdorperweg 63
2597 AK The Hague
The Netherlands
{anthony.cramp, tom.vandenberg, wim.huiskamp}@tno.nl

ABSTRACT: *Service oriented modeling and simulation (M&S) is being pursued by many nations and organizations. Approaches being taken span from the provision of M&S tools and applications via -as-a-Service cloud computing technologies to the actual construction of M&S via service oriented techniques. Often these approaches propose that new technologies are required to achieve service orientation. However, the benefits of service orientation derive from its core concepts of services, high interoperability, and loose coupling. Achieving service orientation in a High Level Architecture (HLA) context requires rethinking how federations are designed including the decomposition of simulation functionality into federates. This paper presents a federation designed in a traditional manner, illustrates how service oriented design alters the federation design, discusses the benefits and drawbacks of the two approaches, and provides possible improvements to the Distributed Simulation Engineering and Execution Process (DSEEP) that may become part of the next iteration of the standard.*

1 Introduction

1.1 Service orientation

Service orientation is an approach to the design of heterogeneous, distributed systems in which solution logic is structured in the form of interoperating services. Service orientation has goals of creating intrinsic interoperability across a distributed system and, thereby, improving the ability to federate across a multitude of individual service implementations.

Erl [1] proposes that service orientation is a design paradigm and constitutes one element of service oriented computing. Erl's elements of service oriented computing along with their inter-relationships are illustrated in Figure 1.

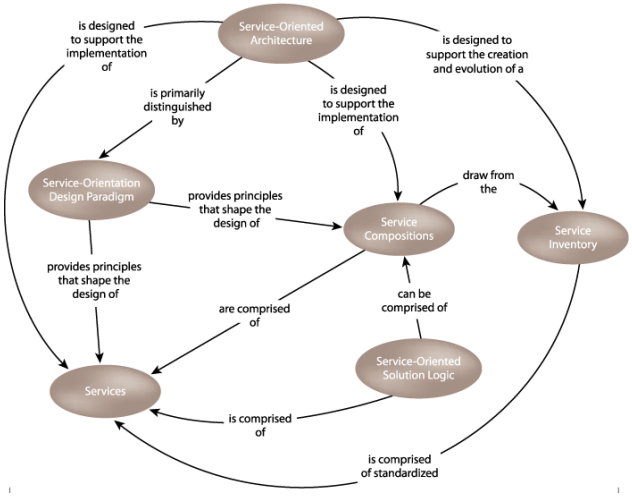

Figure 1: Service oriented computing.

As is seen in the diagram, all arrows point to Services as being the core element of service oriented computing. The benefits of service orientation derive from the structuring of solution logic (aka simulation) in the form of services.

A Service Oriented Architecture (SOA) captures the necessary technical information required for services to achieve meaningful interoperability. A SOA will exist per distributed system; there is not one single SOA applicable to all distributed systems. The Open Group has defined a SOA Reference Architecture (SOA RA) that can be used for creating a SOA [2]. Quoting the Open Group: "The SOA RA provides guidelines and options for making architectural, design, and implementation decisions in the implementation of solutions. The goal of this SOA RA is to provide a blueprint for creating or evaluating architecture."

The question arises: how can distributed simulation, being a type of distributed system, benefit from service orientation? A first step is to analyze how distributed simulations can be designed with service orientation in mind, since we believe this is a pre-requisite for realizing a SOA solution. To this end it is beneficial to look at the Distributed Simulation Engineering and Execution Process (DSEEP) [3] - a recommended practice for the engineering and execution of a distributed simulation environment - to see how service orientation fits. To date, DSEEP does not include any specific activities or tasks for the design and construction of a simulation environment based on service orientation.

Before we continue, let's first have a brief summary of DSEEP, provided in the next section.

1.2 Distributed Simulation Engineering and Execution Process

DSEEP is a seven step process model for the engineering and execution of a distributed simulation environment. Each step is broken down into activities and tasks. The idea is that the process model is tailored to the needs of a project, integrating other systems engineering activities and tasks native to the organization.

The seven steps are in summary:

- Step 1: Define simulation environment objectives. Define and document a set of user/sponsor needs that are to be addressed and transform these needs into a more detailed list of specific objectives for that environment.
- Step 2: Perform conceptual analysis. Develop an appropriate representation of the real-world domain that applies to the defined problem space and develop the appropriate scenario. Transform the objectives for the simulation environment to simulation environment requirements.
- Step 3: Design simulation environment. Produce the design of the simulation environment. This involves identifying applications that will assume some defined role in the simulation environment (member applications) that are suitable for reuse, creating new member applications if required, allocating the required functionality to the member application representatives.
- Step 4: Develop simulation environment. Define the information that will be exchanged at runtime during the execution of the simulation environment, establish interface agreements, modify member applications if necessary, and prepare the simulation environment for integration and test.
- Step 5: Integrate and test simulation environment. Integration activities are performed, and (formal) testing is conducted to verify that interoperability requirements are being met.
- Step 6: Execute simulation. The simulation is executed and the output data from the execution is captured and pre-processed.
- Step 7: Analyze data and evaluate results. The output data from the execution is analyzed and evaluated, and results are reported back to the user/sponsor.

1.3 Document overview

This paper discusses a modified DSEEP that includes service orientation activities in steps 3 and 4. The paper shows how a particular simulation environment is designed using the "classic" DSEEP and then shows how the same simulation environment would be designed with a "service oriented" DSEEP. Section 2 briefly describes the Case Study. Section 3 presents the classic DSEEP and section 4 presents the service oriented DSEEP using IBM's Service Oriented Modeling Architecture (SOMA) as a guide. Section 5 briefly discusses how the High Level Architecture (HLA) may serve as a SOA for services. Lastly, section 6 provides a summary and conclusions.

2 Case study

The case study is called "Situational Awareness in Maritime Missile Defense", where units in a task force exchange information to create a common tactical air picture for real world objects in a real world environment. Real world objects are for example missiles and aircraft. Each unit has sensors to track real world objects and a tactical data link to exchange and manage tracks of real world objects with the other units in the task force.

The objective of the simulation environment is to measure and evaluate the different options in creating such a common tactical air picture. The simulation environment models the real world environment and real world objects, the units and their sensors, the relevant command and control processes, and the tactical data link. The simulation is a non-real time Monte Carlo simulation, with stochastic variations in sensor parameters. The scope of the case study is the design of the simulation environment, i.e. DSEEP steps 3 and 4.

The case study makes use of SoaML (Service oriented architecture Modeling Language) and SysML (Systems Modeling Language) for the design of the simulation environment. SoaML [5] and SysML [6] are two UML (Unified Modeling Language) profiles from the Object Management Group. SoaML is a modeling language for the specification and design of services within a service-oriented architecture. SysML is a general-purpose modeling language for systems engineering applications.

3 Designing a simulation environment – the classic way

For the design of a simulation environment, DSEEP steps 3 and 4 are applicable. DSEEP design activities are:

Step 3: Design simulation environment

- 3.1: Select member applications
- 3.2: Design simulation environment
- 3.3: Design member applications

Step 4: Develop simulation environment

- 4.1: Develop simulation environment data exchange model
- 4.2: Develop simulation environment agreements

These activities are elaborated in the following sections.

3.1 (3.1) Select member applications

The purpose of this activity is to determine the suitability of individual simulation systems to become member applications of the simulation environment. This includes amongst others a search of existing repositories, an analysis of the capabilities of potential member applications, and a trade-off analysis.

An example of a simple trade-off table is shown in Figure 2. The rows list the requirements (that is, the operational activities that the simulation environment must model) and the columns list the potential member applications. Each cell lists the suitability (in the range 1 to 3) of the potential member application for fulfilling the listed requirement. A more complex trade-off table may indicate how various design options perform against relevant criteria.

	SV-1 Candidate::SIM F	SV-1 Candidate::SIM E	SV-1 Candidate::SIM D	SV-1 Candidate::SIM C	SV-1 Candidate::SIM B	SV-1 Candidate::SIM A
1 OV-5::Deploy Weapon	2					
1.1 OV-5::Launch	2					
1.2 OV-5::Fly	2					
1.3 OV-5::Detonate	1					
2 OV-5::Situation Awareness		2	3	3		3
2.1 OV-5::Assess RWO		2	2	2		2
2.1.1 OV-5::Classify RWO		2	2	2		2
2.1.2 OV-5::Identify RWO		2	2	2		2
2.1.3 OV-5::Resolve Identification and Classification Conflicts			2	2		2
2.2 OV-5::Compile CTP		2	3	3		3

Figure 2: A simple trade-off table.

3.2 (3.2) Design simulation environment

The purpose of this activity is to prepare the simulation environment design and allocate the responsibility to represent the entities and actions in the conceptual model to the member applications.

Once the member applications are selected, the physical architecture of the simulation environment can be developed. This includes amongst others the allocation of modeling responsibilities to member applications and the evaluation of design options.

The structure of the simulation environment can be represented with a SysML Block Definition Diagram shown in Figure 3.

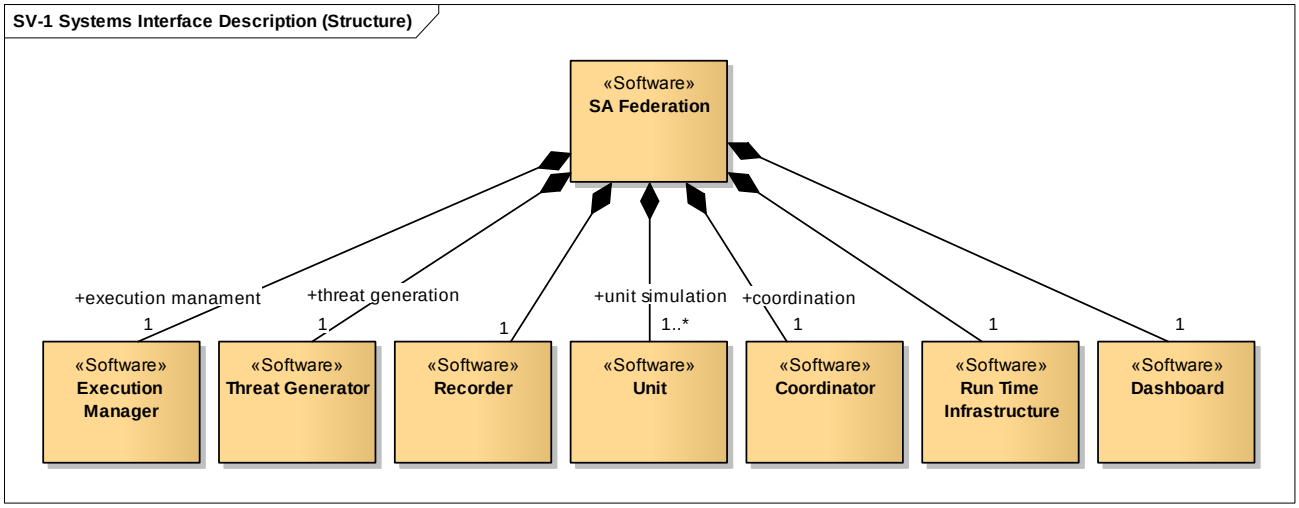

Figure 3: Physical structure of the simulation environment.

Figure 3 shows the block named SA Federation, which is composed of a block named Execution Manager, a block named Threat Generator, etc. The composition relationship shows that the SA Federation is composed of one Execution Manager that fulfills the role execution management, one Threat Generator that fulfills the role of threat generation, and one or more Units that fulfill the role unit simulation.

The allocation of modeling responsibilities to member applications can be visualized using the SysML allocation relationship as shown in Figure 4. Here the allocation relationships are stereotyped with IsCapableOfPerforming.

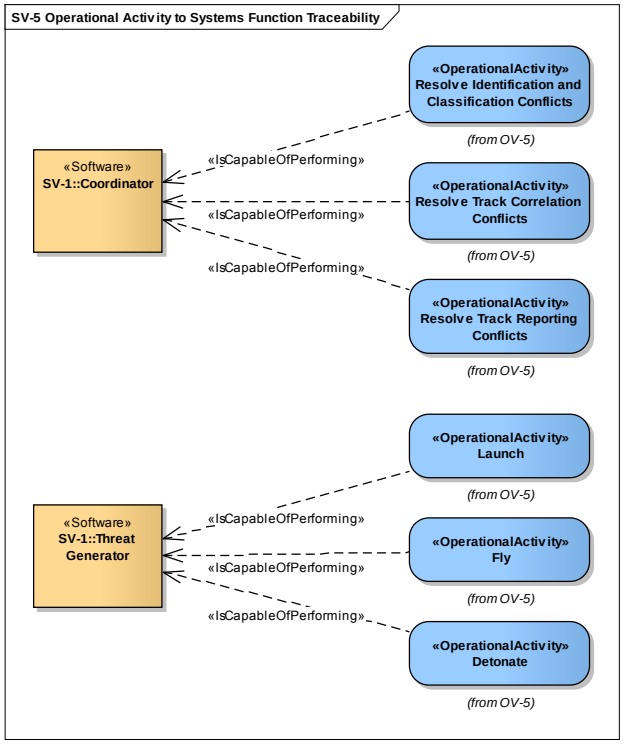

Figure 4: Allocation of modeling responsibilities.

3.3 (3.3) Design member applications

The purpose of this activity is to design member applications.

This activity is not further elaborated.

3.4 (4.1) Develop simulation environment data exchange model

The purpose of this activity is to develop the simulation data exchange model.

The allocation of responsibility of modeling operational activities to member applications will lead to requirements on information exchange between member applications. The information exchange requirements can be specified in a logical data model, a logical representation of the Simulation Data Exchange Model (SDEM). The entity items represent the data exchanged by member applications. The entity items may be grouped in modules, and augmented with attributes and data types. An example of a logical data model where the data is grouped in modules is shown in Figure 5.

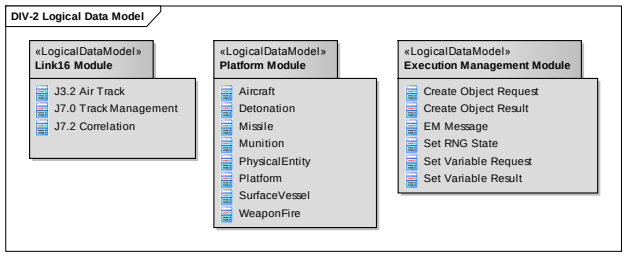

Figure 5: Logical data model of SDEM.

Based on the entity items specified in the logical data model, a physical representation of the SDEM can be developed; for HLA the physical representation corresponds to the Federation Object Model (FOM).

3.5 (4.2) Establish simulation environment agreements

The purpose of this activity is to establish the operating agreements among member applications to establish a fully consistent, interoperable, simulation environment.

Various diagrams can be used to capture simulation environment agreements and describe the way in which simulation architecture services (i.e. HLA RTI API services) are used by member applications. Simulation environment agreements cover issues such as initialization, synchronization, termination, progression of time, events, life cycle of entities, update rates, etc. Diagrams that can be used to visualize agreements between member applications include UML state transition diagrams (e.g. for simulation execution states), UML sequence diagrams (e.g. for specific event traces), and UML activity diagrams (e.g. for describing the actions that must be performed in a certain state). These diagrams are generally system oriented and describe agreements from a system point of view.

4 Designing a simulation environment – a service oriented approach

The previous section described the approach for designing a simulation environment using a more systems oriented approach based on DSEEP. This section introduces the notion of service and identifies activities and tasks for designing a simulation environment following a services oriented approach that is currently lacking in DSEEP. This approach is based on the main steps in SOMA (Service-Oriented Modeling and Architecture), a methodology from IBM for the identification, specification and realization of services in a service oriented architecture [4]. Using a service oriented approach based on SOMA, the activities for DSEEP step 3 and 4 are now as follows:

Step 3: Design simulation environment

- 3.1: Identify services
- 3.2: Specify services
- 3.3: Realize services

Step 4: Develop simulation environment

- 4.1: Compose services
- 4.2: Implement services

Throughout these activities the service orientation design principles from [1] should be followed. The activities are described in the following sections.

4.1 (3.1) Identify services

The purpose of this activity is to identify candidate services that are involved in the simulation environment. SOMA describes three techniques for identifying services: goal-service modeling (using objectives and performance measures), domain decomposition (a top down-down approach), and existing asset analysis (a bottom up approach). These three techniques can equally be applied to the identification of services in a simulation environment. While identifying candidate services also modeling responsibilities should be allocated to these services. In case of existing services, modeling responsibilities should also be derived from these services (reverse-engineered).

This activity includes the following tasks:

- Perform top-down analysis to identify services, by using the conceptual model and simulation environment requirements.
- Perform bottom-up analysis to identify services, by searching repositories and surveying existing components, interfaces, services, etc.
- Perform goal-service modeling to identify services, by using the objectives and performance measures of the simulation environment.
- Categorize services in a service hierarchy or grouping.
- Allocate modeling responsibilities to services.

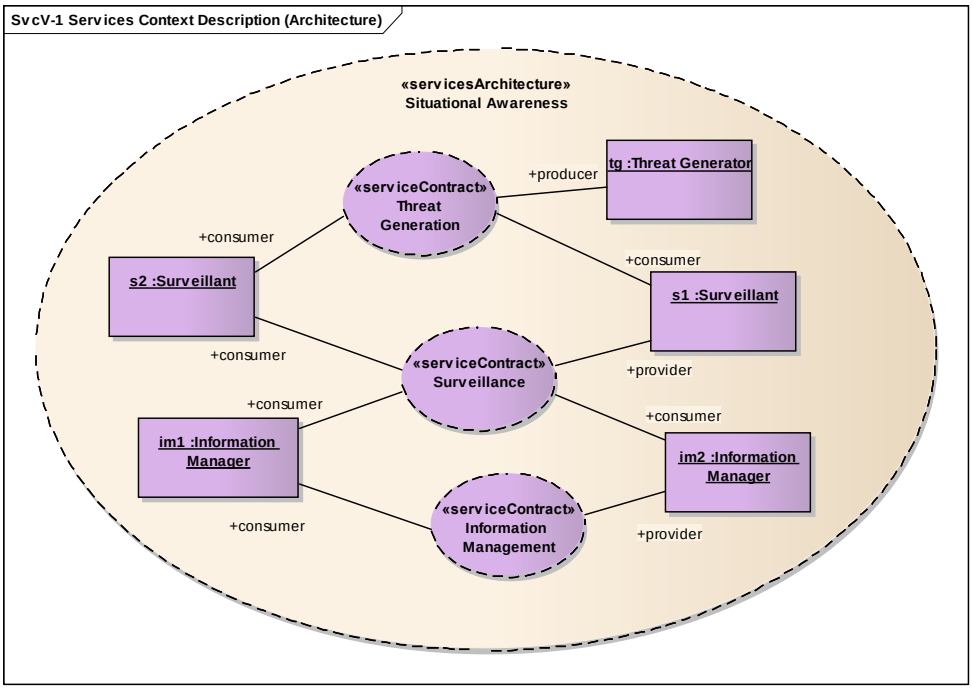

Figure 6: Services architecture.

The SoaML services architecture diagram can be used to describe the identified services. The services architecture is a high-level view of the Service Oriented Architecture of the simulation environment. It shows how participants work together for some purpose by providing and using services. Figure 6 provides the Service Oriented Architecture for Situational Awareness. It shows the service participants, the role of each participant, and service contracts. A service contract is the specification of the agreement between providers and consumers of the service. It specifies for example the information exchanged between service participants and the obligations that must be met in doing so. The details of service contracts and service participants will be provided by later activities (service specification and service realization). At this stage a high level overview of candidate services is sufficient.

The identified candidate services are:

- Threat Generation: a service for the generation of real world objects.
- Surveillance: a service for the tracking of real world objects.
- Information Management: a service for the general management of tracks.

The Surveillance and Information Management services are defined in line with the Link 16 message grouping.

4.2 (3.2) Specify services

The purpose of this activity is to elaborate and detail the identified services, and to specify the service interfaces.

This activity includes the following tasks:

- Specify service interfaces.
- Specify service dependencies on other services.
- Specify the flow of information among services.
- Develop service data exchange model.
- Develop service agreements.

The service interface specifies how the service provider and service consumer interact, and specifies the conditions under which they interact. Referring to SoaML [5], the service interface specifies:

- Name of the service interface.
- Required and provided interfaces.
- Service interface interactions with data input/output, pre-conditions, post-conditions, and exceptions.
- Any rules or constraints.
- Qualities of the service interface, such as performance and fidelity.
- Policies for using the service interface.

The required and provided interfaces will result in what is called a service data exchange model. This data exchange model specifies the messages that the service produces and consumes. For HLA this is comparable with the Simulation Object Model (SOM).

The service interface interactions, rules, constraints, qualities, policies, etc., are collectively called the service agreements. They specify the conditions or terms under which the service may be used. For HLA this is comparable with the agreements that are associated with the SOM.

The services architecture diagram for Situational Awareness shown in the previous activity lists a number of service contracts and service participants. The connector between a service contract and a participant shows the role of the participant in the service contract, e.g. consumer or producer. This role relates to a service interface.

When looking at these roles in the services architecture diagram the following service interfaces can be identified: Threat Generation Service, Surveillance Service, and Information Management Service. Figure 7 lists the service interfaces. The diagram also shows two additional service interfaces: Recording Service and Execution Management Service. These are not shown in the services architecture diagram for space reasons.

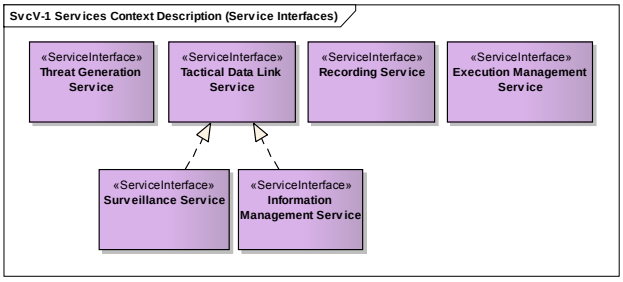

Figure 7: Service interfaces.

A service interface can be bi-directional and is specified from the perspective of the service provider. The service interface specifies the realized interfaces (by the provider) and the used interfaces (of the consumer), and the messages that will be received by the provider or consumer respectively. The service interface can also have an associated behavior that specifies the interactions between provider and consumer.

Figure 8 shows the interfaces associated with the service interfaces Information Management Service, Surveillance Service, and Threat Generation Service. In this example, the service interface Information Management Service only realizes the Information Management interface on which the service receives Track Management and Track Correlation messages. This service interface does not use an interface to send messages back to the consumer. Similarly, the service interface Surveillance Service only realizes the Surveillance interface on which the service receives Air Track messages. And the service interface Threat Generation Service uses the Threat Generation interface of the consumer to send WeaponFire, Munition and Detonation messages on. Other service interfaces may both realize and use interfaces to receive and send messages on.

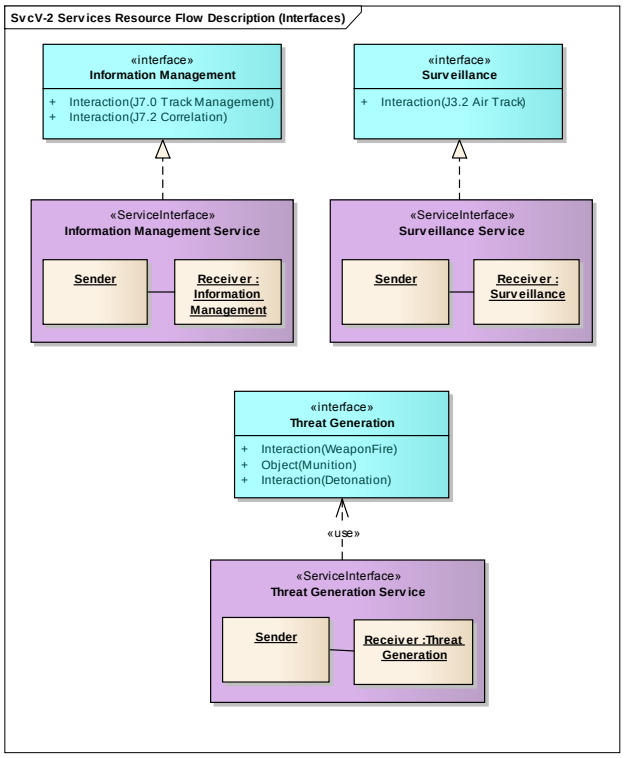

Figure 8: Service interfaces details.

Following this example, the service data exchange model for the Surveillance Service consists of an Air Track message that the service consumes and produces.

The behavior associated with a service interface may be described by a UML sequence diagram, a UML state diagram, or a UML activity diagram. For example, the sequence diagram in Figure 9 shows that Air Track messages are sent every 8 to 20 seconds from a sender to a receiver interface. This is of course a simple example, but the same diagram(s) may be used to illustrate much more complex interactions between service consumer and producer. In SoaML the behavior diagram can be associated with the service interface using the containment relationship.

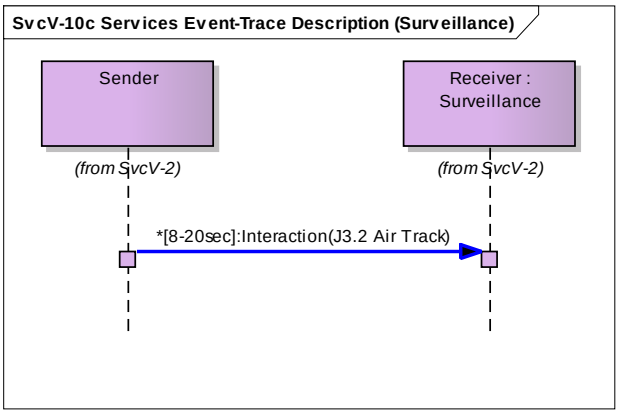

Figure 9: Services event trace.

Rules, constraints, etc., may apply to the service interface. These are usually described in text, although UML constructs can be used to associate these with the service interface that is specified in SoaML.

4.3 (3.3) Realize services

The purpose of this activity is to evaluate service realization options and decide on which member application will realize what service participant.

At this point in the process the actual services do not exist yet and still need to be implemented (unless services are provided by existing member applications). The idea is to evaluate service realization options and select the most feasible option for implementation.

This activity includes the following tasks:

- Analyze candidate member applications for service realization.
- Evaluate service realization options.
- Determine technical feasibility.
- Record realization decisions.
- Document simulation environment design.
- Design member applications.

A service participant is the type of a provider and/or consumer of a service. The services architecture shown earlier lists a number of participants, which are ultimately realized by a member application in the simulation environment. Each participant plays a role in a service contract and uses the service interfaces to interact with other service participants. A participant uses ports to provide or request services.

Figure 10 shows three service participants and their ports. Each port has a name and is stereotyped with either "Service" (offering a service) or "Request" (consuming a service). For example, the Surveillant has two ports related to Surveillance Service and one port related to Threat Generation Service. One Surveillance Service port is used for receiving Air Track messages (the port named SS), and the other for sending Air Track messages (the port named SR). The third port is named TGR, and is used to receive threat generation messages on.

In the diagram the Request port is a "conjugate" port, that is, it inverts the required and provided interfaces of a service interface. The tilde ("~") is used to indicate that a port is a conjugate port.

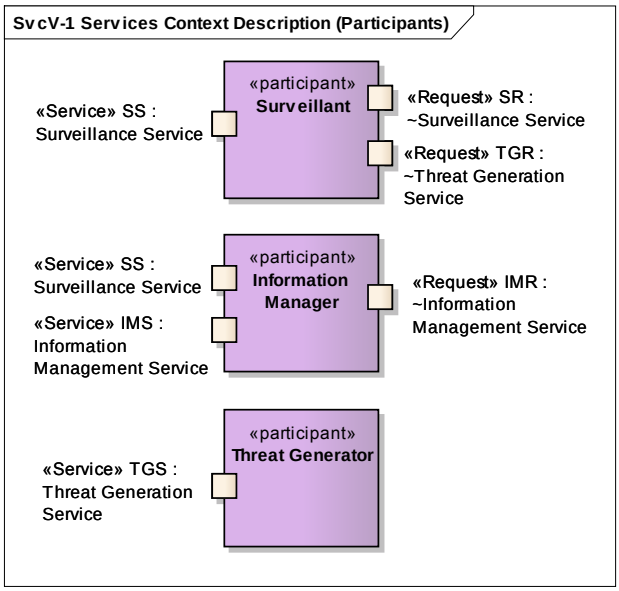

Figure 10: Service participants.

The above diagram can be expanded to show more detail on the interfaces that each port supports. Figure 11 shows the provided and required interface per port. Depending on the service interface, a port can have any number of provided or required interfaces.

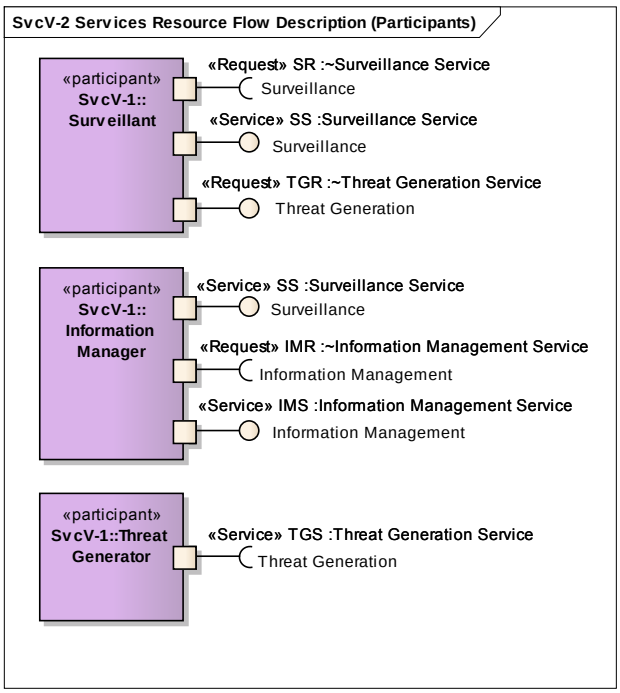

Figure 11: Service participants: ports and interfaces.

The first task is to analyze the suitability of candidate member applications to represent service participants and realize the specified services. In some cases member applications need to be designed from scratch, and/or candidate member applications need to be modified. .

The next task is to evaluate the service realization options. Decision factors for service realization include service security, service cohesion, service coupling, service usage and service deployment. Depending on the situation there may be various options in the realization of services by member applications. For example, the execution management service can be provided by a dedicated member application, or can be provided by a member application that also provides other services such as threat generation. These options need to be evaluated. Also the technical feasibility of the different options should be determined and the decisions regarding the service realizations should be recorded.

Once all member applications are identified and service realization decisions are taken, the next task is to document the simulation environment design. Some of the diagrams that may be used to document the design are described next.

The structure of the simulation environment can be described with a SysML Block Definition Diagram as already discussed in section 3.2. Figure 12 shows the service realization by member applications using the UML Realizes relationship between participant and

selected member application. By using this relationship each member application inherits the ports that are defined for the participant. So, in this example, the Coordinator is both an EM Participant and an Information Manager, and inherits the ports from Information Manager and from EM Participant.

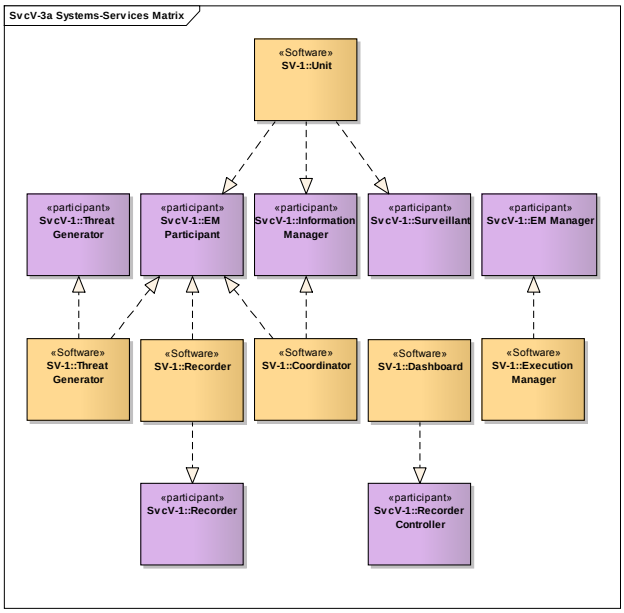

Figure 12: Systems-services realization.

The inherited service ports can subsequently be used in the description of the information flow between member applications. A SysML Internal Block Diagram can be used for this. Figure 13 provides an example of a simulation environment with three member applications and the information flow between them: one Coordinator, one Unit and one Threat Generator. For simplicity, the Recorder and Execution Manager are left out of the picture.

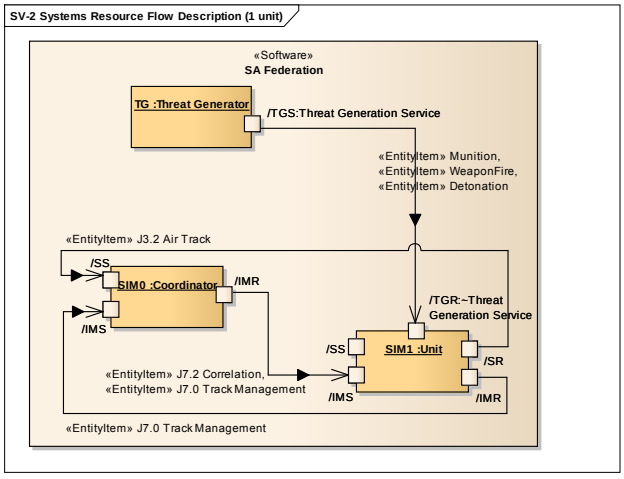

Figure 13: Systems resource flow.

4.4 (4.1) Compose services

The purpose of this activity is to assemble a simulation data exchange model and agreements from the individual service data exchange models and service agreements.

This activity includes the following tasks:

- Develop simulation data exchange model from service data exchange models.
- Develop simulation environment agreements from service agreements.

This activity is not further elaborated.

4.5 (4.2) Implement services

The purpose of this activity is to implement the services in the member applications.

This activity includes the following tasks:

- Implement member application designs.
- Implement simulation environment infrastructure.

This activity is not further elaborated.

5 Service oriented HLA Federations: discussion

A distributed simulation architecture designed with service orientation in mind needs to be realized in some manner. This realization is captured in a Service Oriented Architecture (SOA) that, according to Erl [1], is developed with eight service oriented design principles in mind:

- Standardized Service Contract
- Service Loose Coupling
- Service Abstraction
- Service Reusability
- Service Autonomy
- Service Statelessness
- Service Discoverability
- Service Composability

How these design principles are captured in a distributed simulation framework such as the High Level Architecture (HLA) is the next major work activity on the path to true service oriented distributed simulations.

Mapping these design principles to the HLA requires thought on how services are implemented as federates and

what service support is provided by the HLA Runtime Infrastructure (RTI). Federates could provide one or more services depending on granularity and cohesiveness of the defined services. On the other hand, higher level composed services might implemented across multiple services and require service orchestration functionality within the RTI. The following paragraphs describe how some of the service design principles listed above might be achieved in the HLA context.

What would a standardized service contract mean in the context of the HLA? The Simulation Object Model (SOM) is more a data dictionary than it is a service interface description. Base Object Models (BOMs) provide a possible path towards the definition of a service contract [7][8].

Does the Service Discoverability design principle make sense for the HLA, considering federations are usually engineered and executed in a manner that ensures all data providers and consumers are present and available by design? One possible use for service discoverability within HLA could be to provide a runtime check within a federate to ensure all required services are available. How would the HLA support discoverability? One approach could leverage the use of BOMs as service contracts and registered with a federation execution as FOM modules. In the current HLA standard it would be possible to discover which service contracts (BOMs) a particular federate supports by having service provider federates register their BOMs as FOM modules and service consumer federates inspecting the Management Object Model (MOM) for the FOM modules supported by particular federates. Another approach would be to define a standard FOM module containing generic "Service Information Objects" that service providers would register and to be discovered by service consumers. Each of these solutions have drawbacks and a nicer solution would be to introduce the concept of a service to the HLA and provide mechanisms within the HLA to explicitly allow a federate to register services they provide and for a consumer to discover services as they appear.

Service Composability provides for the creation of higher level services formed from the composition of lower level services. These composed services require orchestration in how the lower level services are accessed. This orchestration can be done automatically from a high level description of the composition. Adding support to the HLA RTI for the execution of such a high level description could allow for the automatic provision of composed services to service consumers. This would require the HLA standard to define an appropriate composition language (probably by adopting an existing standard) and adding the requirement that the HLA RTI provide an execution engine for the language.

Mapping service orientation to the HLA requires more work. Note that the goal here is not to connect to the RTI in a service oriented manner; rather the goal is for federates to interact in a service oriented manner. The interaction would be facilitated via the RTI providing the role of the Enterprise Service Bus (ESB) although it is expected that additional services or a layer above the existing services be added in order to fully support service oriented HLA federations. These new services or service-oriented layer would not (necessarily) be reliant on web service technologies. The existing language APIs are fully capable of supporting service oriented message patterns.

6 Summary and conclusions

The DSEEP is currently lacking a service oriented development approach. This paper has identified additional activities for inclusion in the DSEEP to support service oriented development of a distributed simulation. Further evaluation needs to be conducted on how these activities can be best be integrated into DSEEP.

The description of a simulation environment in terms of services provides potential for enhanced reuse of simulation functionality. Services are typically small and self-contained units of simulation logic and, as such, provide ideal building blocks for composing simulations. This is definitely a benefit. A drawback (or better said, a risk) might be that services result in many member applications all providing only one or a couple of services. This risk of added complexity and overhead should however be addressed in activity (3.3) Realize services, where the different service realization options are evaluated.

A service oriented approach introduces new notions such as service interfaces, service data exchange model and service agreements. How these concepts are captured in particular distributed simulation technical architectures is an open question. One option could be Base Object Models (BOMs). BOMs began life in the HLA community but are being expanded to other distributed simulation environment, such as Test and Training Enabling Architecture (TENA), and have Service-Oriented Architectures (SOAs) as one of their potential application domains. More work needs to directed in this area.

Mapping service orientation to distributed simulation technical architectures will be a challenging task. Note that the goal is to have distributed simulation participants interact in a service-oriented manner rather than have the participants interact with the simulation infrastructure as if it were a service. In terms of the HLA, the goal is for federates to see each other as service providers and to

interact as such instead of purely interacting with the RTI. Although, the RTI is an essential component of an HLA based SOA as it provides the core activities of an Enterprise Service Bus (ESB).

The authors recommend that SISO, as the simulation interoperability standards organisation and in its role as custodian of HLA and DSEEP, should lead the investigation w.r.t. the application of SOA in the M&S domain. A study group should research the topic in more detail and develop recommendations regarding service oriented methodology and possible extensions of a future iteration of the DSEEP and HLA standards.

7 References

[1] Thomas Erl, *"SOA: Principles of Service Design"*, Prentice Hall, 2007.

[2] The Open Group, *"SOA Reference Architecture Technical Standard"*, Document Number C119, November, 2011.

[3] *"IEEE Standard 1730-2010 – IEEE Recommended Practice For Distributed Simulation Engineering and Execution Process (DSEEP)"*, Institute of Electrical and Electronics Engineers, 2010.

[4] Norbert Bieberstein et al, *"Executing SOA: A Practical Guide for the Service-Oriented Architect"*, IBM Press, 2008.

[5] *"Service Oriented Architecture Modeling Language (SoaML), Version 1.0.1"*, Object Management Group, May 2012.

[6] *"Systems Modeling Language (SysML), Version 1.3"*, Object Management Group, June 2012.

[7] *"SISO-STD-003-2006: Base Object Model (BOM) Template Specification"*, Simulation Interoperability Standards Organization, 2006.

[8] *"SISO-STD-003.1-2006: Guide for Base Object Model (BOM) Use and Implementation"*, Simulation Interoperability Standards Organization, 2006.

8 Author biographies

ANTHONY CRAMP is a Defence Science and Technology Organisation (DSTO, Australia) International Fellow working within the Modeling, Simulation and Gaming department of TNO Defence, Security and Safety, The Netherlands until February 2015 on topics including M&S as a Service and HLA performance. He joined DSTO in 1999 working on the Virtual Maritime System Architecture, a HLA based simulation framework for maritime C2 centric experimentation, and received his PhD from the University of Adelaide in 2009 based on research into the construction of HLA simulations containing multiple systems of systems. His research interests include distributed systems and simulations, software architectures, and programming languages. From March 2015 he can be contacted via his DSTO email address: anthony.cramp@dsto.defence.gov.au.

TOM VAN DEN BERG is a scientist in the Modeling, Simulation and Gaming department at TNO Defence, Security and Safety, The Netherlands. He holds an M.Sc. degree in Mathematics and Computing Science from Delft Technical University. His research area includes simulation systems engineering, distributed simulation architectures and concept development & experimentation.

WIM HUISKAMP is Chief Scientist Modelling, Simulation and Gaming in the M&S department at TNO Defence, Security and Safety in the Netherlands. Wim leads TNO's research program on Simulation, which is carried out on behalf of the Dutch MOD. Wim is a member of the NATO Modelling and Simulation Group (NMSG) and currently acts as its Chairman. He has also chaired several NMSG Technical Working groups, including the NMSG M&S Standards Subgroup (MS3) and he is the liaison of the NMSG to the Simulation Interoperability Standards Organization (SISO).