An Ontology System for Interoperation between DDS and HLA

Min-Seok Oh, Yun-Hee Son and Kyu-Chul Lee

Department of Computer Engineering, Chungnam National University, 220 Gung-Dong, Yuseong-Gu, Daejeon, Korea; oms8907@cnu.ac.kr, mellow211@cnu.ac.kr, kclee@cnu.ac.kr

Abstract
Simulation systems have been used in many fields of society. Training effects, budget reductions, and situation prediction can be achieved by experiencing activities in the real world through simulation. Because of these advantages, simulation systems have been used in national defense fields in particular. National defense simulations can be divided into three parts, live simulations, virtual simulations, and constructive simulations, and each simulation has respective features and advantages. L-V-C integrated simulation in the national defense field is becoming increasingly important because it allows a variety of operations/training scenarios to be conducted by joint training. It also improves the accuracy of training and can reduce both time and budget. However, in the case where L-V-C have heterogeneous middleware, interoperability of each middleware is necessary for integration of L-V-C. In this research, we propose an ontology-based interoperation between DDS (a live middleware) and HLA (a virtual middleware). For this, we constructed an ontology that has relation information of APIs and parameters between HLA and DDS, and stored the ontology in a triple store. Furthermore, we developed an ontology processing module for management of the ontology. The proposed ontology-based system allows interoperation between HLA and DDS.

Keywords: DDS, HLA, Interoperation, L-V-C Middleware, Ontology, Simulation

1. Introduction
Simulation systems are used in the national defense field for training and to reduce budget and predict situations. National defense simulations can be divided into three types: First, a live simulation is a distributed application to support testing and training with real, physical assets such as soldiers, airplanes, and tanks. Second, a virtual simulation is a distributed simulation to create simulators of physical assets that support testing and training in a virtual environment such as airplane simulators and tank simulators. Third, a constructive simulation is a distributed simulation to create synthetic environments in which simulation models of physical assets are used for testing and training primarily of procedures, doctrines, and commands. Each simulation has distinct features and advantages, but commonly suffers difficulties in describing the given situation and limitations in representing reality. To solve these problems, L-V-C integrated simulations are becoming increasingly important. A L-V-C integrated simulation can conduct a variety of operations / training scenarios by joint training, and improves the accuracy of training while reducing time and budget. L-V-C simulations typically use different middleware. To integrate them, interoperation between heterogeneous middleware is thus necessary. OMG Data-Distribution Service (DDS) is an emerging specification for publish-subscribe data-distribution systems and uses Live Simulation Middleware. The High Level Architecture (HLA) provides the specifications of a common technical architecture for use across all classes of simulations in the US Department of Defense and it uses Virtual Simulation Middleware. Distributed Interactive Simulation (DIS) is a government/industry initiative to define an infrastructure.
for linking simulations of various types at multiple locations to create realistic, complex, virtual worlds for the simulation of highly interactive activities and it uses Constructive Simulation Middleware. For an integrated L-V-C simulation, the different types of middleware must be able to communicate with each other. For this, we constructed an ontology that has relation information of APIs and parameters between HLA and DDS, and stored the ontology in a triple store. Furthermore, we developed ontology processing module for management of thee ontology. The ontology defines the concepts and relations (also referred to as ‘terms’) used to describe and represent an area of concern. Ontologies are used to classify the terms that can be used in a particular application, characterize possible relations, and define possible constraints on using those terms. Relation mapping by an ontology can offer scalability and portability because of the ontology’s characteristic. We construct ontology between HLA and DDS in the present study, but soon we will also construct a n ontology for HLA-DDS-DIS APIs and Parameters.

The remainder of this paper is organized as follows. In Section 2, we explain the overall architecture of our system. In Section 3 we introduce the ontology modeling of our system. In Section 4, we explain the storage of ontology data in our system. In Section 5, we explain the ontology processing module that makes queries to the ontology store.

2. Overall Architecture

Figure 1 represents the overall architecture of the ontology. It consists of DDS, HLA, Gateway, Interface, and Ontology System. DDS is L environment middleware and HLA is V environment middleware. The Gateway and Ontology System are required for interoperation of DDS and HLA. DDS-HLA interoperation means that the functionally same API in HLA is called when one of the APIs in DDS is called. The Gateway’s role is to support scalability, traffic engineering, and time management, and manage the DDS and HLA events. When the Gateway receives a DDS or HLA event, it sends the event to the ontology system to receive the mapping API and parameters.

Our ontology system can be divided into a storage part and an ontology processing module, and is based on a Jena framework. Jena is a Java framework for building semantic web applications. It provides extensive Java libraries for helping developers develop code that handles RDF, RDFS, RDFa, OWL, and SPARQL in line with published W3C recommendations. Although our Gateway’s environment is based on a c++ environment, the ontology system is based on Java. Because of this problem, we developed a c++ based interface using JNI. JNI is a standard programming interface for writing Java native methods and embedding the JVM into native applications. Section 2.1 shows the event flow of our system.

Figure 1. Overall Architecture.

2.1 System Flow
• DDS or HLA APIs called by an event. This event is sent to Gateway.
• Gateway sends event data(API, Parameter) to Interface.
• The Interface changes the C++ based event to a Java-based event by JNI and sends it to the ontology system.
• The ontology processing module sends a query to Jena in-memory storage (ontology storage) and receives mapping event data(API, Parameter).
• The ontology processing module sends the result to the Interface.
• The Interface changes the Java-based event to a C++ based event by JNI and sends it to the Gateway.
• The Gateway receives the result and calls the mapping API.

3. Ontology Modeling

3.1 Modeling Considerations
DDS-HLA interoperation means that the functionally same API and API related parameter in HLA is called when one of the API and API related parameters in
DDS is called. For this, an ontology modeling process is required. Ontology modeling is a process of constructing schema to show the relation of instances, and it has relations and layers. When designing an ontology designs, consideration of the relations between instances is necessary. In this paper, we designed an ontology that consists of functionally mapped relations between L simulation middleware (DDS) and V simulation middleware (HLA).

3.1.1 API Relation

DDS and HLA API have four types of relations, 1:1, 1:N, N:1, and N:M. First, a 1:1 functional mapping relation is defined as one-to-one relations. For example, when an HLA API is called, it is exactly related to calling of a DDS API. Second, a 1:N functional mapping relation is defined as one-to-N relations. For example, when an HLA API is called, it is exactly related to calling of N DDS API. Third, a N:1 functional mapping relation is defined as N-to-one relations. For example, when N HLA API is called, it is exactly related to calling of a DDS API. Finally, a N:M functional mapping relation is defined as N-to-M relations. For example, when N HLA API is called, it is exactly related to calling of M DDS API, such as 2:2 and 2:3 relations.

3.1.2 Parameter Relation

To functionally interoperate between DDS API and HLA API, parameters related to an API must implement mapping to a suitable parameter value. There are parameters that can be mapped to a certain value such as HLA Federation Execution Name and DDS domainId. But in some cases, there are parameters that cannot implement mapping to a certain value. An example is DDS QOS. In this paper, we map the parameters that can have a relation but we do not map the parameters that cannot have a relation. We designed parameters that cannot have a relation with a certain value to have a default value.

3.2 Ontology Model Overview

We designed an ontology model as illustrated in Figure 2 for interoperation between HLA and DDS middleware. The ontology model consists of five layers. The upper layer and lower layer are related with some relation; for example, has Function, has Service, has API, has Parameter. The upper layer’s instance can have some lower layer’s instances. The middleware layer can have an instance that consists of the name of the middleware, such as HLA. The function layer is used to distinguish the various APIs functionally, and it offers readability to people. The service layer is an abstract layer for functional mapping of APIs. Because of the service layer, it is possible to represent various relations between heterogeneous middleware APIs. The API layer consists of middleware API instances. The parameter layer consists of parameters that are related to APIs. More information on each layer is presented in section 2.3.

3.3 Layer

The layer concept is required to construct the ontology model. By using layers, the ontology has a more powerful expression. Our ontology model consists of 5 layers. Each layer has unique characteristics. We explain each layer in this section.
3.3.1 Middleware Layer

The Middleware Layer is the top layer. It consists of the name of the Middleware, such as HLA, DDS, or DIS. It can represent a Middleware instance. The Middleware Layer serves as a representative of lower instance.

3.3.2 Function Layer

The Function Layer is used to distinguish the various APIs functionally, and it offers readability to people. Examples include HLA Federation Management and Object Management. Users can ascertain the function of an API through this Layer.

3.3.3 Service Layer

The Service Layer is created for functional mapping of API relations that consist of 1:1, 1:N, N:1, or N:M. For example, when an HLA create Federation Execution and join Federation Execution are called, it is exactly related to calling of a DDS create_participant. In this case, assuming that the ontology model has a functional relation (sameAs relation) in the API layer, it cannot represent a N:1 relation because DDS create_participant is returned when one of the HLA APIs (join Federation Execution, create Federation Execution) is called. To solve this problem, the ontology model has a Service Layer. The Service Layer’s basic concept is comparison specific count variable (initialized to 0) to the Service layer’s number of API value (Figure 3). When HLA or DDS APIs are called, the Service Layer (upper of the HLA or DDS APIs)’s count variable increases to 1 and compared to the Service Layer’s number of API value. If the count value is the same as the number of API value, then the ontology processing module returns the mapping API and parameters.

3.3.3.1 Example

We explain the example in Figure 3. In this example HLA create Federation Execution and join Federation Execution are called, and this is exactly related to the occurrence of a DDS create_participant.

- Create Federation Execution is executed in HLA. At that time, the count value initialized 0 increase to 1. But it can’t call sameAs because it is different to Federation Join’s number Of Api value.
- Join Federation Execution is executed in HLA. At that time, the count value also increase 1, it becomes 2, and it is the same value of Federation Join’s number Of Api value. Ontology processing module can call sameAs to Participant Join and return create_participant API that has a relation with Participant Join by hasAPI.

3.3.4 API Layer

The API Layer consists of Middleware APIs. Calling functionally mapped APIs always occurs by the service layer’s same as call.

---

Figure 3. Ontology Example.
3.3.5 Parameter Layer

The Parameter Layer consists of parameters that are related with API. In the case of parameters that can be mapped to certain parameters. For example, HLA Federation Execution Name and DDS Domain Id, Federation Execution Name and Federation Execution Name are related by is Mapping to relation. But in cases where the parameter cannot be mapped to a certain parameter such a DDS QOS, it has a default value.

4. Storing Ontology

The modeled ontology has to be written in a specific file format. Although there are many file formats to write ontologies such as RDF/XML, Turtle, N-TRIPLE, etc., we used the N-Triples format because of its ease of representation and simplicity. N-Triples is a line-based, plain text format for encoding an RDF graph. We stored the N-triples file in Apache Jena triple store, and we confirmed it by Apache Jena Fuseki. This is shown in Figure 4.

5. Ontology Processing Module

The ontology processing module is for management of the ontology. When the gateway receives a DDS or HLA event, it sends the event to the Jena based ontology system and the ontology processing module executes a SPARQL query to the ontology. SPARQL is the query language of the ontology and it can be used to express queries across diverse data sources. After the query, it receives the mapping data and send the data to the gateway.

6. Conclusion

In this paper, we proposed ontology-based inter operation between DDS and HLA. We designed an ontology model for relations of between DDS APIs and HLA APIs, and store it to Apache Jena’s triple store. Furthermore, we develop an ontology processing module to query the ontology stored in Jena. We currently provide only DDS and HLA middleware mapping information for integration between L and V. However, we will soon map between DIS and HLA, and construct the Ontology by Inference. It will be able to provide DDS, HLA, and DIS mapping information for L-V-C.

7. References

1. Noseworthy JR. Supporting the Decentralized Development of Large-Scale Distributed Realtime LVC Simulation Systems with TENA. IEEE/ACM, IEEE/ACM Symposium on Distributed Simulation and Real-time Applications; 2010.
2. Pardo-Castellote G. OMG Data-Distribution Service: Architectural Overview. 23rd Int Conf Distrib Comput Syst Workshops; 2003 May.
3. Dahmann JS, Richard M, Fujimoto. Weatherly. The Department Of Defense High Level Architecture. Winter Simulation Conference; 1997.

4. IEEE Computer Society. IEEE Standard for Distributed Interactive Simulation–Application Protocols. SISO Standards Activity Committee. New York; 2012.

5. Available from: http://www.w3.org/standards/semanticweb/ontology

6. Available from: https://jena.apache.org/about_jena/about.html

7. Available from: http://docs.oracle.com/javase/7/docs/technotes/guides/jni/

8. Available from: http://www.w3.org/TR/n-triples/

9. Available from: https://jena.apache.org/documentation/fuseki2/index.html

10. Available from: http://www.w3.org/TR/rdf-sparql-query/