Adding the Infrastructure Class Hierarchy To the EXHORT Framework for Object-Oriented Deployment Simulations

James F. Burke, Jr.
Charles N. Van Groningen
Mark J. Bragen
Charles M. Macal
Argonne National Laboratory
9700 S. Cass Ave., Bldg. 900
Argonne, IL 60439
630-252-9009
jay@anl.gov

Keywords:
class framework
class hierarchy
Common Object Request Broker Architecture (CORBA)
Defense Information Infrastructure Common Operating Environment (DII COE)
deployment
High Level Architecture (HLA)
logistics
object-oriented analysis and design
simulation
transportation

ABSTRACT: *One of the objectives of the U.S. Department of Defense is to standardize all classes used in object-oriented deployment simulations by developing a standard class attribute representation and behavior for all deployment simulations that rely on an underlying class representation. The EXtensive Hierarchy and Object Representation for Transportation Simulations (EXHORT) is a class framework composed of two hierarchies that together constitute a standard and consistent class attribute representation and behavior that could be used directly by a large set of deployment simulations. The first hierarchy, the Transportation Class Hierarchy (TCH), was submitted to the Army Modeling & Simulation Office's (AMSO) Army Standards Repository in 1999 and presented at the Fall Simulation Interoperability Workshop in the same year. The second hierarchy, the Infrastructure Class Hierarchy (ICH), describes the encapsulation of the rest of the defense transportation system and is the primary focus of this paper. The entire EXHORT framework lets deployment simulations use the same set of underlying class data, ensures transparent exchanges, reduces the effort needed to integrate simulations, and permits a detailed analysis of the defense transportation system.*

1. Introduction

1.1 Background and purpose

Logistics and mobility have become increasingly important in our rapidly changing world. Since September 11th, it has become even more critical to move soldiers and supplies with limited resources in support of the war on terrorism. After the Cold War, U.S. military forces were pulled back to the continental United States, so contingency planning for deploying forces to overseas locations took on central importance. Besides the new war-fighting missions it appears are on the horizon, the U.S. Department of Defense (DoD) has been called upon to perform an ever-increasing number of non-traditional missions, including peacekeeping and disaster relief, both in the United States and overseas. Such needs have motivated investment in new simulation models and technologies, because it is more efficient and effective to initially simulate deployments on a computer than to first test them in the real world. Accordingly, over the past 5 to 10 years, the DoD has made greater use of simulations to help meet its mis-

sion objectives. These tools are used to analyze, plan, train for, and execute deployments.

In conducting analyses of the defense transportation system, deployment models and simulation systems rely heavily on data pertaining to military cargo, transportation assets, and infrastructure. Currently, throughout the simulations in the military transportation/deployment community, the military cargo and transportation asset data are entered into simulation models separately and in various formats. This practice results in many data inconsistencies and makes simulation integration very difficult and time-consuming.

To get the greatest return on investment, the DoD has attempted to team sometimes very different simulations in various combinations to satisfy a diverse and ever-evolving set of user needs, a practice that has mandated a rigid adherence to standards and a greater focus on simulation interoperability guidelines. In response, the Army Transportation and Deployment Community has moved to standardize all classes used in object-oriented deployment simulations that rely on an underlying class representation. This paper describes a class framework called the EXtensive Hierarchy and Object Representation for Transport Simulations (EXHORT), which is designed to standardize the classes used in object-oriented deployment simulations.

EXHORT consists of two hierarchies that together constitute a standard and consistent class attribute representation and behavior that could be used directly by a large set of deployment simulations. The hierarchies are unusual in that they provide for a detailed design class factoring, rather than merely an aggregate analysis view. In order to encapsulate the characteristics of multiple deployment simulations written at different levels of aggregation, the class abstractions in EXHORT needed to be specified at a design level view rather than at the higher, analysis level view that is more typical of community object modeling. This greater detail was incorporated because we decided that these classes were "key abstractions" and are well suited for use in new federations of simulations where reusability and intercommunication for new simulations is important.

The two EXHORT hierarchies will allow many deployment simulations to use the same set of underlying class data and will significantly reduce the effort needed to integrate simulations and analyze the defense transportation system. The first hierarchy, the Transportation Class Hierarchy (TCH), covers commercial transportation assets and military cargo, and is summarized in Section 3 [1]. This hierarchy is already an Army Modeling and Simulation Office (AMSO) Deployment/Redeployment Community Standard [2].

The second hierarchy is the Infrastructure Class Hierarchy (ICH), which deals with locations where activities are performed and with the physical infrastructure, such as a motor pool at a fort or a berth at a port. These areas contain the resources needed to support deployment (e.g., ramps, rough terrain container handlers, cranes, and forklifts). Both hierarchies are needed to describe major portions of the defense transportation system. The goal is to define EXHORT so it could provide a standardized code structure for object-oriented deployment simulations to ensure meaningful data exchanges.

1.2 Deployment simulations

Argonne National Laboratory (Argonne), in collaboration with the Military Traffic Management Command Transportation Engineering Agency, has developed three deployment simulation systems (ELIST — Enhanced Logistics Intra-theater Support Tool, PORTSIM — Port Simulation Model, and TRANSCAP — Transportation System Capability Model) and prototyped a fourth (CITM — Coastal Integrated Throughput Model). These systems simulate the processes required for the different components of the defense transportation system. While developing these force projection simulations, Argonne has gained experience in working with transportation logistics, class frameworks, and simulation interoperability.

- **ELIST** — Simulates military transportation movement across a theater of operations. ELIST also predicts whether infrastructure and transportation assets can support the war-fighting commander's required force delivery dates.

- **PORTSIM** — Simulates seaport operations and determines port throughput. The simulation also identifies system and infrastructure constraints and port-specific force clearance profiles.

- **TRANSCAP** — Simulates installation transportation operations and computes time-phased outloading capability. It will also identify system and infrastructure constraints and installation-specific force departure profiles.

- **CITM** — Simulates logistics for over-the-shore operations at austere ports linking the bare beach to the transportation infrastructure.

1.3 Content and organization

This paper summarizes EXHORT's first hierarchy, the TCH, describes the second, the ICH, and describes why it

is a helpful tool in modeling major portions of the defense transportation system.

Section 2 introduces programming standards and their relation to EXHORT. Section 3 summarizes the TCH, while Section 4 provides information about the classes of the ICH. Finally, Section 5 describes the benefits and future directions of EXHORT.

2. Programming Standards

Programmers use standards as guides to designing "components" that ensure reusability and interoperability among simulations. When developing state-of-the art models and simulation tools for DoD, programmers must follow the DoD standards and compliance mandates that are intended for all software applications. Two of these are the Defense Information Infrastructure Common Operating Environment (DII COE) and the High Level Architecture (HLA). For real-time distributed simulation systems, another was produced, called the Real-time Platform Reference Federation Object Model (RPR FOM). It extends HLA and the Institute of Electrical and Electronics Engineers' (IEEE) Standard for Distributed Interactive Simulation (DIS). DII COE specifies how simulations will use a common operating environment, whereas HLA and the RPR FOM superset specify how the simulations will communicate.

It is also useful to incorporate non-DoD standards into simulations. One such standard is the Common Object Request Broker Architecture (CORBA) that standardizes distributed objects so that simulations (as well as other computer applications) can interact in a heterogeneous environment (e.g., different platforms or programming languages) [3]. The next three parts of this section provide an overview of DII COE, HLA, CORBA, and RPR FOM, leading to an illustration of the relationship between these standards and the entire EXHORT framework.

2.1 The Defense Information Infrastructure Common Operating Environment (DII COE)

DII COE is a flexible approach to building interoperable systems that was developed and implemented by the Defense Information Systems Agency (DISA). DII COE includes a collection of reusable software components, a software infrastructure for supporting mission-area simulations, and a set of guidelines, standards, and specifications. The guidelines, standards, and specifications describe how to reuse existing software and properly build new software to ensure seamless and automated integration. The core concept of DII COE compliance is ensuring that simulations use as many standard conventions and modules as possible for maximum interoperability.

The heart of DII COE is the Integration and Runtime Specification (I&RTS), which describes the rules programmers need to follow to achieve DII COE compliance. Compliance is measured by the degree to which a simulation follows the I&RTS guidelines. There are four compliance categories: runtime environment, style guide, architectural compatibility, and software quality. Currently, DISA focuses on only the first category, runtime environment. The entry level at which DISA will evaluate mission applications for runtime environment compliance is Level 5. Acceptance as an official product requires demonstrated interoperable compliance (Level 7) and a migration strategy for attaining full DII COE compliance (Level 8).

2.2 High Level Architecture (HLA)

The DoD's Defense Modeling and Simulation Office (DMSO) is leading a DoD-wide effort to support reuse and interoperability throughout the DoD's wide spectrum of models and simulations [4]. HLA is a general-purpose architecture to encourage reuse and interoperability of simulations [5]. HLA brings together systems built for separate purposes, technologies from different eras, products from various vendors, and platforms from various services, enabling them to interoperate in a virtual environment [5].

HLA is based on the premise that no one simulation can satisfy all uses and users. From this premise comes an approach that links different types of simulations at multiple locations to create a realistic, complex, virtual world. HLA's intent is to support reuse of capabilities available in different simulations, ultimately reducing the cost and time required to create virtual environments for new purposes [5]. Both the Object Management Group (OMG), a large ad hoc organization composed of several companies, and the DoD has adopted HLA as a standard for future defense simulation development [5], and the IEEE is codifying HLA as three IEEE 1516 standards [6].

2.2.1 How HLA works

An individual simulation or set of simulations developed for one purpose can then meet the dynamic data needs of other simulations under a *federation* — the concept used in HLA for a group of interacting simulations. HLA defines rules and specifications governing how these simulations, or *federates*, interact with each other in a federation [5]. The federates structure data according to an Object Model Template (OMT) and communicate through a data distribution mechanism called the Runtime Infrastructure (RTI).

2.2.2 The Object Model Template (OMT)

The OMT provides a standard format for documenting the objects, attributes, and interactions that are exchanged during a federation execution [5]. The OMT promotes the reuse of single federates or a federation as a whole and can be viewed as a contract between federates regarding how a common federation is executed.

2.2.3 The Interface Specification

The Interface Specification specifies actions a simulation may perform, or be asked to perform, during an HLA federation execution [5]. The Interface Specification prescribes the interface between each federate and the RTI and provides communication and coordination services to the federates. Federation communication takes place only between each federate and the RTI, not among the federates themselves. The RTI is, in effect, a distributed operating system that is designed to provide the simulations with a standardized set of services that were previously handled by the individual simulation [5].

2.3 The Common Object Request Broker Architecture (CORBA)

CORBA was developed to address troublesome versioning and security problems in large-scale, worldwide client/server applications. Client/server computing is characterized by a two-tiered architecture having a client tier and a server tier. In a two-tiered model, the client (the requestor of information) consists of a graphical user interface (GUI) and program logic. The latter manipulates the raw data from the server (the repository or provider of information).

Consider a single deployment server with clients distributed among many locations around the world. All clients connect to the single server for raw data. A change to the client (e.g., new logic for staging area usage) requires that all copies of the client software be updated. Since changes to the client simulation logic can be frequent, the "fat client" approach makes it difficult to ensure that all client simulations are the latest version.

This two-tiered approach has the additional problem of encoding potentially sensitive, proprietary, or copyrighted information into the business logic, which is the part of the system that is distributed to clients. This information could constitute a company's competitive advantage over its competitors. Such a loss of control is a calculated risk that many organizations are not willing to take. In addition, there may be legal issues concerning the distribution of certain information, even in an encoded format.

The security and versioning problems can be avoided by using a three-tiered, or n-tiered approach that separates the program or business logic from the GUI. The program logic can then be maintained on a single server, under the control of the organization, while the part of the application that is actually distributed can remain "thin" (usually consisting primarily of the GUI). In a Web-based simulation using a "multi-tier" architecture, the client simulation communicates with the model-logic server, which, in turn, communicates with the data server. This is the approach taken in CORBA applications, as well as in other modern, distributed, "multi-tiered" commercial applications built using Java, particularly Java applications that conform to Sun's J2EE specification [7].

A new complexity, however, is introduced by this distributed object architecture, because objects residing in the GUI need to communicate with objects residing on the simulation server. But if the server and client both use the same class framework (e.g., EXHORT), this complexity is minimized and the communication is simplified.

2.3.1 How CORBA works

CORBA, developed by the OMG, standardizes communications among distributed objects. CORBA defines the structure and design of a system. It allows objects to communicate in a heterogeneous computing environment regardless of where the objects reside or how they are implemented. Only the public interface needs to be known. The public interface allows an object residing on system X to invoke a method of an object residing on system Y, where X and Y can be completely different systems or objects implemented in different programming languages. This interaction is possible because each object must have an interface defined according to the Interface Definition Language. This interface defines the operation, parameter types, user-defined exceptions for operation failure, and the context or environment. Interface Definition Language interfaces are registered with the Object Request Broker (ORB) and are stored in the interface repository, where they can be referenced when a request for an operation is received [4].

The ORB is responsible for delivering requests and replies between clients and servers. The ORB provides the communication infrastructure needed to deliver requests and associated parameters to the servers. It is responsible for making the connection to the server, sending the parameters for transfer across a network, and returning the operation result to the client. If the client knows which server object it will interact with, all references can be made through the Static Invocation Interface. In this case, the exact reference of the object and its location are known (e.g., Internet address) at the time the programmer is writing the code, and the code can be written to "*get X*

at Y." These requests are synchronous, meaning they wait until the server returns with a value (a positive return value — e.g., a document) or an exception (a negative return value — e.g., no such data available). The Dynamic Invocation Interface is available for requests that will not be known at compile time. Because the programmer does not know the location of some of the data or code at the time the code is written, the code "*get X at Y*" cannot be written. Instead, the programmer writes the generic "*get*" code and X and Y are left to be found at runtime. Essentially, the code contacts a search engine to find the values. This second strategy allows the client to (1) dynamically query the ORB for a set of objects that can fulfill a specific operation, (2) retrieve the interface of the selected objects, (3) construct the request, (4) invoke the request, and (5) receive the results at runtime [3].

2.4 EXHORT and the Real-Time Platform Reference Federation Object Model (RPR FOM)

RPR FOM was designed to support real-time simulations where the principal participants are discrete physical entities such as planes, ships, soldiers, and munitions [9]. At first glance, RPR FOM seems designed to address the same entities as EXHORT. However, several differences exist that make them incompatible and designed for different types of applications.

Real-time Platform Reference Federation Object Model (RPR FOM) is built on the Distributed Interactive Simulation (DIS) and implements the DIS protocol data units in the form of a hierarchy of classes and interactions. The primary mission of DIS is 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 [8]. Conversely, EXHORT's mission is to develop an infrastructure for linking transportation and infrastructure models to create realistic, complex, logistics systems for simulating logistics activities. Their difference is fundamental — RPR FOM is designed to be used in continuous, real-time, virtual environment simulations, whereas EXHORT is designed to be used in simulations in discrete-event, deliberative planning involving mathematical throughput and utilization studies of military transportation logistics systems. The different intended audiences give rise to differences in design.

EXHORT also addresses the issue of transportation entities differently than RPR FOM. RPR FOM's entities usually are at a higher level of detail, whereas the class abstractions in EXHORT are more aggregate, encapsulating only the characteristics required of deployments simulations. Specifically, soldiers and munitions are addressed aggregately in EXHORT, but RPR FOM simulations often address each soldier and munition. The TCH defines a *PAX* as one piece of cargo, even though it really is a collection of soldiers. The individual soldiers are only defined as part of an aggregate that must be processed and shipped. EXHORT simulations may model a 1-soldier unit or a 1-soldier leftover after the rest of his unit moved, but both of these cases are extremely unlikely. Even then, it is still represented as a *PAX* object, not as an individual soldier object. RPR FOM allows for the definition of individual soldiers as their own object when they cannot be addressed as part of a larger object [9], but EXHORT simulations do not require this level of detail.

EXHORT and RPR FOM address transportation entities differently because of the level of detail needed to run real-time simulations. Although some of the classes found in RPR FOM and EXHORT may seem structurally similar (e.g., *Aircraft/Aircraft*, *Physicalentity/Physical*, and *Groundvehicle/Vehicle*), the attributes of these classes lend themselves to very different outcomes. For example, RPR FOM's *Aircraft* class represents entities, such as airplanes or balloons, which operate mainly in the air [9]. The TCH *Aircraft* class encapsulates airborne vessels, such as transport airplanes or helicopters, which carry cargo. Moreover, the attributes of RPR FOM's *Aircraft* class provide a simulation with information for depicting how an airplane will look at any given moment during a simulation. This information is ultimately used to provide visible cues to a trainee in a simulator. RPR FOM's more detailed attributes are not germane to the needs of EXHORT because EXHORT is concerned with only the outcome of the simulation. TRANSCAP uses EXHORT to simulate installation transportation operations, compute time-phased outloading capabilities, compare computed capabilities to outloading requirements, identify system and infrastructure constraints, and installation-specific force departure profiles. All of these goals are outcome-oriented. Because of RPR FOM's differing mission, logical separation, differing approach in addressing transportation entities, and focus on real-time attributes, EXHORT does not use RPR FOM's extendible class framework.

2.5 Programming standards and EXHORT

EXHORT supports the efforts under DII COE to create a collection of reusable parts — some basic, some very complicated — so that programmers can work by assembling existing components rather than continually creating completely new ones. If the EXHORT hierarchies were built in a distributed architecture, a factory pattern in a business logic server with the EXHORT objects could qualify as one or two reusable segments under DII COE. In addition, a very detailed DII COE segmentation might identify reusable algorithm segments. An example better explaining these concepts follows in the next section.

For interactions among simulations, the EXHORT hierarchies would enhance the usefulness of HLA by adding a means of ensuring that the information exchanged through HLA mechanisms is meaningful and accurate. HLA compliance satisfies a condition for interoperability and reuse: a common, efficient technical means of joining simulations in a federation, optionally including live players, and exchanging information in a coherent manner [5]. However, HLA does not specify what constitutes an object. Thus, while HLA allows simulations to communicate, potential variations in object definitions mean that HLA compliance does not guarantee a valid, meaningful exchange of information throughout a federation. Adoption of the EXHORT hierarchies as a standard, consistent class attribute representation for transportation/deployment simulations would provide simpler and more accurate communication among federated simulations in our community.

For example, once the ELIST and TRANSCAP deployment simulations use TCH and ICH, TRANSCAP could track the sizes and quantities of cargo and railcars at a detailed level, making uses of groups of *Railcar* Objects [TCH] and *Siding* Objects [ICH], which would be contained by an *Interchange Yard* object [ICH]. Since the *Interchange Yard* object also has characteristics inher ited from the *Point of Interest* class, TRANSCAP could also place aggregated information into the *Interchange Yard* Object, making it an object that ELIST could use. In this way, ELIST and TRANSCAP can use the identical *Railcar* and *Point of Interest* objects. While HLA can manage this data exchange between TRANSCAP and ELIST, data translations may be required beyond those forced by simulations operating at different levels of aggregation. Combining HLA with EXHORT makes this translation step unnecessary.

Simulations using EXHORT may use a CORBA design strategy. That architecture is valuable because its requirements lead to designs that foster reuse. Under it, components are designed and built independently and then used dynamically as programs are running. For example, instantiations of the ICH or TCH objects could be stored in a CORBA server/object adapter, and thus centralized for use by several simulations. Such combinations of EXHORT and CORBA could ease interactions within HLA federations. In addition, CORBA-like interfaces can be built into subclasses of EXHORT to make it easier to use in developing standards-compliant deployment simulations.

2.6 Bringing All the Standards Together

The following example illustrates the interrelationships among DII COE, HLA, CORBA, RPR FOM, and ICH. Figure 1 illustrates the "ABCD Federation," where A, B, C, and D are cargo deployment simulations and are federates of the ABCD Federation. Simulation A and Simulation C both use EXHORT to represent some of their transportation (*Cargo* class) and infrastructure (*Interchange Yard* class) objects.

Simulations A and C use a CORBA server to hold, for example, all of the instantiations of the TCH *Cargo* class and ICH *Interchange Yard* classes. This ensures that every CORBA client using that server has access to the same definitions and instantiations of the *Cargo* and *Interchange Yard* classes. Without this global access, each simulation would have to set up its own instantiation of each class, introducing the possibility of mismatches between simulations and resulting errors. In addition, Simulations A and C are both using rail deployment segments that are DII COE-compliant, which ensures that they have been verified and tested.

During the execution of a simulation, all of the simulations communicate by means of HLA mechanisms, for example, the RTI. Suppose Simulation A provides information about a *Railcar* object that is needed by Simulation C. Simulation A publishes the railcar information in its public interface. Through the RTI, Simulation C finds out that the information is available. To retrieve the information, Simulation C must construct a *Railcar* object identical to that in Simulation A. If Simulation C used a "proprietary" definition of the *Railcar* class rather than the TCH *Railcar* class, the two simulations might use the same field name to refer to different properties of a railcar. Therefore, Simulation C might successfully retrieve a value for that field from Simulation A, but the value would be incorrect. By using the TCH, Simulation C can use the *Railcar* class to construct a faithful replica of the original object in Simulation A. The issue now becomes an internal one of filling the fields in the object, rather than an intermodel problem of translating between classes.

2.7 EXHORT and Java

The Java programming language was selected for implementing EXHORT because it provides needed standardization and promotes the use of Internet capabilities in the development of simulation and animation tools [7]. Java is an object-oriented programming language that provides the encapsulation and inheritance required for EXHORT. Java is platform-independent and supports network communications. It is robust and provides extensive error detection and handling.

2.8 Benefits of EXHORT

Using EXHORT provides several benefits, because it

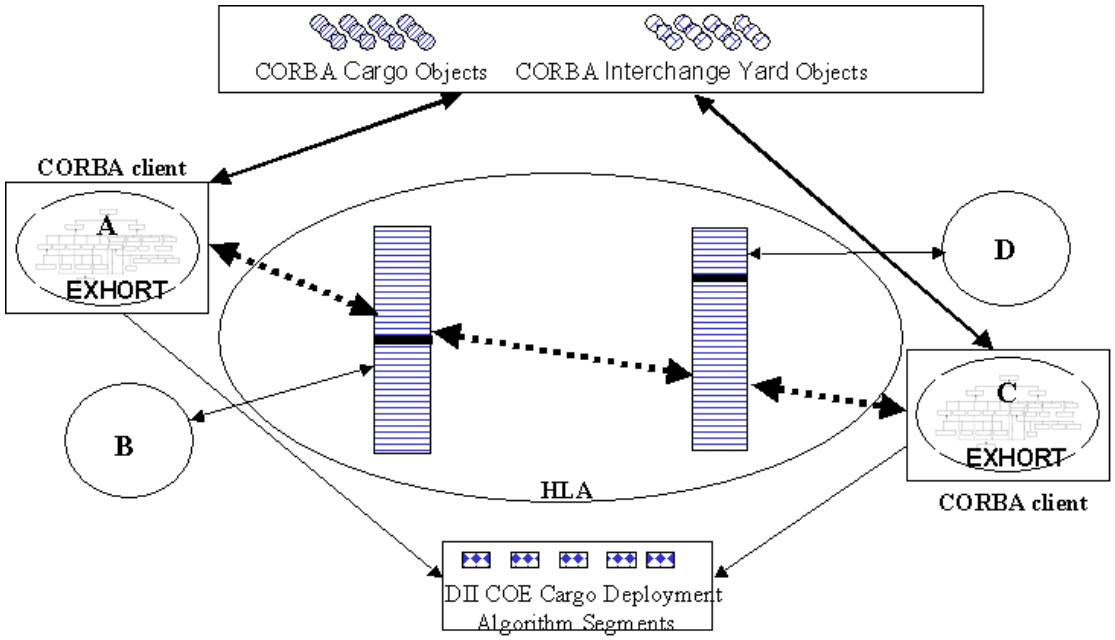

Figure 1. ABCD Federation

- Allows deployment simulations of varying levels of aggregation to use the identical data structures, ensuring exchanges of greater transparency.
- Reduces the effort needed to integrate applications and analyze a defense transportation system.
- Reuses code that is already tested and verified, allowing simulations to be developed more quickly.
- Ensures that simulations at different levels of aggregation can communicate and interoperate easily.
- Uses modern design strategies and accommodates programming standards.
- Provides simulations that implement the EXHORT system with the added flexibility of creating generalized methods or abstract interfaces that are only part of the ICH or TCH, or part of both, because TCH and ICH are divided logically while having a common root class.

3. The TCH

In designing the TCH hierarchy of the EXHORT framework, Argonne drew upon the experiences gained from designing three force projection simulations (ELIST, PORTSIM, and TRANSCAP). The TCH defines the different types of transport assets and military cargo upon which actions (e.g., loading or moving) are performed in a simulation. These assets and cargo are essentially the "clients" that are served at various inspection, loading, and waiting areas.

The TCH is comprised of 35 classes and one interface (as shown in Figure 2) in the Unified Modeling Language (UML). Continuous arrows imply an inheritance relationship. Classes are represented by rectangles. Interfaces are represented by rectangles with the stereotype <<interface>>. Dotted arrows imply a "realize relationship," of which the Java "implements" relationship is one. Abstract classes, which exist for generalization of common information between related classes, are written in italics. For more information, see [1].

The depth of the TCH hierarchy is due to the small differences in the treatment of cargo and transports during the various inspection, staging and loading processes, especially when simulated at different levels of aggregation. The TCH hierarchy has many levels of abstract classes, allowing for more generalization in the simulation specific sub-classes that inherit from the TCH.

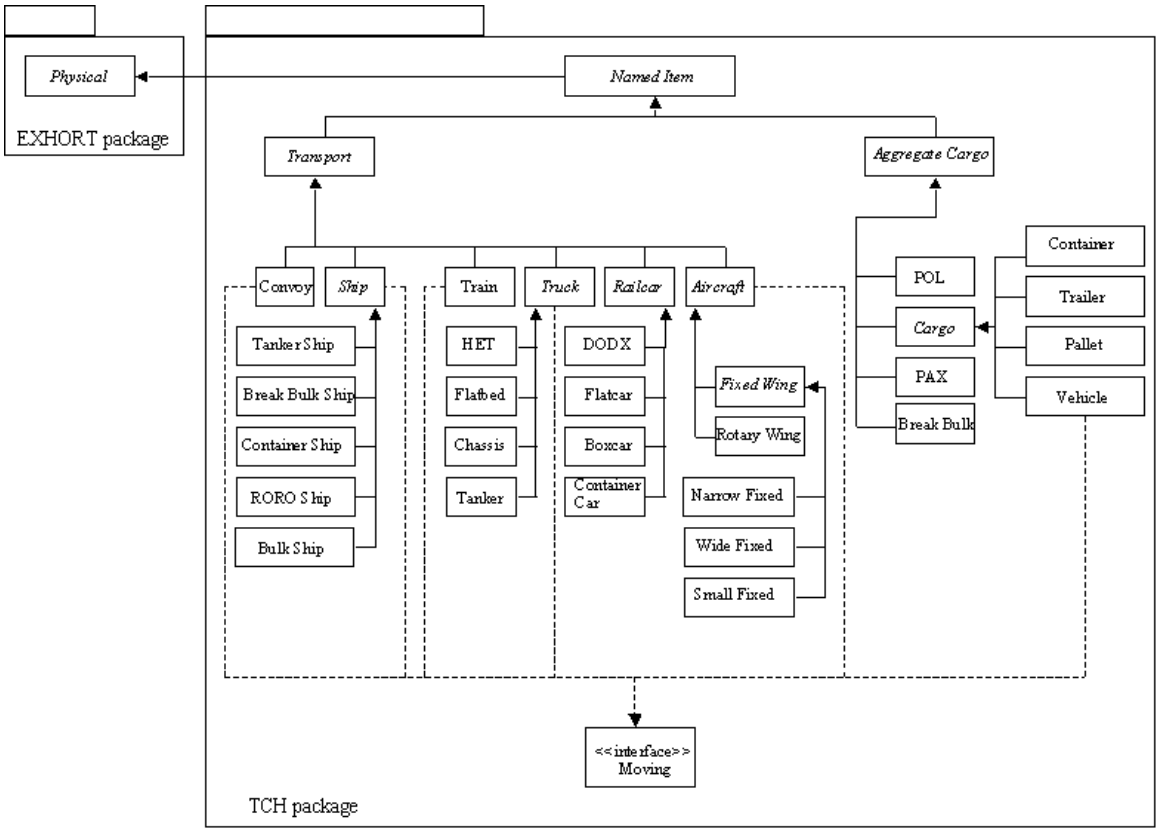

Figure 2 – Transportation Class Hierarchy

4. The ICH

The ICH is comprised of 40 classes, as shown in Figure 3, which is similar to Figure 2. The ICH is written in UML at an early analysis phase, for simplicity. To reduce complexity in reading the drawing for this overview article, the association links between the classes. Most of these associations are composition links that are described in this section. The ICH deals with locations where activities are performed and with the physical infrastructure, such as a motor pool at a fort or a berth at a port.

4.1 Relationship between the TCH and the ICH

It is difficult to define the key abstractions in one hierarchy without reference to the other hierarchy, since our definitions are restricted to the needs of transportation simulations. For example, a car is a transportation vehicle that moves across a road. Now, how could a road be defined? A road is an open public way used by cars — the infrastructure upon which cars move.

In terms of EXHORT, the TCH is a hierarchy of classes that encapsulates the transportation assets that "move across a road," or something that exists in nature (i.e., an object instantiating the *NamedItem* class or instantiating a class that inherits from *NamedItem*) that moves across infrastructure (e.g., a road). Conversely, the ICH is a hierarchy of classes that encapsulates "the infrastructure upon which cars move" or something that has an object from the TCH (i.e., an object representing a car, truck, railcar, locomotive, or container) moving across it.

CORBA-style interface methods are the "links" between the two class hierarchies in a class framework. These are the methods whereby the relationships between the TCH and ICH objects are established. Just as with a car (TCH) driving on a road (ICH), there also needs to be a *drives on* method. Because cars enter and exit a parking lot (ICH), there should also be *enters* and *exits* methods. So, all infrastructure needs *enter* and *exit* methods that accept a TCH object as an input argument. That TCH object is entering or exiting this infrastructure, so that the object is passed to the infrastructure object and the infrastructure object holds it and changes it, as needed (e.g., changing the infrastructure's location pointer from the previous piece of infrastructure to this current one, changing its state to, e.g., looking for a parking spot). Our current implementation of the TCH and ICH does not include these methods. Instead, the methods are left to be defined, with an actual simulation's need in mind, by the implementer of the subclasses that are actually instantiated in the simulation using EXHORT. More specifically,

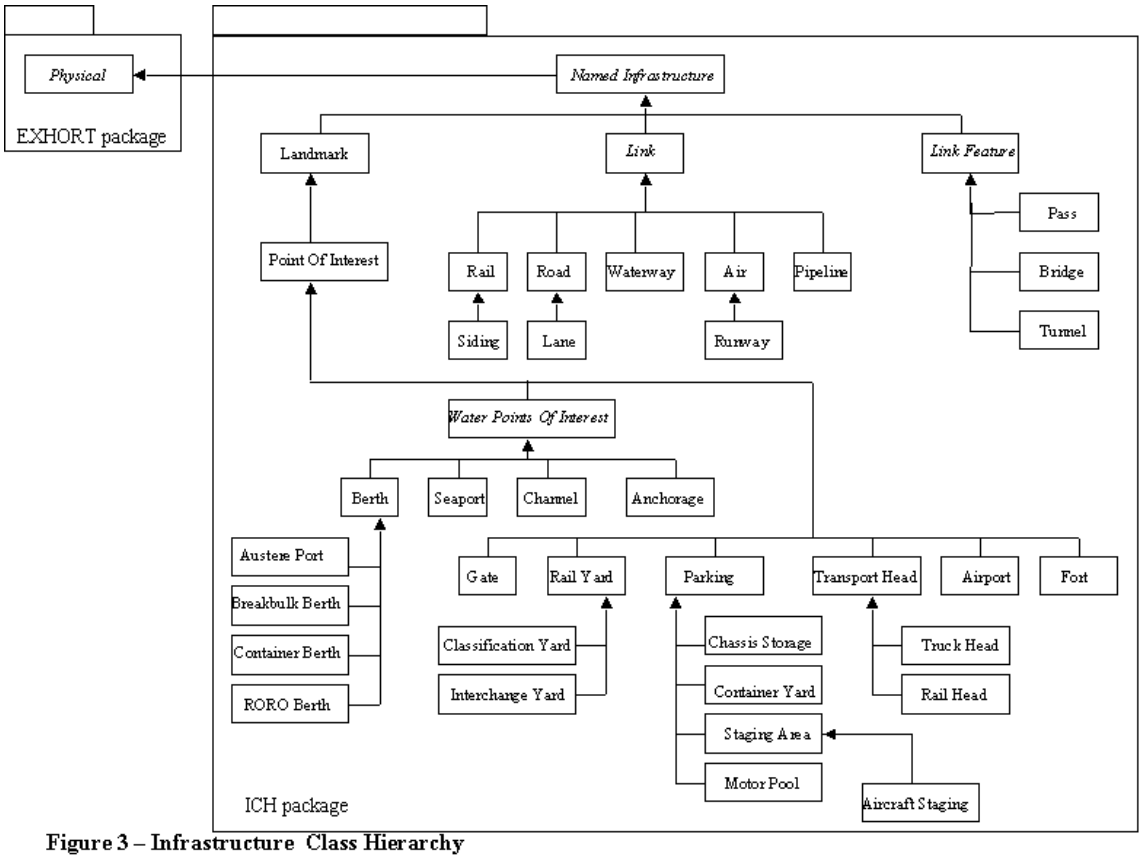

Figure 3 – Infrastructure Class Hierarchy

the listing of the fields common to deployment simulations is finished for the TCH and ICH, but the important interface methods that form the communication between the two hierarchies in EXHORT are best left to the implementer of an actual simulation. It is expected that the ICH will be extended for use in actual simulations. The hierarchy as described here does not necessarily include everything that is required in a real simulation. The hierarchy of classes has been detailed at a design level view, but the details of the classes are still left at an analysis level view. The ICH classes are small, generic, modular, self-sufficient classes, whereas the real simulation specifics will exist in its sub-classes.

4.2 ICH class descriptions

4.2.1 Named Infrastructure class

Named Infrastructure is an abstract class that encapsulates the measurements and location of infrastructure. *Named Infrastructure* describes the physical characteristics of any piece or place of infrastructure where or upon which deployment operations occur. *Named Infrastructure* inherits from its parent class, *Physical*.

4.2.2 Landmark class

Landmark is a concrete class that inherits from its parent, *Named Infrastructure*. A landmark is any piece or place of infrastructure where deployment operations may occur. *Landmark* describes a place of infrastructure where deployment operations occur.

4.2.3 Link class

Link is an abstract class encapsulating different types of routes along which TCH objects move during deployment operations. A link is contained within a landmark or connects two landmarks. *Link* inherits from its parent, *Named Infrastructure/Physical,* and describes the link's origin and destination landmarks, whether or not the link supports one-way movement and which link is preferred.

4.2.4 Link Feature class

Link Feature is an abstract class encapsulating a specific and notable piece of infrastructure on a *Link* between two landmarks that constrains movement (e.g., tunnel or bridge). *Link Feature* inherits from its parent, *Named Infrastructure,* and implements one field that references the link upon which this feature appears.

4.2.5 Point of Interest class

Point of Interest is a concrete class encapsulating a landmark modeled at a more detailed level. Landmarks would be modeled in an aggregate model, such as ELIST, by a few simple, often sequential processes. A *Point of Interest* would be modeled at a more detailed process level by simulations with more complex nonlinear systems.

These differences could involve the containership of "resource objects" that perform specific, detailed processes that a detailed simulation would track. These resources could be spelled out as a separate hierarchy or simply become fields and methods of *Point of Interest* subclasses.

Landmarks and points of interest are the same places. What differentiates them is user interest in a more detailed analysis of some areas. A landmark might be a structure (e.g., gate), an area of land (e.g., staging area), or an area of water (e.g., berth); that is, a landmark is any place where processing is simulated. *Point of Interest* inherits from its parent, *Landmark,* and describes the point of interest's physical characteristics: the storage capacity of equipment, POL, PAX, vehicles, trucks, containers, railcar, and truck transport, and whether or not lights are available for night deployment operations.

4.2.6 Rail, Road, Waterway, Air, and Pipeline classes

These are concrete classes that encapsulate ordered lists of segments connecting two landmarks that some appropriate TCH object can move across during a deployment operation. These classes inherit from their parent, *Link*. The assumed subclasses of each of these classes, which are not part of EXHORT, will contain fields and methods describing the unique aspects of these types of links germane to the particular simulation in which this class will be used.

4.2.7 Pass, Bridge, and Tunnel classes

These classes are concrete classes that encapsulate specific obstacles on a link. They are link features that a deployment simulation might employ to affect throughput. They inherit from their parent, *Link Feature*. The assumed subclasses will contain fields and methods describing the unique aspects of tunnels germane to the particular simulation in which this class will be used.

4.2.8 Siding, Lane, and Runway classes

These classes are concrete classes that encapsulate ordered lists of segments that are contained inside a *Landmark* object. These classes are the locations where TCH objects can be parked, loaded, or otherwise have work performed on them in a deployment operation.

4.2.9 Water Point of Interest class

Water Point of Interest is an abstract class that encapsulates a point of interest that has attributes shared by all subclasses that involve water operations. *Water Point of Interest* inherits from its parent, *Point of Interest*.

4.2.10 Berth, Seaport, Channel, and Anchorage classes

These are concrete classes that encapsulate water-oriented landmarks where cargo and/or transport ships (both TCH objects) have services performed on them, with the goal being that the various types of cargo objects in the TCH will be loaded onto or loaded off of S*hip* objects. They inherit from *Water Point of Interest*. A *Seaport* object may contain *Berth*, *Seaport*, and *Channel* objects (among other, non-water-oriented landmarks), depending on the level of aggregation of the simulation.

4.2.11 Fort, Airport, Transport Head, Parking, Rail Yard, and Gate classes

These are concrete classes that encapsulate landmarks where cargo and transports (both TCH objects) have services performed on them, towards various goals. While the *Gate* class is an interchange point between a landmark and a *Road* object, the other classes listed have a principle goal of loading/unloading various types of cargo objects in the TCH onto/off the different types of *Railcar* and *Truck Transport* objects. They inherit from *Point of Interest*. *Fort* and *Airport* objects may contain *Transport Head*, *Parking*, *Rail Yard*, and *Gate* objects (among other, non-loading landmarks), depending on the level of aggregation of the simulation.

4.2.12 Austere Port, Break Bulk Berth, RORO Berth, and Container Berth classes

These are concrete classes that encapsulate specific berth landmarks at a port that loads cargo from or onto cargo ships. They inherit from their parent, *Berth*. An *Austere Port* class is not intended to be the container class of an entire austere port operation. A *Seaport* class should still be used. Instead, it simply describes the roll-on/roll-off spots for the cargo, elevated piers with a crane for loading and unloading of lighters, and floating piers with a crane that load and unload lighters.

4.2.13 Classification Yard and Interchange Yard classes

These are concrete classes that encapsulate landmarks at a fort, port, or airport. They are places where *Railcar* objects can be inspection, stored, and sorted. They inherit from *Rail Yard*. The simulation-specific subclasses will contain fields and methods describing the unique aspects

germane to the particular simulation in which this class will be used.

4.2.14 Truck Head and Rail Head classes

These are concrete classes that encapsulate landmarks at a fort, port, or airport that serve as places where cargo is unloaded or loaded from or onto transports. They inherit from *Transport Head*. The assumed subclasses will contain fields and methods describing the unique aspects of transport heads germane to the particular simulation in which this class will be used

4.2.15 Chassis Storage, Container Yard, Staging Area, and Motor Pool classes

These are concrete classes that encapsulate landmarks at a fort, port, or airport where different levels of staging and inspection-oriented processing are done on different types of cargo in advance of loading and unloading at other landmarks. They inherit from *Parking*. The simulation-specific subclasses will contain fields and methods describing the unique aspects germane to the particular simulation in which this class will be used.

4.2.16 Aircraft Staging class

Aircraft Staging is a concrete class that encapsulates a landmark at a fort, port, or airport. It is an inspection or storage area where groups of airplanes leave in small groups or convoys headed for another landmark at an airport. *Aircraft Staging* inherits from its parent, *Staging Area*, and implements four fields and eight methods that describe whether there is space available for maximum-on-ground wide, narrow, and small-body planes.

5. Summary and Future Directions

With the current war on terrorism, the U.S. military appears to be entering a new era in which it will be called upon to make even more war-fighting deployments while simultaneously continuing to perform nontraditional missions, including disaster relief and peacekeeping. Modeling and simulation can help the DoD plan for, improve upon, and reduce the costs of many deployments. However, efficient and effective simulation of deployment logistics depends on "interoperability" — the accurate and efficient exchange of data among the many and varied simulations available.

To promote interoperability, the defense community is encouraging, and in some cases requiring, software programmers to employ certain standard practices and architectures. Three major standards are the Defense Information Infrastructure Common Operating Environment (DII COE), the High Level Architecture (HLA), and the Common Object Request Broker Architecture (CORBA). Each has a slightly different role in standardizing the modeling environment. DII COE is focused on verifying levels of standardization in the runtime environment. HLA provides conventions for streamlining data sharing among a specified group of simulations (a *federation*). CORBA provides conventions for setting up a "clearing-house" function to manage distributed objects used by multiple simulations. To augment this set of standards, Argonne National Laboratory has designed EXHORT, a framework of two hierarchies that together will constitute a standard and consistent class attribute representation and behavior applicable to transportation deployment simulations. EXHORT describes major portions of the defense transportation system, providing a standardized code structure for object-oriented deployment simulations that can help ensure meaningful data exchanges.

EXHORT reduces the need for translation among class representations, thereby reducing the risk of introducing errors when different simulations communicate. It uses modern design strategies and accommodates programming standards. EXHORT is an efficient and reliable development method and could be a candidate as a reusable component in the DII COE. It could improve the accuracy of exchanges among simulations in an HLA federation. EXHORT may also be used with CORBA to make exchanges more accurate among simulations that share a distributed object architecture. EXHORT's first hierarchy, the TCH, has been accepted as an Army Deployment/Redeployment community standard (SRD 00068). The ICH is currently a candidate to the same body. EXHORT allows deployment simulations to use the same set of underlying class data, ensures transparent exchanges, reduces the effort needed to integrate simulations, and permits a detailed analysis of the defense transportation system.

In the future, Argonne would like to document further work that we have done in building a third hierarchy, the Resource Class Hierarchy. This hierarchy separates process-performing components from the physical entities in the ICH. This achieves a smoother, more logical separation that would reduce some of the coupling between the classes at no loss of cohesion.

6. Acknowledgements

This work was supported under a military interdepartmental purchase request from the U.S. Department of Defense, Military Traffic Management Command Transportation Engineering Agency (MTMCTEA), through the U.S. Department of Energy contract W-31-109-ENG-109. The authors acknowledge the support of our program manager, Melvin Sutton, of MTMCTEA.

7. References

[1] Burke, J. F., Macal, C. M., Nevins, M. R., VanGroningen, C. N., Howard, D. L., and Jackson, J., "Standardization of Transportation Classes for Object-Oriented Deployment Simulations," 99F-SIW-193 in Simulation Interoperability Standards Organization 1999 Fall Simulation Interoperability Workshop, Volume III, pp. 1142-1152, 1999.

[2] SRD 00068 at AMSO's Army Standard Repository, at http://www.msrr.army.mil/astars.

[3] Object Management Group, "CORBA Basics," at http://www.omg.org/gettingstarted/corbafaq.htm, May 2002, accessed June 2002.

[4] DMSO, "High Level Architecture," at https://www.dmso.mil/public/transition/hla/, Defense Modeling and Simulation Office, U.S. Department of Defense, February 2002, accessed June 2002.

[5] AMSO, "HLA Overview Briefing," at http://www.amso.army.mil/topic/hla/overview.ppt, September 1999, accessed June 2002.

[6] SISO, "HLA Standards Development," at http://www.sisostds.org/stdsdev/hla, accessed June 2002.

[7] Sun Microsystems, "Java Developer Connection: Java 2 Platform, Enterprise Edition: Overview," at http://java.sun.com/j2ee/overview.html, accessed June 2002.

[8] DIS Steering Committee, "The DIS Vision: A Map to the Future of Distributed Simulation," Version 1, Institute for Simulations & Training, Orlando, FL, 1994.

[9] SISO, "Guidance, Rationale, and Interoperability Modalities for the Real-time Platform Reference Federation Object Model (RPR FOM)," Sean Reilly and Keith Briggs (eds), draft 1.0v2, 10 September 1999.

Author Biographies

JAMES F. BURKE, JR., is a software engineer in the Modeling, Simulation, and Visualization Section of the Decision and Information Sciences Division of Argonne National Laboratory. He is the project leader and lead designer of the TRANSCAP model, which is a part of the Logistics Modeling and Simulation Program. He is working on a Ph.D. at the Illinois Institute of Technology, studying simulation component reuse and standardization. He received an M.S. in computer science from the Illinois Institute of Technology and a B.S. in computer science from Benedictine University. His research interests include reuse and standardization of object-oriented simulations, simulation modeling, and object-oriented analysis and design. He is a member of IEEE and ACM.

CHARLES N. VAN GRONINGEN leads the development team for the ELIST model at Argonne National Laboratory. He received his Ph.D. in artificial intelligence from the Illinois Institute of Technology in 1993; his M.S. in computer science is from DePaul University and his B.S. in math is from Trinity Christian College. His research interests include knowledge representation, modeling, and simulation. He is a member of the Military Operations Research Society and the American Association for Artificial Intelligence.

CHARLES M. MACAL directs the Simulation and Visualization Section and leads the Logistics Modeling and Simulation Program at Argonne National Laboratory. His research interests include simulation modeling and architectures and agent-based modeling. He received a Ph.D. in operations research from Northwestern University, as well as degrees from Purdue University. He is a registered professional engineer in Illinois and a member of INFORMS, the American Association for Artificial Intelligence, the Society for Computer Simulation, and Tau Beta Pi.

MARK J. BRAGEN is a software engineer in the Modeling, Simulation and Visualization Group of the Decision and Information Sciences Division of Argonne National Laboratory. He leads the development of the PORTSIM model. He received a BA and MS in Computer Science from the Illinois Institute of Technology. His research interests include simulation modeling and relational databases. He is a member of IEEE and ACM.