Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

DEVELOPING FEDERATION OBJECT MODELS USING ONTOLOGIES

Tarun Rathnam
Christiaan J.J. Paredis

Systems Realization Laboratory,
George W. Woodruff School of Mechanical Engineering,
Georgia Institute of Technology,
Atlanta, GA 30332, U.S.A.

ABSTRACT

The reuse of existing simulations in multiple federations is an important goal of distributed simulation frameworks. However, in order to reuse a federate, its simulation code often has to be modified so as to comply with the object and interaction representations defined in a corresponding Federation Object Model (FOM). Such modifications imply added time and effort, which diminishes the efficacy of reuse in federation development. In this paper, we present an ontology-based framework for modeling federates and supporting their reuse in multiple federations. Ontologies are used to specify the semantics of objects and interactions in federate domains in a formal, computer-sensible fashion. Using these formal semantics the relationships between federate simulation concepts are described in a reusable fashion. In doing so, a suitable federation representation for a set of related federate concepts and the required set of transformations between federate and federation representations are automatically derived.

1 INTRODUCTION

The use of federate simulations in multiple federations is an important goal of distributed simulation frameworks such as the High Level Architecture (HLA). The HLA Federation Development and Execution Process (FEDEP) model (Defense Modeling and Simulation Office 1999) prescribes the reuse of existing FOM components in federation development. This helps to reduce the overall time and effort invested to achieve interoperability between federate simulations. However, changing the composition of a federation requires some changes to the corresponding FOM. In turn, this almost always implies changes in the participating federates. All federates have to conform to the common representation for the full set of exchangeable information defined in a FOM. To ensure such consistency throughout the federation, each federate simulation's code may have to be modified and extended. Therefore, the efficacy of implementing reuse in HLA is marred by the fact that cost and time to achieve reuse are strongly affected by the uniformity of the federate representations (Nance 1999). The ability to reuse federates in federations with disparate FOMs without having to modify their individual representation will significantly simplify the federation development process.

In this paper, we present an ontology-based framework for modeling federates and supporting their reuse in multiple federations. Our approach improves federate reusability by formalizing the semantics of concepts (objects and interactions) defined in federate simulations. Using these formal semantics, the relationships between federate simulation concepts are described in a reusable fashion. In order to allow a federate to participate in several federations without modification, procedures to transform Simulation Object Model (SOM) information representations into those of the target FOM are requisite . Our framework applies the knowledge of the relationships between federate simulation concepts (captured in an ontology) to generate a suitable FOM and the required set of transformation procedures for a given federation. In this manner, we facilitate reuse in federation development by way of semantically rich information models.

2 RELATED WORK

Significant research and development is already underway to enable the participation of federates in multiple federations in a seamless fashion. The current practice is to standardize FOM representations through the use of reference FOMs and an Object Model Data Dictionary (Bouwens, Miller, Scrudder and Lutz 1998). This approach only partly solves the problem because it imposes restrictions on the ability of federate developers to specify their own information representations.

An Agile FOM Framework (AFF) has been developed to allow federates with disparate SOM representations to participate in multiple federations (Macannuco 1998). The AFF provides a federated simulation with a set of classes that automatically map the internal SOM representations to the external FOM representation. This framework uses converter objects to manipulate data between FOM and SOM representations. Converters are application level code, defined by extending a base converter class. A set of basic converters is included to perform unit and enumerated type conversions.

The AFF is conceptually similar to the framework presented in this paper. Both frameworks are based on the notion of federate reuse through the definition of transformations between SOM and target FOM representations. Our approach using ontologies offers the key advantage of capturing the relationship between representations in a formal, declarative fashion that enables automation to a greater degree. The AFF assumes that federation developers specify a FOM representation for a given federation; whereas our framework supports automated FOM generation based on a given set of SOM representations and relationships between them. Further, in the AFF the specification of custom converters is not automated. In the approach presented in this paper, the required transformation routines are *inferred* based on existing knowledge of relationships in the federation domain, prompting federation developers for additional knowledge as required.

Another approach to support reuse in federation development has been implemented in Base Object Models (BOMs) (Gustavson 1998). BOMs are meant to serve as building-blocks that enable engineers to design component-based federations. A BOM is a component of a simulation that can encapsulate objects, their attributes, interactions involving those objects and an associated set of parameters. BOMs include meta-data that define their focus, intent and origin. BOMs may be developed for individual domains and reused to simplify and speed-up SOM and FOM construction.

While the BOM framework facilitates reuse in the development of a FOM, the issue of reusing federates without modification is not addressed. However, the BOM framework can potentially be used to implement rapid integration of federate simulations through the definition of BOM-level mappings between SOMs and FOMs. Such mappings could be specified by identifying similar "patterns" in the structure of the SOM and FOM (Base Object Model Study Group 2001). To achieve this mapping functionality in an automated fashion, a richer set of BOM meta-data (semantics) would be required. The ontology based framework presented below enables the capture of such semantics, thereby supporting the automated mapping of SOM and FOM concepts.

3 ONTOLOGY-BASED FOM DEVELOPMENT FRAMEWORK

An ontology is an explicit specification of a conceptualization i.e. the objects, concepts, and other entities that are assumed to exist in a domain and the relationships that hold among them (Gruber 1993). The key ingredients that make up an ontology are a vocabulary of basic terms, a precise specification of what those terms mean and how they relate to each other. By organizing knowledge in a discrete layer for use by information systems, ontologies enable communication between computer systems in a way that is independent of the individual system technologies, information architectures and applications (Berners-Lee, Hendler and Lassila 2001; TopQuadrant 2003). This is why we turn to ontologies to alleviate the difficulties faced in federate reuse. We use ontologies to model federate and federation domains (analogous to current HLA SOM and FOM) based on a common set of concepts and relationships between them. This enables the inference of relationships between individual federate representations of shared objects, attributes, interactions and parameters.

A high-level illustration of the framework for ontology based federation development is provided in Figure 1. The major components involved are the simulation (federate) ontologies (SONT), a target federation ontology (FONT) and a meta-model that corresponds to the OMT, called the World Ontology. The SONT specifies the object-attribute architecture corresponding to a given federate simulation. The FONT specifies a *common* representation for all objects and interactions that are shared among different federates, and captures the relationships between the federate and common representations. The World Ontology contains meta-data and specifies the structure of objects, attributes, interactions, parameters and data types. It also includes data structures to capture the relationship between these. Finally, this ontology includes a set of primitive data types and defines the relationships between them.

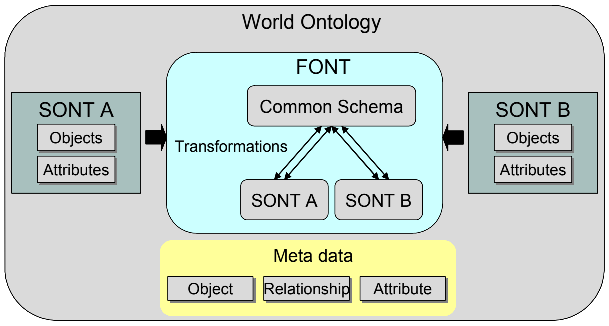

Figure 1: Ontology-Based FOM Development Framework

Based on the structure provided in the World Ontology, SONTs are specified by domain experts who play a major role in the development of a given simulation model. This process is analogous to documenting a SOM in current HLA practice. However, a SONT captures concepts and relationships between them in a formal, computer-sensible fashion that is much richer than the unstructured text that comprises a SOM. Unlike a SOM, a SONT contains knowledge that a computer can use to make inferences. Once a SONT has been specified, it does not have to be changed when it participates in different federations.

According to the FEDEP model, the federation development is centered on the specification of a FOM. Once the FOM has been specified, the individual federates are modified to be consistent with the FOM representation. Contrary to this approach, we propose the extraction of the FONT based on the representation of the objects in the participating SONTs and the relationship between them. When the federation developers reach the point in the FEDEP process where they have decided on a set of federates, they may access the corresponding SONTs and specify which objects and interactions relate to each other (in terms of attribute and parameter relations). Once this information has been specified, a suitable common representation for the related entities is derived automatically based on their individual federate representations.

Along with automated FONT generation, the required transformation mechanisms to convert data to and from federate to common representations are created. This task amounts to representing the relationship between two representations of a shared entity in a procedural format. Arriving at this relationship is where knowledge reuse comes into play. The World Ontology provides a common conceptualization of terms as well as a set of data types and relationships between them. SONTs and FONTs are defined using the vocabulary defined in this common conceptualization. Therefore the relationship between shared entities can be derived based on the relationships defined at a higher level of abstraction (in the World Ontology). That is, the transformation routine between two representations of a given entity is inferred from knowledge about the relationships between their data types.

The ontology-based federation development process can be summarized in the following steps:

- Define World Ontology (one-time task)
- Define SONTs based on World Ontology
- Generate FONT
    - Determine common representation
    - Generate transformation routines.

The following sub-sections detail the above-mentioned steps in the context of our implementation of this framework in the Protégé ontology development software tool. A simple example federation development is also presented in support of the discussion.

3.1 Defining the World Ontology

The world ontology is analogous to the HLA Object Model Template (OMT) (IEEE 2000): it defines the information schema for specifying objects, attributes, parameters and interactions. This meta-model is defined in terms of the frame-based representation supported by Protégé (Noy, Fergerson and Musen 2000), which is similar to the table architecture defined in the current OMT specification. Concepts in Protégé are specified as frames (classes), defined in terms of their slots (attributes). Individual entities are represented as instances of these classes. Finally meta-classes and meta-slots can be defined as templates for specifying specialized classes and attributes. Figure 2 is a graphical illustration of the world ontology implemented in Protégé.

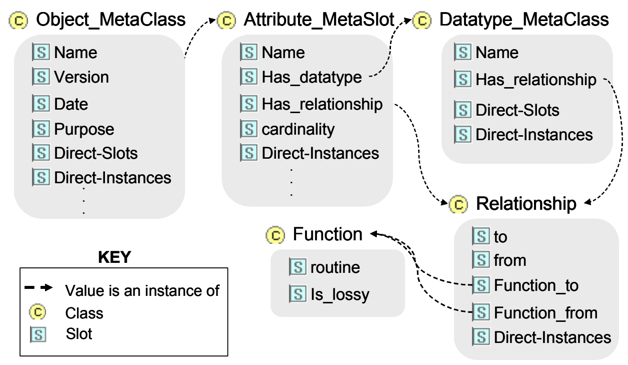

Figure 2: Information Schema in the World Ontology

Templates for HLA objects and interactions are specified using meta-classes. Therefore, individual objects and interactions are defined as classes (instances of the Object/ Interaction meta-classes). This allows SONT developers to specify a hierarchy of objects and interactions, thus capturing the equivalent information specified in HLA object and interaction class structure tables in an ontology. The meta-class for objects includes slots that correspond to all fields specified in the HLA Object Model Identification Table.

Along the same lines, templates for HLA attributes and parameters are specified using meta-slots. This enables the instantiation of attributes as slots of individual Objects. These meta-slots are composed of their own set of slots corresponding to the HLA attribute/ parameter table fields.

The world ontology also includes a *relationship* class to hold the required information about the relationship between attributes or parameters in a FONT. The relationship between a particular attribute and its common representation is represented as an instance of this class. Every attribute has the slot *has_relationship* whose value is an instance of the *relationship* class. This class consists of the following slots:

- *to*: whose value is the target attribute or parameter,
- *from*: whose value is the subject attribute or parameter,
- *function_to:* whose value is an instance of the *function* class and holds information about the transformation routine from the subject attribute or parameter to the specified target,
- *function_from:* which is analogous to *function_to*, except going from the target attribute or parameter to the subject.

The *function* class consists of two slots: *routine* and *is_lossy*. The *routine* slot contains the procedure to convert instances of one attribute or parameter to the other. *Is_lossy* has a Boolean value that indicates whether the transformation from one representation to the other leads to a loss of information. In Section 3.3, we discuss the use of *is_lossy* to determine the FONT representation for a set of related SONT attributes or parameters.

The data type of a given HLA attribute indicates the class of which that attribute is an instance. Since data types are classes, a template for data types is provided as a meta-class. This meta-class includes the *has_relationship* slot to capture the relationship between custom data types.

Finally, we instantiate a set of data types that are expected to be used consistently across all SONTs and FONTs. As an example, we have defined a set of units data types to enable individual SONTs to specify their own unit of measure for different quantities. The relationship between two units of a certain measurable quantity is of multiplication or division by a constant conversion factor. A certain system of measurement is chosen as a reference to which all conversion factors are determined. Novak (1995) has shown that with the knowledge of the conversion factors relating a set of simple units (Meter, Second, Kelvin etc.), the conversion factor for any composite unit (a product or quotient of simple units, such as meter per second) can be derived. Hence, we include a slot *conversion_factor* in the definition of simple unit data types and capture the representation of composite units as a product of simple units. The algorithm for deriving a procedural conversion between different units is implemented as part of the system that generates transformation routines. Note that since units are predefined data types, we do not have to specify a value for their *has_relationship* slots; that is reserved for specifying relationships between custom data types in a FONT.

The relationship between data types is the primary knowledge that is reused to determine the required transformation routines. From the relationship between the data types of two attributes that are otherwise equivalent, the procedure for transferring values between these two attributes can be inferred with no further input from the federation developer.

Like the OMT, the world ontology is defined once and for all, and every SONT and FONT definition must conform to it. The following section discusses SONT development based on the world ontology.

3.2 Defining a Simulation Ontology (SONT)

The ontologies for individual simulations are defined using the information constructs defined in the world ontology. That is, we represent objects as classes—instances of the object meta-class defined in the world ontology. Similarly, attributes are represented as slots of object classes. To enable this instantiation, the world ontology must be included as part of each SONT domain, so that a consistent definition of the terms object, interaction, attribute, parameter and data type exist. Objects and interactions in a SONT are arranged in a hierarchy, such that subordinate objects and interactions inherit the attributes/parameters of their parents.

As an example SONT specification, consider the specification of a SONT for a simple traffic simulation (Figure 3). The object *vehicle* is created with the attribute *position*, whose data type is *2-D coordinate*. Each of these entities is specified as an instance of its respective meta-class. The data type *2-D coordinate* consists of two members: *x* and *y* of the unit data type *meter*. The ontology automatically captures the relationships between the three: attribute *position* is a member of class *vehicle* and its value is an instance of class *2-D coordinate*. At this stage, the *has_relationship* slot of *position* and *2-D coordinate* have not been assigned values, i.e. the *relationship* class has no instances. Obviously, this information will not be specified for a stand alone federate; it is provided in a FONT when different federate objects (and data types) are related to their common counterparts.

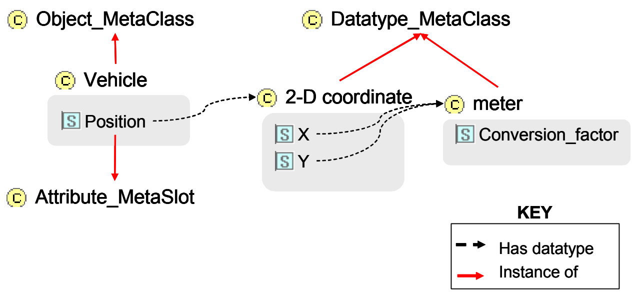

Figure 3: Traffic Simulation SONT Specification

3.3 Generating a Federation Ontology (FONT)

A federation ontology (FONT) serves as a common representation to and from which federates can convert shared information. Therefore the FONT consists of (its own representation of) all shared objects, interactions, attributes and parameters in a federation. Further, this ontology must include the definition of the relationships between the SONT and common representations of shared concepts. In order to specify a relationship between two entities, both entities must be defined in the same ontology. Therefore, the FONT includes all SONTs plus a common schema that is a liaison between individual SONT representations of shared concepts (Figure 1). The overall FONT generation process model, illustrated in Figure 4, is discussed below.

The first step in FONT generation is creating a new ontology that includes all the SONTs that are part of the federation. Following this, the federation developers must specify the knowledge as to which SONT objects relate to (publish or subscribe to) each other. When a relationship between two or more SONT objects is specified, a *common or shared* representation for those objects is created. Ultimately, all relationships are defined between federate and common representations of shared concepts. However, the federation developer specifies relationships directly between objects in any two SONTs.

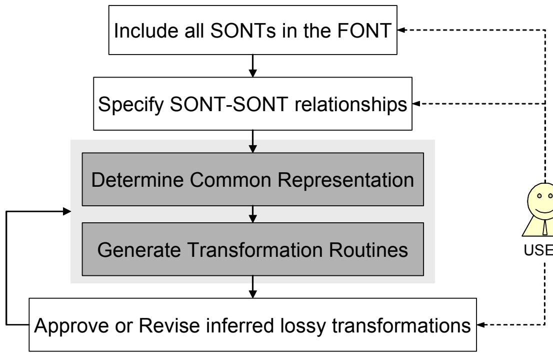
Include all SONTs in the FONT
Specify SONT-SONT relationships
Determine Common Representation
Generate Transformation Routines
USER
Approve or Revise inferred lossy transformations

Figure 4: FONT Generation Process Flow

Given the relationships specified by the user, new relationships can be inferred automatically by composing existing relationships together. The complete set of relationships are used to determine the appropriate common representation (Section 3.3.1). Following this the procedural transformations associated with these inferred relationships are also composed automatically (Section 3.3.2). During these steps, the user is prompted to provide additional knowledge about transformations as required. Having completed these steps, the user is presented with the set of inferred relationships and transformations, so as to either approve them or revise them if an available direct relationship is preferable. If revisions are made, the common representation, and the associated transformations are recomputed. In this manner, the process of defining relationships in a FONT is an iterative process that employs feedback from the user to refine automatically generated common representation and transformation routines.

Having presented the overall FONT generation process flow, the specifics of automatically arriving at a common representation and associated transformations is discussed in greater detail in the following sub-sections.

3.3.1 Determining the Common Representation

As mentioned above, the relationship between SONT objects is captured in the FONT as a relationship between each object and a corresponding common object. To determine the attributes of this common object, the federation developer must specify which attributes of the individual SONT objects relate to each other. For every set of relating SONT attributes, a common attribute is automatically instantiated. While it makes sense for this common attribute to correspond directly to one of the SONT attribute representations (this ensures that at least one of the transformation routines will be trivial), it is important to choose a representation that avoids any unnecessary loss of information when exchanging data in a federation. The importance of this choice may not be evident when there are only two related attributes; in fact it is irrelevant in this case. However, this choice becomes significant when three or more SONT attributes in a federation relate to each other. For example, if the SONT attribute *position* of data type *2-D coordinate* relates to attribute *location* (in another SONT domain) of type *3-D coordinate*, and attribute *point* also of type *3-D coordinate*, the corresponding common attribute must be of type *3-D coordinate*. If it is selected to be of type *2-D coordinate*, then there is an avoidable loss of information. Both attributes *location* and *point* have three coordinates, yet when *location* subscribes to *point* (or vice-versa), the value is converted from *3-D* to *2-D* (common representation) and back to *3-D*, resulting in a loss of the third coordinate's value. To avoid this scenario, the common representation of a set of related attributes *should* have a representation that does not lead to any avoidable loss of information.

In order to determine which SONT representation of a shared attribute is the appropriate common representation, we introduce the notion of *lossiness*. A transformation from one representation to another is *lossy* if any information is lost in that transformation. In the example above, the transformation from attribute *location* to *position* is lossy (while the inverse is not). The information about lossiness is captured in the *is_lossy* slot of a given function. In a relationship where *from* = *position* and *to*= *location*, the value of *function_to* (an instance of the function class) has *is_lossy* = true, while that of *function_from* has *is_lossy* = false. The common representation for a set of related attributes is determined as the representation that leads to the *fewest* number of lossy transformations. In the event that there are several SONT representations that lead to the same number of minimal lossy transformations, any of them may be picked as the common representation.

The lossiness in a transformation between two attributes is determined in terms of the lossiness in the transformation between their respective data types. Therefore, in order to determine if information is lost in a transformation between two SONT entities, the transformation procedures between their data types have to be derived first. The complexity of determining data type transformations depends on whether those data types are primitive or custom, leading to the following two cases:

Case 1: Both data types are primitive. The knowledge of the relationship between primitive data types is already encoded into the software system that extracts transformation routines. Therefore, the required transformation routines are easily created in such cases. For example the transformation from data type *meter* relates to data type *foot* is:

```
foot meter_to_foot (meter input) {
foot output;
output=(input/foot.conversion_factor
*meter.conversion_factor);
return output;}
```

The knowledge of lossiness between primitive classes is already encoded in the software system. Hence, no input is required from the user to determine that a valid transformation between unit data types is never lossy.

Case 2: One or both data types are not primitive. If the relationship involves transformation between custom data types, then the knowledge about the relationship between these data types does not preexist. This knowledge cannot be created automatically; it must be defined by the user. If all the individual fields of the related custom data types are primitive data types, then with knowledge as to which fields relate, the required relationship can be derived automatically. For example, consider that the data type *3-D coordinate* has fields (*x*, *y* and *z* of unit data type *foot*). The relationship between *3-D coordinate* and *2-D coordinate* (*x* and *y* in *meters*) can be derived automatically if the user specifies that the respective *x* and *y* fields equate to each other. Since these fields have primitive unit data types, the transformations between the custom data types can be derived automatically, such as:

```
3D 2D_to_3D (2D input) {
3D output;
output.x = meter_to_foot(input.x);
output.y = meter_to_foot(input.y);
output.z= 0;     // user specified default
return output; }
```

In general, if one or more fields are not primitive or the relationship between them is not that of equivalence, the user must explicitly define the relationship between the custom data types. At the same time, the lossiness in the data type transformation routines must also be specified.

Note that the user is not compelled to specify relationships between all data types involved in a relationship. The relationship between two data types can be inferred transitively as a chain of relationships. That is, if a transformation from data types A to B and B to C exist , the transformation from A to C can be derived from these. In such a case, it is possible that the derived transformation is lossy when in theory it can be defined in a non-lossy form. The user is made aware of such lossiness in derived transformations and given the option of explicitly specifying a less lossy transformation (Figure 5).

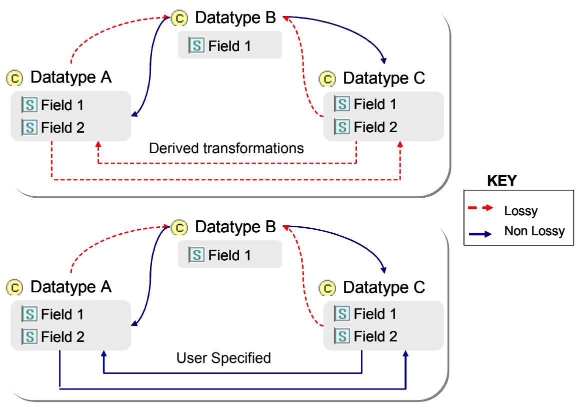

Figure 5: Transitively Derived Transformations

With the knowledge of lossiness in data type transformations for a set of related SONT attributes or parameters, the common representation for those attributes or parameters can be determined. Consider the example relationship between the attributes *position*, *location* and *point*. Based on lossiness in transformations between *2D* and *3D coordinates*, it is automatically determined that the transformation from *location* or *point* to *position* is lossy (while the reverse is not) and that from *location* to *point* and back is not. The smallest number of lossy transformations occurs when the common attribute corresponds directly to either *location* or *point* (Figure 6). Note that when reusing FONTs, the representation that is deemed fitting for a given common attribute is subject to change if one or more federates are added to or leave the federation. At all times, the common attribute's representation should lead to the smallest number of lossy transformations.

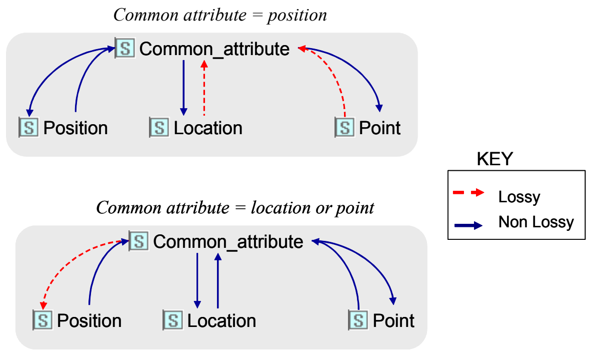

Figure 6: Selecting a Common Representation that Leads to the Fewest Number of Lossy Transformations

Once the common representation for all shared objects, interactions (and their respective attributes and parameters, respectively) have been instantiated, they must be arranged in a hierarchy. This step is vital to facilitate inheritance in publication and subscription of federate objects or interactions. That is, if a certain object subscribes to another SONT's parent object, it should be notified of all updates to the children of that parent object. The set of common objects and interactions are arranged into a hierarchy based on Classification—the process of constructing a concept hierarchy in which more general concepts are located above more specific ones according to the subsumption order . The subsumption relationship between two objects in a schema is defined such that an object B *subsumes* an object A if the set of attributes that comprise B includes the set of attributes that comprise A. In this case, object B is a refinement of object A, or A is the parent of B. Algorithms to perform subsumption tests have been developed by Schmolze and Lipkis (1983) and can be leveraged to arrange common representations of shared objects and interactions in a hierarchy. Note that object-by-object comparison and subsumption testing can become time-consuming when the number of shared entities in a federation is large. To avoid this problem, a more complex, but efficient parallel classification algorithm (Kim 93) can be employed to arrange the automatically generated objects and interactions.

3.3.2 Generating Transformation Routines

At this point, a common representation between all shared SONT objects, interactions, attributes and parameters in a federation is generated and ordered. The last piece of the puzzle is creating the procedural knowledge of the relationships between the federate and common attributes or parameters. These routines are represented as a chained 2-step procedure: one to convert between data types and the other to convert between the related concepts. As an example, consider that a federation developer specifies a relationship between the attribute *radius* of type *meter* in one SONT and *diameter* of type *foot* in another. Clearly, these two concepts are related, but they are not the same. The user must specify the knowledge as to how these two concepts relate. Ideally, we would like the user to specify this relationship in a declarative fashion, from which the transformations in either direction can be derived (e.g. `radius - (diameter/2) = 0`). However, a declarative relationship between two entities can be converted into two procedures (to perform transformations in either direction) only if that relationship is analytically invertible. Hence we assume that whenever the user explicitly specifies a relationship, he or she does so in a procedural form (e.g. `radius = diameter/2; and  diameter = radius*2;`). The transformation from radius to diameter is derived as:

```
Function_to.routine:
foot radius_to_diameter (meter radius) {
foot diameter;
diameter= (meter_to_foot(radius))*2;
return diameter;}
```

Note that the user is not constrained to always specify a relationship between a SONT representation and what ends up being the common representation in a set of related attributes. He or she may specify the relationship between any two SONT relationships. Just as with data types, relationships to and from the common representation can be derived transitively from existing relationships, if a sufficient set of relationships exists.

3.3.3 Example FONT Development

We conclude the discussion on FONT and transformation generation with an example FONT development scenario. Consider that a federation of simulations is to be developed that includes the previously defined traffic simulation (section 3.2) and a wireless network simulation. The goal of this federated simulation is to simulate traffic behavior when vehicles can communicate with each other using wireless and GPS technologies. The SONT for the wireless network simulation contains the object *Node* having the attribute *location*, whose data type is *3-D coordinate*. The federation developer wishes to relate the traffic simulation's *vehicle* object to *Node*, in terms of a relationship between *position* and *location* attributes. To do so, the following steps are undertaken:

- The set of SONTs that are part of the federation (traffic and wireless network SONTs) are specified. These SONTs are then automatically included in the FONT being developed
- The user indicates the existence of a relationship between *vehicle* and *node* objects in terms of their attributes *position* and *location*. A corresponding *common_object* is created in the FONT
- The federation developer specifies that the relationship between attributes *position* and *location* is that of equivalence. Since the fields of custom data types *2-D coordinate* and *3-D coordinate* have primitive data types, the user is prompted to specify the relationship between these fields
- An equivalence relationship between the *x* and *y* fields of the data types is specified. A *relationship* instance is automatically created between *3-D* and *2-D coordinate* data types. The *function_to* and *function_from* slots' values are automatically derived as discussed in Case 2. The *is_lossy* values for these functions are specified by the user
- Since only two attributes are being related, the choice of the common representation is inconsequential. Assume that the attribute *common_attribute* is created with data type *3-D coordinate*
- The *function_to* slot's value for a relationship instance *from position to common_attribute* is specified as:

```
Function_to.routine:
3D position_to_common (2D input){
3D output;
output = 2D_to_3D (input);
return output;}
```

- The *function_from* slot for this relationship is also derived in a similar fashion, as are the functions for the relationship between *location* and *common_attribute*. The resultant FONT information structure is depicted in Figure 7.

Having defined all required transformation routines, the FONT contains the complete set of information required to enable consistent data exchange in a federated simulation. The routines defined here are in pseudo-code; in general the federation developer will specify an object-oriented programming (OOP) language syntax in which the

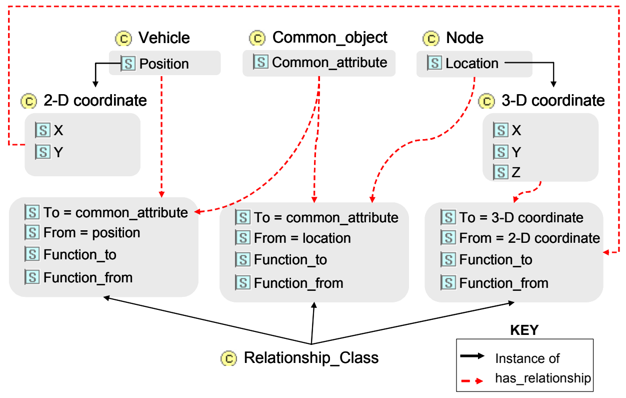

Figure 7: Example FONT Information Structure

transformation routines are to be represented. The FONT can be passed to the RTI as a set of classes and functions in the specified OOP language.

4 CONCLUDING REMARKS

The ontology-based framework presented in this paper facilitates simplified reuse of federates in multiple FOMs. Ontologies allow us to capture knowledge about object and interaction representations (and the relationships between them) in a formal manner. Applying this knowledge, we have defined a process for automatically arriving at a federation object model and a set of procedures to transfer data between federate and federation representations. The end-result is a semantically rich model that spans the entire federation and contains all the information required to enable consistent data transfer. This approach to federation development offers considerable benefits when one or more federates are being reused in a new federation. In federations where reuse is not prominent, this approach offers a simplified way of specifying complex relationships between a large number of entities.

Using ontologies as simulation information models, the process of FOM development has been significantly trivialized. However, there is still room to simplify the FEDEP process via ontology-based modeling, given the constantly advancing state of art in ontology management. Specifically, systems to automatically specify mappings between various ontologies have been developed to support information processing across the widely-distributed semantic web. For example, GLUE uses machine-learning techniques to automatically identify matches between similar concepts in two or more ontologies. By applying such systems to ontology-based federation development, relationships between SONTS could conceivably be determined completely autonomously, resulting in the ultimate simplification of the FOM development process.

An important aspect of this framework that has not yet been addressed is the interface between the federation developer and the underlying software. Given the iterative nature of the FONT generation process, this interface (currently being developed) should provide an intuitive method by which users can specify knowledge and provide feedback to the system. Finally, there is still the issue of using the information in a FONT to actually manage consistent data transfer at run-time. In this paper, we have not focused on the implementation of an RTI that can avail of the relationships defined in the FONT. The development of a next-generation architecture that uses the transformations captured in a FONT to provide real-time conversions between disparate representations is on going research at Georgia Tech (Fitzgibbons and Fujimoto 2004).

ACKNOWLEDGMENTS

We would like to acknowledge the members of the Systems Realization Laboratory and the Parallel and Distributed Simulation Group at the Georgia Tech for their comments and feedback on this work. This work is supported by Sandia National Laboratories under contract A0356-253878. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

Base Object Model Study Group. 2001. BOM Methodology Strawman (BMS) Specification. Simulation Interoperability Standards Organization, Orlando, FL.

Berners-Lee, T., J. Hendler and O. Lassila 2001. The Semantic Web. *The Scientific American Magazine* (279).

Bouwens, C., R. Miller, R. Scrudder and R. Lutz. 1998. Object Model Development: Tools and Techniques. Simulation Interoperability Workshop.

Defense Modeling and Simulation Office. 1999. High Level Architecture: Federation Development and Execution Process (FEDEP) Model.

Doan, A., J. Madhavan, P. Domingos and A. Halvey. 2002. Learning to Map Between Ontologies on the Semantic Web, In *Proceedings of the International Word Wide Web Conference*, Hawaii.

Fitzgibbons, J. and R. Fujimoto. 2004. IDSim: An Extensible Framework for Interoperable Distributed Simulation, In *Proceedings of the International Conference on Web Services*, San Diego, CA.

Gruber, T. 1993. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Workshop on Formal Ontology, Padova, Italy.

Gustavson, P. L., J. P. Hancock and M. McAuliffe 1998. Base Object Models (BOMs): Reusable Component Objects for Federation Development, Simulation Interoperability Workshop.

IEEE 2000. Std 1516.2-2000, Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification.

Kim, J.-T. 1993. Classification and Retrieval of Knowledge on a Parallel Marker-Passing Architecture. In *IEEE Transactions on Knowledge and Data Engineering*, 5(5): 753-761.

Macannuco, D., B. Dufault and L. Ingraham. 1998. An Agile FOM Framework. Simulation Interoperability Workshop.

Nance, R. E. 1999. Distributed Simulation With Federated Models: Expectations, Realizations and Limitations. In *Proceedings of the 1999 Winter Simulation Conference*, ed. P.A. Farrington, H.B. Nembhard, D.T. Sturrock and G.W. Evans, 1026-1031. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Novak, G. S. 1995. Conversion of Units of Measurement. In the *IEEE Transactions on Software Engineering*, 21(8): 651-661.

Noy, N., R. Fergerson and M. Musen. 2000. The Knowledge Model of Protege 2000: Combining Interoperability and Flexibility. Stanford Medical Informatics Technical Report, Stanford University, Stanford, CA.

Schmolze, J. G. and T. Lipkis. 1983. Classification in the KL-One Knowledge Representation System. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany.

TopQuadrant. 2003. Semantic Integration: Strategies and Tools. TopQuadrant Inc. Whitepaper, Beaver Falls, PA.

AUTHOR BIOGRAPHIES

TARUN RATHNAM is a graduate research assistant at the Systems Realization Laboratory at Georgia Tech, where he is working towards his Master's degree in Mechanical Engineering. His research is focused on the application of semantic technologies to support information management in engineering design and analysis. His email address is <gtg378k@prism.gatech.edu>.

CHRIS PAREDIS Ph.D., is an Assistant Professor in the G.W. Woodruff School of Mechanical Engineering at Georgia Tech. He received his M.S. degree in Mechanical Engineering from the Catholic University of Leuven (Belgium) in 1988, and his M.S. and Ph.D. in Electrical and Computer Engineering from Carnegie Mellon University in 1990 and 1996, respectively. From 1996 to 2002, he was a Research Scientist at the Institute for Complex Engineered Systems at Carnegie Mellon University. Dr. Paredis has a broad, multidisciplinary background. In his research, he combines aspects of information technology, simulation, and systems theory to support the design of mechatronic systems. The goal of his current research is to develop an integrated IT framework for simulation-based design. He is also still active in the robotics and mechatronics areas in which he did his Ph.D. research. His email address is <chris.paredis@me.gatech.edu>.