Standardization of Transportation Classes for Object-Oriented Deployment Simulations

James F. Burke, Jr.
Charles M. Macal
Michael R. Nevins
Charles N. Van Groningen
Dawn L. Howard
Jill Jackson
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
simulation
transportation

ABSTRACT: *Many recent efforts to integrate transportation and deployment simulations, although beneficial, have lacked a feature vital for seamless integration: a common data class representation. It is an objective of the Department of Defense (DoD) 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 collection of three hierarchies that together will 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 is the Transportation Class Hierarchy (TCH), which describes a significant portion of the defense transportation system; the other two deal with infrastructure and resource classes. EXHORT will allow deployment simulations to use the same set of underlying class data, ensure transparent exchanges, reduce the effort needed to integrate simulations, and permit a detailed analysis of the defense transportation system. This paper describes EXHORT's first hierarchy, the TCH, and provides a rationale for why it is a helpful tool for modeling major portions of the defense transportation system.*

1. Introduction

1.1 Background and Purpose

Logistics and mobility have become increasingly important because of the rapidly changing nature of the world. There is a critical need to move more people and supplies with fewer resources. With the pullback of U.S. military forces to the continental United States (CONUS), contingency planning for deploying forces to overseas locations is now of central importance. The U.S. Department of Defense (DoD) also has new missions, including disaster relief, both in the United States and overseas, and peacekeeping. Over the past 5 to 10 years, the DoD has made greater use of simulations to help meet its mission objectives [1]. These tools are used to analyze, plan, train for, and execute deployments. Such needs have motivated investment in new simulation models and technologies. It is more efficient and effective to simulate deployments first on a computer than to test them in the real world.

The Military Traffic Management Command Transportation Engineering Agency (MTMCTEA) and others are

DISCLAIMER

developing simulations to analyze military deployments. In conducting analyses of the defense transportation system, deployment models and simulation systems rely heavily on data regarding military cargo, transportation assets, and infrastructure. Currently, 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.

Simulations need to be reused to the maximum possible extent, and new simulations should be designed only when existing simulations cannot effectively provide DoD with the needed capability [1]. To realize the greatest return on investment, DoD must team these simulations together in different combinations to satisfy a diverse and ever-evolving set of user needs. Using and adhering to standards, accepted measures that define or represent and are used to establish norms, can meet these goals.

Many recent efforts to integrate transportation and deployment simulations, although beneficial, have lacked a feature vital for seamless integration: a common data class representation. It is an objective of the DoD 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 is a collection of three hierarchies that together will 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 take a bottom-up rather than a top-down approach. The class abstractions in EXHORT are specified at a detailed level that encapsulates the characteristics of deployment simulations, rather than at the higher, more generic level that is used more frequently in object modeling. This bottom-up approach is well suited for DoD's rapid prototyping environment.

These three hierarchies will allow many deployment simulations to use the same set of underlying class data and significantly reduce the effort needed to integrate simulations and to analyze the defense transportation system. The first hierarchy is the Transportation Class Hierarchy (TCH), which covers commercial transportation assets and military cargo (Section 3). 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. The third hierarchy is the Resource Class Hierarchy (RCH), which describes the resources needed to support deployment (e.g., ramps, rough terrain container handlers [RTCH], cranes, and forklifts). All three hierarchies are needed to describe major portions of the defense transportation system. The goal is to use EXHORT to 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 MTMCTEA, has developed three force projection simulation systems (ELIST — Enhanced Logistics Intra-theater Support Tool, PORTSIM — Port Simulation Model, and TRANSCAP — Transportation System Capability Model) and is beginning work on 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 movement from ports across the land transportation system through intermediate locations to the final destinations. ELIST also predicts whether the infrastructure and transportation assets can support the warfighting 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, computes time-phased outloading capability, and compares computed capability to outloading requirements. It will also identify system and infrastructure constraints and installation-specific force clearance profiles.

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

1.3 Content and Organization

This paper describes EXHORT's first hierarchy, the TCH, and provides a rationale for why it is a helpful tool for modeling major portions of the defense transportation system.

Section 2 introduces programming standards and their relation to the TCH. Section 3 provides specific informa-

tion about the classes of the TCH. Finally, Section 4 discusses the benefits and future directions of EXHORT.

2. Programming Standards

Programmers use standards as a guide to designing "standard" 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. Two of these are the Defense Information Infrastructure Common Operating Environment (DII COE) and the High Level Architecture (HLA) [2]. DII COE and HLA are completely separate and parallel undertakings. DII COE sets the standard for how simulations will use a common operating environment, whereas HLA standardizes how the simulations will communicate. Aside from the DoD standards, it is also useful to incorporate non-DoD standards into simulations to ensure greater standardization. One such standard is the Common Object Request Broker Architecture (CORBA) which standardizes distributed objects so that simulations can interact in a heterogeneous environment (e.g., different platforms or programming languages) [3,4]. The next three parts of this section provide an overview of DII COE, HLA, and CORBA, and the final part illustrates the relationship among these standards and the TCH.

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

2.1.1 What is the Purpose of DII COE?

DII COE is a flexible approach for building interoperable systems. It was developed and implemented by the Defense Information Systems Agency (DISA). DII COE is comprised of 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 how to build new software properly 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.

2.1.2 How Does DII COE Work?

The heart of DII COE is the Integration and Runtime Specification (I&RTS), which describes the steps 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 only on the first category, runtime environment. There are eight levels of runtime environment compliance, with a minimum compliance of 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)

2.2.1 What Is the Purpose of 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 — 451 simulations, each with unique user-defined needs [5]. HLA is a general-purpose architecture to encourage reuse and interoperability of simulations [2,5]. 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 [2].

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. The intent of HLA is to support reuse of capabilities available in different simulations, ultimately reducing the cost and time required creating virtual environments for new purposes.

The DoD has adopted HLA as a standard for future defense simulation development [6]. Even though HLA originated from the military, the architecture is also suited for civilian simulations [7]. The Institute of Electrical and Electronics Engineers (IEEE) is now in the process of codifying HLA in three IEEE 1516 standards [1].

2.2.2 How Does HLA Work?

An individual simulation or set of simulations developed for one purpose can then be applied to 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 [8]. The federates structure data according to an Object Model Template (OMT) and communicate through a data distribution mechanism called the Runtime Infrastructure (RTI).

2.2.3 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 [6,7]. The OMT promotes the reuse of single federates or a federation as a whole. The OMT can be viewed as a contract between federates on how a common federation is executed.

2.2.4 The Interface Specification

The Interface Specification is a precise specification of actions a simulation may perform, or be asked to perform, during an HLA federation execution [6,7]. 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; designed to provide the simulations with a standardized set of services that were previously handled by the individual simulation [9].

2.3 The Common Object Request Broker Architecture (CORBA)

2.3.1 What Is the Purpose of CORBA?

CORBA was developed to address a troublesome versioning problem in large-scale, worldwide client/server applications. Client/server computing is characterized by a two-tiered architecture, with 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 that manipulates the data from the server (the repository or provider of information). The server simply provides raw data requested by the client, and the client is responsible for processing and displaying the raw data. In the two-tiered approach, the client is referred to as "fat" because the program logic to process the raw data and the GUI code to display the raw data are physically located in the client software — thus making the code large in size, i.e., "fat."

Consider a single deployment server with the client distributed to many locations around the world. All clients connect to the single server for raw data. When the raw data changes (e.g., a new type of material handling equipment is introduced), the distributed copies of the simulation software are not affected. However, 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 presents a versioning problem: it is difficult to ensure that all client simulations are the latest version.

This versioning problem can be avoided by using a three-tiered approach that separates the simulation logic from the GUI. The simulation logic can then be maintained on a single server, and the clients are "thin" (consisting primarily of the GUI). The client simulation communicates with the model-logic server, which, in turn, communicates with the data server. This is the approach taken by CORBA. A new complexity, however, is introduced by this distributed object architecture. 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.

2.3.2 How Does CORBA Work?

CORBA, developed by the Object Management Group (a large, ad hoc organization of different companies), 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 an object resides or how it is 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. The 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 [3,4].

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 — e.g., a document) or an exception (a negative return — 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 code "*get*" 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 object, (3) construct the request, (4) invoke the request, and (5) receive the results at runtime [3,4].

2.4 Programming Standards and the TCH

The TCH supports the efforts under DII COE to create a collection of reusable parts, some very basic, some complicated, so that programmers can work by assembling existing components rather than creating completely new ones. As a faster, more reliable code development system, the TCH could qualify as a reusable component under DII COE. The other hierarchies to be developed under EXHORT would also be candidate components, as would the EXHORT framework itself.

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

For example, once the ELIST and TRANSCAP deployment simulations use the TCH, the Railcar class will be the same in both. In an integrated scenario execution under HLA, a railcar object will be loaded by TRANSCAP and transferred (without any alterations) to ELIST for processing through the rail network. Similarly, the object used in TRANSCAP to represent a vehicle loaded on a railcar will be identical to the object used in ELIST to represent the vehicle that is unloaded from the railcar once it reaches its destination. While HLA can manage this data exchange between TRANSCAP and ELIST, data translations may be required. Combining HLA with the TCH makes this translation step unnecessary.

This verification of meaningful exchanges can also be accomplished through CORBA, but the TCH provides a more direct method. CORBA allows components from different simulations to interact with the same data structures (objects) by means of an intermediary, the "object broker." Roughly speaking, CORBA is a distributed operating system. In contrast, components that use TCH objects will "know" how to interact with TCH objects in other components without an intermediary.

Simulations using the TCH may also use CORBA strategies. 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. CORBA-like interfaces are being built into the TCH to make it easier to use in developing standards-compliant deployment simulations. For example, instantiations of the TCH objects could be stored in a CORBA server/object adapter and thus centralized for use by several simulations. Such combinations of TCH and CORBA could ease interactions within HLA federations.

The following example illustrates the interrelationships among DII COE, HLA, CORBA, and the TCH. 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 are both using the TCH to represent their transportation classes, one of which is the Cargo class.

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

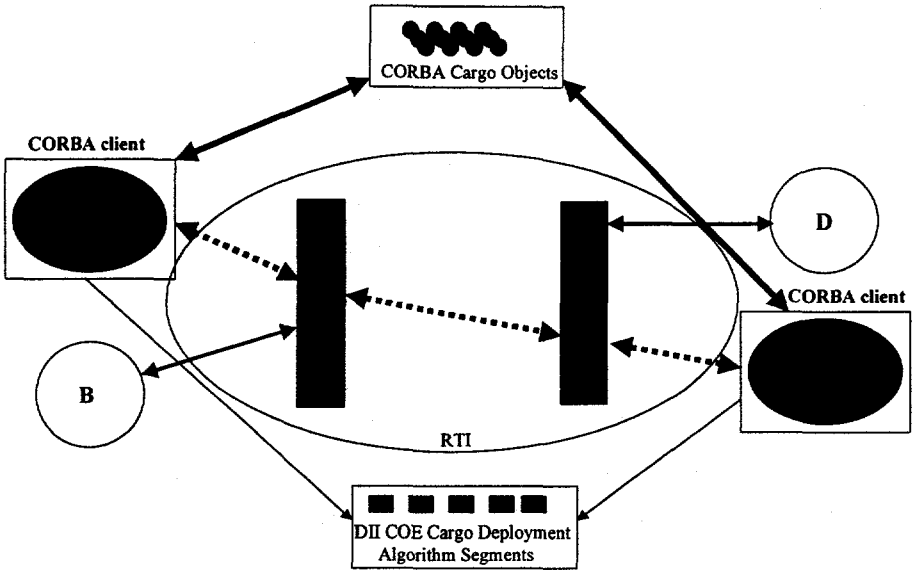

Figure 1. ABCD Federation

During the execution of a simulation, all the simulations communicate by the mechanisms of HLA, for example, the RTI. Suppose Simulation A provides information about a railcar object, and Simulation C needs to use that information. 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.

3. The TCH

In designing the TCH, 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. EXHORT, which is the entire framework, proposes a standardized code structure for object-oriented deployment simulations. EXHORT will standardize the way deployment simulations interoperate by allowing them to speak the same language, define terms the same way, and break data up in the same fashion.

3.1 TCH and JAVA

The Java programming language was selected for implementing the TCH. Java provides the needed standardization and at the same time 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 the TCH. Java is platform-independent and provides support for network communications. It is robust and provides extensive error detection and handling. Java's syntax is similar to the C and C++ languages, but its programmers have removed several C-language constructs that tended to cause problems in programming and replaced them with constructs that make Java programs more robust.

3.2 Benefits of the TCH

Using the TCH provides several benefits, because it

- Allows deployment simulations to use the same set of underlying class data, ensuring transparent exchanges.

- Reduces the effort needed to integrate applications and to analyze a defense transportation system.
- Reuses code that is already tested and verified, allowing simulations to be developed more quickly.
- Ensures that simulations can communicate and interoperate easily.
- Uses modern design strategies and accommodates programming standards.

3.3 TCH Description

The TCH is comprised of 24 classes and one interface, as shown in Figure 2 in the Unified Modeling Language. Continuous arrows imply an inheritance relationship. Classes are represented by rectangles. Interfaces are represented by rectangles with <<Interface>> written on the first line. Dotted arrows imply a "realize relationship," of which the Java "implements" relationship is one. The keyword "implements" is included to specify which type of realization relationship a set of arrows represents. Abstract classes are written in Italics.

While the listing of the fields common to deployment simulations is finished for the TCH, as well as their *set* and *get* methods (one to obtain and one to change the value of a field), the important interface methods in the TCH that form the communication among the three hierarchies in EXHORT are still being designed as the ICH and RCH are developed. For example, some of the fields of the Physical class are *height, length, width,* and *weight.* The class must have methods named *setHeight, setLength, setWidth,* and *setWeight,* as well as *getHeight, getLength, getWidth,* and *getWeight,* to allow the simulation to communicate with the object.

It is expected that the TCH will be expanded for use in actual simulations. The hierarchy as described here does not necessarily include everything that is required in a real simulation. The TCH resembles a CORBA design strategy because it possesses small, generic, modular, self-sufficient classes, whereas the real simulation specifics will exist in other classes. Gaps may exist in the TCH because the other two hierarchies of EXHORT have not yet been developed. EXHORT will continue evolving as work on the ICH and RCH identifies needs for additional data fields.

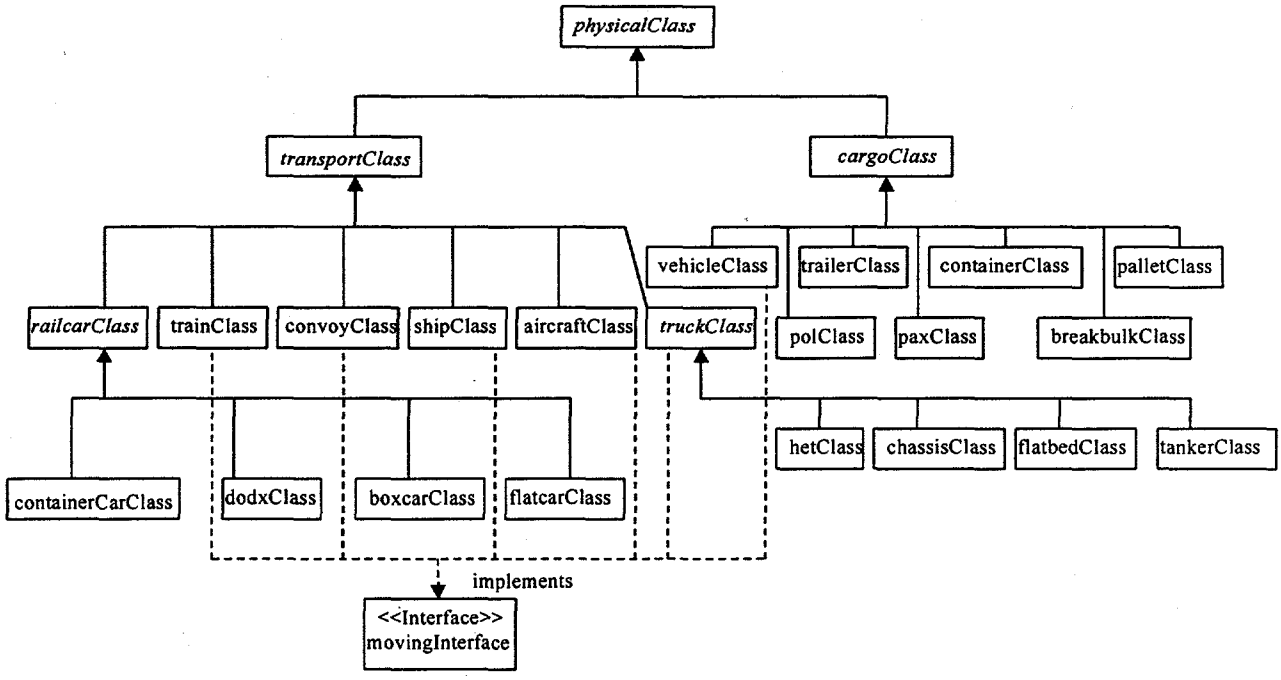

Figure 2. Transportation Class Hierarchy

3.3.1 Moving Interface

An interface is a device or system that allows unrelated entities to interact. Similarly, a Java interface allows unrelated objects to interact. A Java interface defines a set of methods but does not implement them. A class that implements an interface agrees to implement all methods defined by the interface, thereby agreeing to a certain behavior [10].

The Moving interface provides a mechanism for objects within the TCH to include capabilities for moving from one location to another. It will list methods describing interactions with resources and locations generic to moving things. The six classes that represent a prime mover (anything that can move by itself — Train, Convoy, Ship, Aircraft, Truck and Vehicle classes) must implement this interface.

3.3.2 Physical Class

Physical is an abstract class containing the physical dimensions, naming conventions, and other information that simulations require. It is generic to every asset acted upon at a node. A node is an origin or destination such as a fort, a port, an intermediate staging base, a tactical assembly area, or a beach.

3.3.3 Cargo Class

Cargo is an abstract class that encapsulates cargo. Cargo is anything that needs to be shipped (e.g., vehicle, container, or passengers [PAX]). Some cargo can move under its own power (e.g., a vehicle); other cargo cannot (e.g., a container). At this level of the hierarchy, this information is currently unknown and therefore is left to the child classes to determine. Later this class will specify how cargo calls a transport (e.g., heavy equipment transport) for pickup and how the transport will pick up the cargo (how they connect) by means of interactions with resources (e.g., RTCHs and forklifts) at a node. Cargo inherits from its parent class, Physical.

3.3.4 Transport Class

Transport is an abstract class encapsulating entities (e.g., trucks, trains, and aircraft) that transport cargo (e.g., vehicles, containers, trailers, or PAX). Once the TCH is integrated with the two other EXHORT hierarchies (ICH and RCH), the TCH will contain processes of interaction with resources that are generic to all types of transports.

3.3.5 Vehicle Class

Vehicle is a concrete class encapsulating military vehicles, at the bumper-number level of detail. Examples include tanks or high-utility motor vehicles that are being shipped from a continental United States home fort to a tactical assembly area outside the continental United States. Vehicle inherits from Cargo, because it is a deploying piece of cargo. Since a vehicle is a prime mover, this class will implement future methods that describe how the moving piece interacts with servers (e.g., inspectors, drivers, or end ramps) and must be implemented by means of the Moving interface.

3.3.6 Container, Pallet, Trailer, and Breakbulk Classes

The Container, Pallet, Trailer, and Breakbulk concrete classes inherit from their parent Cargo. Containers, pallets, trailers, and breakbulk are cargo that have physical dimensions and a name (inherited from Physical) and must be processed and shipped (inherited from Cargo). They are not prime movers because they cannot move themselves and require a vehicle or transport asset (e.g., a commercial chassis or a container car) to move them around.

3.3.7 PAX Class

PAX is a concrete class that encapsulates the military personnel who will be united with their equipment (e.g., cargo) in the host country. In EXHORT, PAX are considered as pieces of cargo that have physical dimension and a name (inherited from Physical) and must be processed and shipped (inherited from Cargo). PAX are not prime movers because they cannot move themselves and require a vehicle or a transportation asset (e.g., an aircraft) to reach a destination.

3.3.8 Petroleum, Oils, and Lubricants (POL) Class

Petroleum, Oils, and Lubricants (POL) is a concrete class that inherits from its parent Cargo. POL is cargo that has physical dimensions and a name (inherited from Physical) and must be processed and shipped (inherited from Cargo). POL is not a prime mover because it cannot move itself and requires a tanker truck to haul it around.

3.3.9 Convoy Class

Convoy is a concrete class that inherits from its parent Transport. Convoy encapsulates a group of road-capable military vehicles, prime movers, and towing trailers. Convoys move from node to node under their own power without assistance from transport assets. Because Convoy is a collection of prime movers (towing trailers), it implements the Moving interface.

3.3.10 Railcar Class

Railcar is an abstract class that will contain the fields and methods common to all railcars, such as how they interact with locomotives or how they are placed on the tracks of interchange and classification yards. Railcar inherits from Transport because it moves cargo in groups. It does not implement the Moving interface because it moves as part of a train.

3.3.11 Flatcar, Container Car, Boxcar, and DODX Classes

Flatcar, Container Car, Boxcar, and DODX (a DoD railcar) are concrete classes encapsulating types of railcars either rented from commercial carriers or owned by the DoD. A flatcar is a flat payload that moves lighter equipment. A container car is designed for moving containers. A boxcar moves mostly pallets of cargo. A DODX moves heavier equipment. These four classes inherit from Railcar and will contain fields and methods describing the different ways such railcars interact with the different types of resources (e.g., a flatcar with an end ramp or a container car with an RTCH).

3.3.12 Truck Class

Truck is an abstract class that inherits from Transport and will contain the fields and methods common to all commercial truck transport assets, such as how they interact with a resource such as a lane or a gate. Truck implements two methods regarding the speed of the transport object. Since trucks are prime movers, Truck implements the methods of the Moving interface.

3.3.13 HET, Chassis, Flatbed, and Tanker Classes

HET, Chassis, Flatbed, and Tanker are concrete classes encapsulating commercial truck transport assets, usually rented from commercial carriers. A HET (heavy equipment transport) is a tractor/cabin attached to a payload equipped for moving heavy equipment. A chassis is a tractor/cabin with an I-beam to which a container is attached. A flatbed is a tractor/cabin with a flatbed payload designed for lighter equipment. A tanker is a tractor/cabin attached to a liquid storage container used for transporting POLs. These four classes inherit from Truck and will contain the fields and methods describing the different ways they interact with the different types of resources (e.g., a flatbed with an end ramp or a chassis with an RTCH).

3.3.14 Train Class

Train is a concrete class that inherits from its parent Transport and will contain methods specifying how it moves a group of cargo. A train is a string of DODX, general purpose flatcar, bi-level, or container cars that is loaded and unloaded with cargo at different nodes during deployment. Since a train is a prime mover, having a locomotive and a caboose that move the whole train under its own power, Train will implement methods describing how a moving piece interacts with servers (e.g., a train interacting with an RTCH while loading a container or a ramp while loading vehicles) and must be implemented by means of the Moving interface.

3.3.15 Ship Class

Ship is a concrete class encapsulating an ocean-going vessel that carries cargo. Since ships are powered, Ship implements the methods of the Moving interface. Ship inherits from its parent Transport. Ship implements methods to describe the different ways a ship interacts with different types of resources (e.g., a boom or a crane).

3.3.16 Aircraft Class

Aircraft is a concrete class encapsulating an airborne vessel, such as a transport airplane (e.g., a C-130) or a transport helicopter that carries cargo. Since aircraft are powered, Aircraft implements the methods of the Moving interface. Aircraft inherits from its parent Transport.

4. Future Directions

The U.S. military is being called upon to make large-scale deployments for new types of missions, including disaster relief and peacekeeping. Modeling and simulation can help the DoD plan for and improve such deployments. However, efficient and effective modeling 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 follow 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 "clearinghouse" function to manage distributed objects used by multiple simulations.

To augment this set of standards, Argonne National Laboratory has designed EXHORT, a framework of three hierarchies that together will constitute a standard and consistent class attribute representation and behavior applicable to transportation deployment simulations. Argonne is currently developing EXHORT's first hierarchy, the Transportation Class Hierarchy (TCH), and the two other hierarchies, for representing infrastructure and resources, are planned for 1999-2000. When completed, all three hierarchies will be integrated into EXHORT and used to describe major portions of the defense transportation system. EXHORT provides a standardized code structure for object-oriented deployment simulations that will help ensure meaningful data exchanges.

EXHORT eliminates 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 is currently a candidate as an Army Model Simulation Office deployment/ redeployment standard, and it provides DoD with another tool to improve the efficiency and accuracy of deployment simulations. 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.

5. Acknowledgements

This work was supported under 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.

Authors also thank Jane Andrew of Argonne's Information and Publishing Division for her editorial assistance and insightful suggestions.

6. References

[1] DMSO, 1998a, "DoD High Level Architecture Overview" [http://www.dmso.mil/hla/general/annotate/sld001.htm], Modeling and Simulation Office, U.S. Department of Defense, September 1998 (February 1999).

[2] DoD, 1999, "High Level Architecture (HLA)" [http://www.ott.navy.mil/1_3/1_3_3/index.htm], U.S. Department of Defense, March 1999 (February 1999).

[3] Kahkipuro, P., 1998, "Overview of CORBA" [http://www.cs.helsinki.fi/~kahkipur/corkur/ corbaintro.html/], University of Helsinki (April 1999).

[4] Kassay, D., 1996, "A CORBA Tutorial" [http://www.das.harvard.edu/cs/academics/courses/cs265/1996/proj/corba/kassay.htm], Harvard University (April 1999).

[5] DMSO, 1998b, "High-Level Architecture (HLA) Transition Report" [http://199.75.72.2/hla/policy/rept611.html], Defense Modeling and Simulation Office, U.S. Department of Defense, March 1998 (February 1999).

[6] Petty, M., [mpetty@ist.ucf.edu], 1999, "HLA Gateway" http://www.ist.ucf.edu/labsproj/projects/hgprode.htm], Institute for Simulation and Training, Orlando, FL (February 1999).

[7] Klein, U., Strabburger, S., and Beikirch, J., 1999, "Distributed Simulation with Java GPSS Based on the High Level Architecture" [http://www.cs.unimagdeburg.de/~strassbu/publications/websim/kleinu.html], Institute for Simulation and Graphics, Otto-von-Guericke University (January 1999).

[8] Fullford, D., Hoxie, S., and Lubetsky, B., 1998, "Transitioning Your DIS Simulator To HLA" [http://www.mak.com/tech/dis_to_hla_article.htm], MÄK Technologies, Cambridge, MA, (February 1999).

[9] Wettach, H. L., [hlwettac@adala.smith.cis.syr.edu], 1997, "A Comprehensive Discussion: The Relationship between CORBA and HLA" [http://smith.cis.syr.edu/~hlwettac/cis700/HLA.html], Center for Science and Technology, Syracuse University (February 1999).

[10] Sun Microsystems, 1999, "The Source of Java Technology" [http://java.sun.com], April 7, 1999 (April 1999).

Author Biographies

James F. Burke, Jr., is a software engineer in the 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 Illinois Institute of Technology, studying simulation component reuse and standardization. He received an M.S. in computer science from 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, CORBA, simulation modeling, and object-oriented software design and development. He is a member of IEEE.

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, 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.

Michael R. Nevins is a software engineer in the Decision and Information Sciences Division of Argonne National Laboratory. His is the project leader and lead designer of the PORTSIM model developed in the Simulation and Visualization Section. He received an M.S. in computer science from DePaul University and a B.S. in Computer Science from Elmhurst College. His research interests include simulation modeling and object-oriented software design and development. He is a member of Phi Kappa Phi, Pi Mu Epsilon, and the Society for Computer Simulation.

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 from DePaul University, and his B.S. in math 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.

Dawn L. Howard is a software engineer in the Decision and Information Sciences Division of Argonne National Laboratory. She is the project leader of the CITM model, being developed in the Simulation and Visualization Section. She received her M.S. in computer science from the Illinois Institute of Technology and has a B.S. in computer science from St. Xavier University. She also holds a B.A. in English from DePaul University. Her research interests include object-oriented development and simulation modeling.

Jill Jackson is a communications and benchmarking analyst in the Decision and Information Sciences Division of Argonne National Laboratory. She received an M.S. in environmental management from the Illinois Institute of Technology and a B.A. in communication studies from the University of Iowa. Her research interests include communicating advanced computer concepts and applications to users and decision-makers.