Multi-Resolutional Knowledge Representation Using Prototypes and Properties

Jeffrey Berliner, Michael Thome, Daniel Cerys
BBN Technologies
10 Moulton St
Cambridge, MA 02138
berliner@bbn.com
mthome@bbn.com
cerys@bbn.com

Abstract—*In the course of developing a distributed logistics command and control application based on the Cougaar agent architecture(http://www.cougaar.org/), we were faced with a large knowledge representation problem. The logistics agents needed to reason about many thousands of different types of logistics assets, where each asset had hundreds of attributes. The problem is factorable, however, in that each specialized logistics agent needs to know only a relatively small amount of information about only a relatively small number of assets. By using techniques of prototypes and delegation, adapting the paradigm suggested by Lieberman [1] and others [2, 3], we developed a logical data model (the Cougaar LDM) for logistics that effectively factors this representation problem. This factoring provides a basis for multi-resolutional representations of the entities in the logistics system; the information about any entity at each logistics agent can be limited to only that subset in which the agent has interest.*

1. INTRODUCTION

This paper describes the design and implementation of a mechanism for multi-resolutional representation of entities and concepts across a distributed set of agents. This mechanism is the basis of the Logical Data Model (LDM) component of the Cougaar Cognitive Agent Architecture [4, 5].

Our motivation stemmed from the scope of the application domain we were facing: the design and development of a logistics command and control application spanning a large number of distributed organizations, each with diverse sets of equipment, facilities and personnel, and performing a wide variety of operational and logistics support tasks. The scope of the application domain led us to an architecture based on a distributed set of agents, each of which represented a functional organizational entity. The scope and variety of the information required to describe each of the entities and concepts in this domain led us to a knowledge representation design based on prototypes and delegation as suggested by Lieberman [1], Stein et al [2], Taivalsaari [3], and others. Consideration of the detailed, but often diverse, information requirements for each agent, and the requirement to introduce additional agent types and different reasoning capabilities as the system matured, led us to extend the delegation/prototype design to a multi-level delegation mechanism using properties and behaviors which are dynamically bound to the prototypes. This multi-level delegation provides an efficient, natural mechanism for multi-resolutional knowledge representation throughout our system. As agents need to communicate about logistics entities, prototypes may be sent between the agents or constructed as needed, thus providing control over which properties and behaviors are transmitted or instantiated at each agent. In this way, the information about any entity at each logistics agent can be limited to only that subset in which the agent has interest.

This paper describes the issues arising in the problem domain that motivated this work, the principles of the design which address these issues, some details of the design and implementation, and examples of usage patterns that exploit the benefits of the design.

2. PROBLEM DESCRIPTION AND ISSUES

A key aspect of logistics planning and execution, involves reasoning about things, their properties, their relationships, and the activities in which these things participate. The things under consideration may generally be considered assets of one sort or another, and include equipment (e.g., vehicles, machinery, electronic devices, weapon systems, etc.), materiel (e.g., fuel, machine parts, food, etc.), facilities (e.g., transportation networks, airports, sea ports, repair depots, warehouses, hospitals, etc.), organizations (i.e., civil, military and commercial organizations), and individual people. In order to construct a computer system which participates in logistics planning and execution, it is

KIMAS 2003, October 1-3, 2003, Boston, MA, USA.
Copyright 0-7803-7958-6/03/$17.00 © 2003 IEEE.

necessary to have a mechanism to represent all the properties of assets required for logistics reasoning. Several aspects of these assets and their use in logistics systems make the problem hard:

- The set of asset types and asset properties is very large. These asset properties must describe the forms and functions of each asset required for logistics reasoning.
  - For equipment assets, properties include such things as physical characteristics, functional performance characteristics, environmental requirements, reliability and reparability characteristics, and relational properties such as component specifications and consumable requirements.
  - For facility assets, the properties include such things as physical and geographic characteristics (e.g., location, physical extent), functional capabilities, equipment and staffing characteristics and requirements.
  - For organization assets, the properties include such things personnel membership and skill requirements, equipment ownership and requirements, functional capabilities, and organization relation properties (e.g., superior-subordinate relations, and provider-consumer relations).
- The set of assets and properties evolves continuously over time as new models and types of equipment and materiel are continuously introduced and modified, and older ones are retired.
- Reasoning must be done over a range of granularities since varied amounts of detail are required at different locations, echelons and stages in the logistics planning and execution processes.
  - Early in a planning process, it might be known that about 4,000 people are to be deployed to a disaster site, and that they will need to provide food, water, electricity and shelter for about 90,000 people.
  - Later, it might be known that a set of mobile, diesel powered, electric generators capable of producing an aggregate of 2.5 Megawatts of electricity will be deployed.
  - Still later, it might be known that three of the generators are MEP-208A, 750 kW, 50-60 Hz, skid-mounted, Diesel Engine-Driven generator sets (NSN: 6115-00-450-5881), which have fuel tank capacities for approximately 8 hours operation at rated load.[6]
  - Finally, it might be known that one of the MEP-208A generators is Serial Number A2709B, and that this particular generator has been de-rated and limited to 600 .kW until its next major overhaul.
- Different portions of a logistic planning and execution system require different granularities of specialized knowledge. For example, each participating organization knows all the details of its operation, as well as the details of its interactions with customers and providers. However, it does not need to know or manage irrelevant information which is only required by these other organizations.
  - An emergency relief organization in operations following a hurricane disaster coordinates diverse operations including debris removal, emergency power generation, emergency roofing repair, and food water and ice supply and distribution. The logistics agents supporting its operation need to reason about the requirements for these services and the overall capabilities and availability of the providers of these services.
  - A prime power engineering organization operates and manages its power generation equipment, trains its personnel, and coordinates with the managing relief organization, with transportation providers to get to the disaster site, and with fuel supply providers.
  - A fuel distribution organization manages contracts and orders with its customers and suppliers. It manages fuel inventories, storage facilities, a fleet of distribution trucks, and a set of personnel with varied skills. Though it provides fuel for the electric generators, it reasons only about the fuel delivery requirements specified by the owner of the generators.
  - An equipment transportation organization manages contracts and orders with its customers and facility operators. It manages a fleet of transportation vehicles (trucks, aircraft and/or ships), and a set of personnel with varied skills. Since it provides transportation for the electric generators, it is required to reason only about the transportability of the generators: weight, physical dimensions, shock and rough handling limitations, etc.

3. DESIGN PRINCIPLES

In order to achieve the factoring necessary to cope with these aspects of large logistics systems, we established some basic principles for the design of our logistics knowledge representation.

- Assets are primarily modeled based on their properties rather than a hierarchical representation of what they are. It does not matter whether a towed electric generator is a generator or trailer, as long as it has the properties of a trailer, and the properties of an electric generator. (Similarly, it does not matter whether a tank is a vehicle or a weapon, as long as it has the properties of a vehicle and the properties of a weapon.)
- Related properties are collected in *property groups*. Experience has shown that attributes often occur naturally in groups, and are required by reasoning agents in these groups. For example, Physical Attributes such as Length, Width, Height, Mass, etc. are used together (e.g. when planning to pack items inside a container) and change together (e.g. when an asset is physically modified).

- Specialized property groups known as *behavior groups* encapsulate behavior about their specific properties (i.e., *methods* using objected-oriented terminology). For example, an inventory behavior group encapsulates an inventory management algorithm for a fuel storage tank, and a fuel consumption behavior group encapsulates the fuel consumption characteristics for a vehicle.
- *Asset* instances derive most of their properties from *prototype* instances. This greatly reduces the number of classes required to represent the logistics domain, and allows new types of assets to be defined and created dynamically.
- The class of an asset prototype determines the property groups which must be present in each prototype instance. This provides regularity in the normal properties of related things and allows convenient programming access to these property groups. For example, all CargoTruck prototypes have the ContainProperty. However, it not required that the class of an asset's prototype at one agent be the same as the class of that asset's prototype at another agent. This allows different agents to reason about the same assets from different perspectives. Further, because an arbitrary number of property groups can be added to an instance/prototype (see next bullet), the specific class of an asset should only be relevant to the creators of the objects (encouraging them to provide values for all required property groups). Users of these objects are encouraged to deal only with the object's set of property groups (and not the class).
- A prototype instance may include additional property groups. This provides flexibility in extending the properties of special types of assets. For example, a Truck with a hoist mounted on it can have the LiftProperty. In fact, it is not required that the class of an asset instance match the class of the asset's prototype.
- An asset actual instance may refer to specialized property groups which differ from the prototype. This provides flexibility in specializing the properties of particular instances of actual things. For example, a particular electric generator may have a degraded power generation capability.

4. Application to Logistics Command and Control

Given these principles which factor knowledge of logistics properties and behavior, we have been able to allow the detailed asset representation of any particular asset to differ depending on the details of the asset, the granularity of the processing, and the perspective or interests or needs of the using agent. Thus, the instantiated aspects (properties or attributes) of an asset change as references to that same asset move throughout the society.

Continuing the mobile electric generator example, the following are a few examples of agent roles and the corresponding property groups of interest for a mobile generator asset. An emergency electric power generation company, which owns and operates sets of mobile generators and power distribution systems, needs to know such things as the requirements for power generation, where its generators will be situated, how much fuel will be available for the generators, and when its generators will be out of service for scheduled maintenance or repair. A transoceanic shipping company, which has been contracted to ship the generators, needs to know the physical weight and dimensions of the generators, as well as the shock and rough handling limitations and the temperature and moisture limitations of each generator in its shipping configuration. A generator repair shop needs to know what is wrong with the generators they have been contracted to repair and what resources (parts, repair equipment, mechanical skills, time) will be required to repair each generator.

5. Achieving Multi-Resolutional Knowledge Representation

To achieve this multi-resolutional representation in the Cougaar LDM, an asset is represented abstractly as an object with an instance identifier (UID), a set of property groups (with property-value pairs), behavior groups (with computational methods and possibly additional property value pairs), and a pointer to its prototype. Calls to property accessors on the instance object will first check the local property groups and behavior groups, then will delegate calls to the prototype object. Typically, the prototypes have the majority of the property groups and behavior groups; only very specialized, unique assets instances are likely to have their own. Although we haven't emphasized it in our implementation, prototypes can delegate to other, more abstract prototypes. Because of this, delegation is a recursive operation that traverses the complete prototype chain.

Assets may be transferred between agents by sending the asset's UID, its collection of property groups and the UID of the prototype. On the receiving side, a new object is constructed representing the asset, and then prototype UID is resolved. The prototype resolution process first checks to see if the prototype is already known, if not, it uses a local prototype factory to attempt to construct a prototype object from local resources. Depending on the agent's particular data requirements, the factory will construct a prototype object containing only and exactly those property groups (and properties) needed for local data processing. Some such factories will not provide any details, effectively rendering the asset un-propertied while passing through the agent. The effect is that the object's shape (it's associated properties) changes radically as it moves through the network of agents.

Object aggregates may be formed by creating a new asset instance (e.g., a Box) with a key to a manifest list. The aggregate instance may then be handled as a simplified

opaque object (e.g., the box's length, weight, etc) as it moves through the agents which do not care about the details of the contents. When the aggregate is unpacked, the manifest may be de-referenced by retrieving the original asset list from the agent which performed the aggregation.

6. Examples of Assets, Prototypes, Property Groups, and Delegation

Figure 1 shows a set of typical Cougaar LDM Classes, illustrating how trucks and generators would be represented.

- The Asset class is used to represent any asset. It always has either an attribute TypeIdentificationPG which is always present for prototypes, and which refers to an instance of the TypeIdentificationPG class, or an ItemIdentificationPG, which is always present for asset instances and which refers to an instance of the ItemIdentificationPG class. It also has an attribute OtherPG, which contains a list of all other Property Groups and Behavior Groups which describe a particular asset prototype. For maximum generality, code written to deal with assets, should in general not test whether a particular asset instance or prototype is an instance of any class other than Asset. These other, specialized classes, such as Truck, are intended strictly to be for implementation optimization only.
- The TypeIdentificationPG class specifies the identity of the type of asset being represented. It has attributes typeIdentification which typically specifies an NSN (National Stock Number), altTypeIdentification which typically specifies some other domain-specific identifier, and nomenclature, which provides a human readable descriptive string.
- The ItemIdentificationPG class specifies the identity of a particular asset. It has attributes ItemIdentification, which specifies a user-specific identifier, and nomenclature, which provides a human readable descriptive string
- The Truck class is used to represent asset types which are trucks. It is a subclass of Asset, and it is guaranteed to have additional attributes such as PhysicalPG, TransportabilityPG, GroundSelfPropulsionPG, FuelConsumptionBG, and ContainPG.
- The ElectricGenerator class is used to represent asset types which are generators. It is a subclass of Asset, and it is guaranteed to have additional attributes such as PhysicalPG, ElectricGenerationBG, and FuelConsumptionBG.
- The PhysicalPG class specifies basic physical properties of assets, including length, width, height, footprintArea, volume, and mass.
- The TransportabilityPG specifies attributes of specific importance to transporters, such as air transportability, protective housing, and shock and vibration limitations.
- The GroundSelfPropulsionPG class specifies properties associated with self-propelled ground vehicles. Its attributes include tractionType (Wheeled, tracked, etc.), engineType, maxSpeed, cruiseSpeed, etc.
- The FuelConsumptionBG class specifies the normal and alternate types of fuel, and provides methods to compute fuel consumption based on usage.
- The ContainPG class specifies properties associated with things which contain cargo. Its attributes include maxLength, maxWidth, maxHeight, maxVolume, maxMass, and maxPassengers.
- The ElectricGenerationBG class specifies the functional capabilities of generators, and provides methods to determine specific capabilities as functions of the load and environment.
- The MaintenancePG class holds the maintenance log and planned maintenance schedule for a specific asset, not a

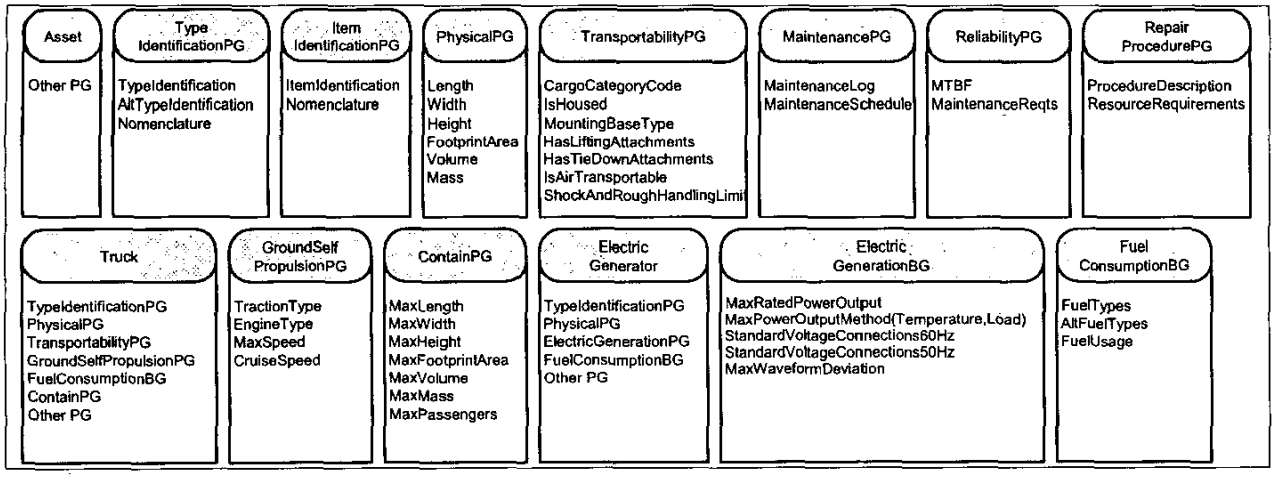

Figure 1 - Typical Cougaar LDM Asset and Property Group Classes

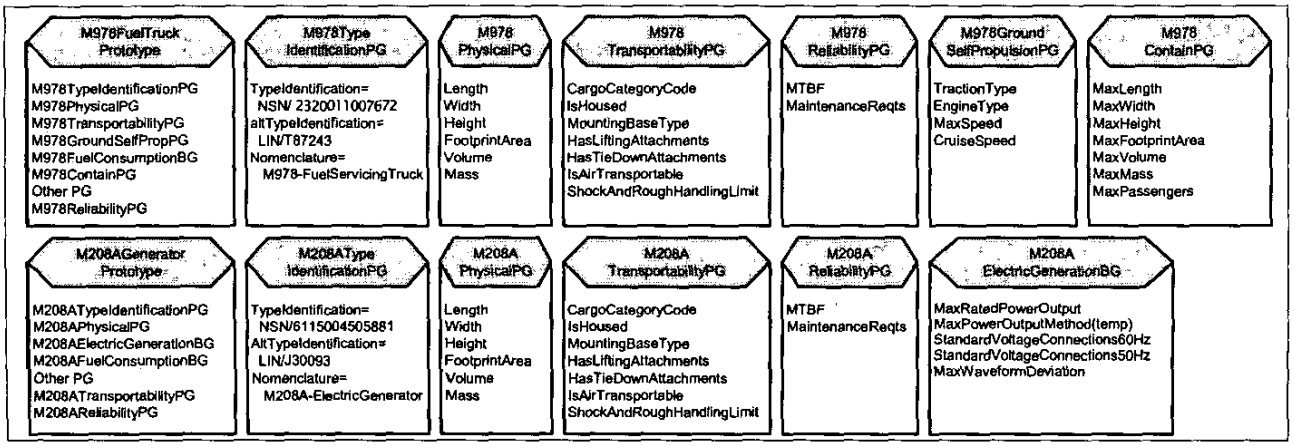

Figure 2 - Asset Prototype, Property Group and Behavior Group Instances

prototype.

- The ReliabilityPG specifies the functional reliability of an asset type, and the preventive maintenance required to achieve that reliability.
- The RepairProcedurePG holds a set of maintenance procedures for an asset. It specifies the procedures and resources (parts, repair equipment, mechanical skills, time) will be required to perform repairs of the asset. This is an example of a specialized PropertyGroup that only a particular agent type would require.

Figure 2 shows instances representing two Asset prototypes (an M978 Fuel Truck [7] and an MEP-208A generator [6]) which are instances of the Truck and Generator classes, and a set of instances of PropertyGroup classes which represent the properties of these prototypes.

The object instances which describe the M978 Fuel Truck and MEP-208A Generator prototypes are:

- The M978 Truck Prototype is an instance of the Truck class. Its attributes refer to Property Group instances which have values which describe the properties of the M978 Truck prototype. These include: M978-TypeIdentificationPG, M978-PhysicalPG, M978-GroundSelfPropulsionPG, M978-FuelConsumptionBG, M978-ContainPG, and M978-TransportabilityPG, as well as M978-ReliabilityPG.
- The MEP-208A-GeneratorPrototype is an instance of the Generator class. Its attributes refer to Property Group instances which have values which describe the properties of the MEP-208A Generator prototype. These include: MEP-208A-TypeIdentificationPG, MEP-208A-PhysicalPG, MEP-208A-ElectricGenerationBG, and MEP-208A-FuelConsumptionPG, as well as MEP-208A-TransportabilityPG, and MEP-208A-ReliabilityPG

Figure 3 shows instances representing several individual M978 trucks and MEP-208A generators. Each truck and generator instance has three key attributes:

- A reference to its prototype. All the M978 trucks share the same M978 Truck Prototype, and all the MEP-208A generators share the same MEP-208A Generator Prototype.
- A reference to an ItemIdentification Property Group. Each asset has its own ItemIdentification Property Group.
- A reference to a MaintenancePG which is specific to the individual asset, and holds the maintenance log and planned maintenance schedule for the asset.

In addition, the MEP-208A Generator, identified by S/N A2709B, has its own ElectricGenerationBG which specifies its unique, de-rated electric generation capabilities.

7. EXAMPLES OF ASSETS WITH MULTI-RESOLUTIONAL REPRESENTATIONS

Consider the interactions between several logistics agents and the different asset representations required by each of them. **Figure 4** shows four logistics agents, and their varied views of two assets:

- POL-TRKCO represents a fuel transportation company. It owns a number of M978 Fuel Trucks, including Truck-789.
- PRIMEPWR-ENGCO represents an engineering company which provides emergency electricity with its portable electric generators. It owns a number of MEP-208A Generators, including .Gen-A2709B.
- MAINTCO represents a fictitious equipment maintenance company which repairs a variety of equipment including trucks and electric generators.

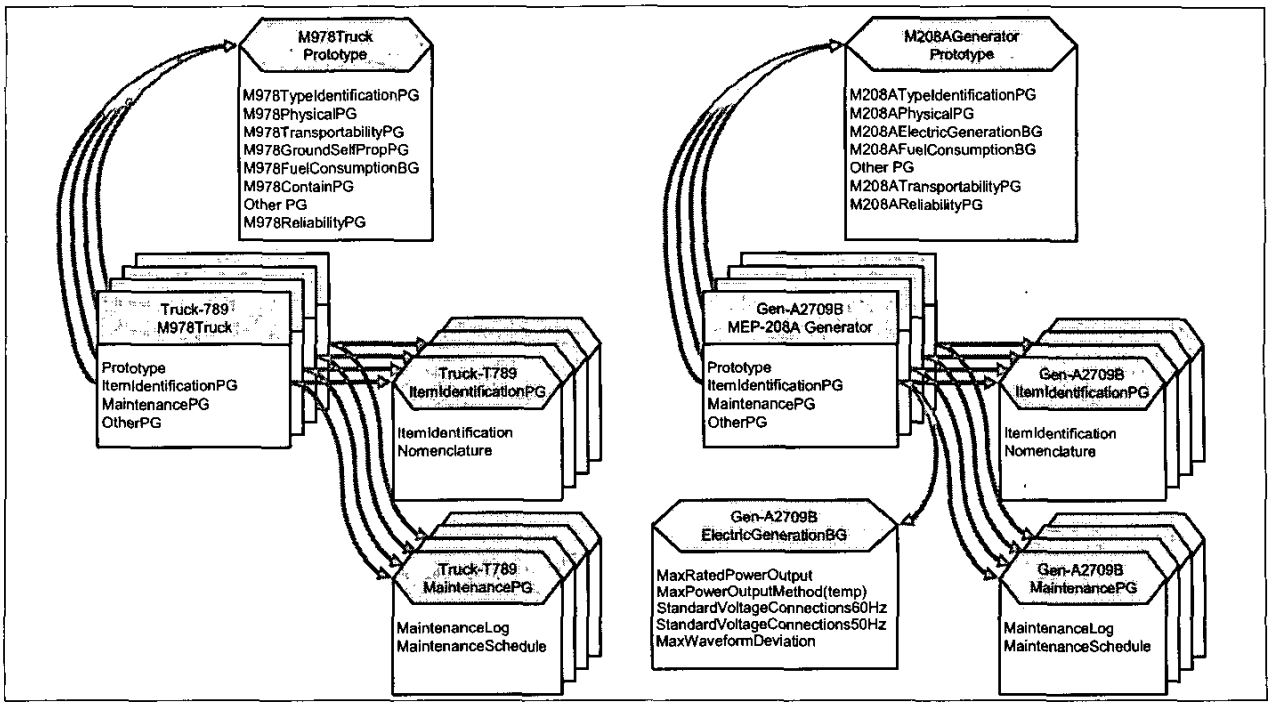

Figure 3 - Asset Instances Representing Several Individual M978 Trucks and MEP-208A Generators

- MSC represents Military Sealift Command which is responsible for shipping military cargo worldwide.

In this example, POL-TRKCO and PRIMEPWR-ENGCO send tasks to MAINTCO and MSC requesting them, respectively, to perform maintenance on their equipment, and to transport their equipment to a disaster site overseas.

As owners and managers of their equipment, POL-TRUCKCO and PRIMEPWR-ENGCO each create fully populated assets and prototypes for all their equipment. Figure 4 shows their fully populated instances of Truck-780 and Generator-A2709B. However, when references to these assets are transmitted to MSC and MAINTCO, these agents construct differently populated assets and prototypes for them. Beyond the ubiquitous ItemIdentification and TypeIdentification PGs, MSC requires only the PhysicalPGs and the TransportabilityPGs. Similarly, MAINTCO requires the MaintenancePGs and ReliabilityPGs. However, MAINTCO is required to add the RepairProcedurePG for all reparable assets, and it may also require the ElectricGenerationBGs for the generator.

In this way, each agent which reasons about an asset, needs to represent only those aspects of the assets with which it is concerned.

8. IMPLEMENTATION

The Cougaar LDM is implemented by means of a set of core components and services available to the components (generally plugins) of the Cougaar agents, as well as a set of domain-specific components which must be implemented for domain-specific applications. The core components include:

- Factory objects which create LDM objects.
- A UIDServer which supports generation of globally-unique keys for distributed objects.
- An asset prototype cache for sharing commonly-used prototypical asset references.
- A mechanism for requesting that LDM Plugins provide Properties for a new asset.

The domain specific components generally comprise:

- Prototype provider plugins to create asset prototypes
- Property and behavior provider plugins which create property and behavior groups and attach them to asset prototypes.

Instances of the class Asset may represent prototypical objects (e.g., "Model MEP-208A Electric Generator Set") or an actual, identifiable instance of such a prototype (e.g., "Model MEP-208A Electric Generator Set" serial number "A2709B"). Identifiable assets usually delegate most of their properties to a shared prototypical asset of the right type. To facilitate the sharing of such assets, the LDM allows Plugins to "cache" any prototypes which they create so that others can find them later.

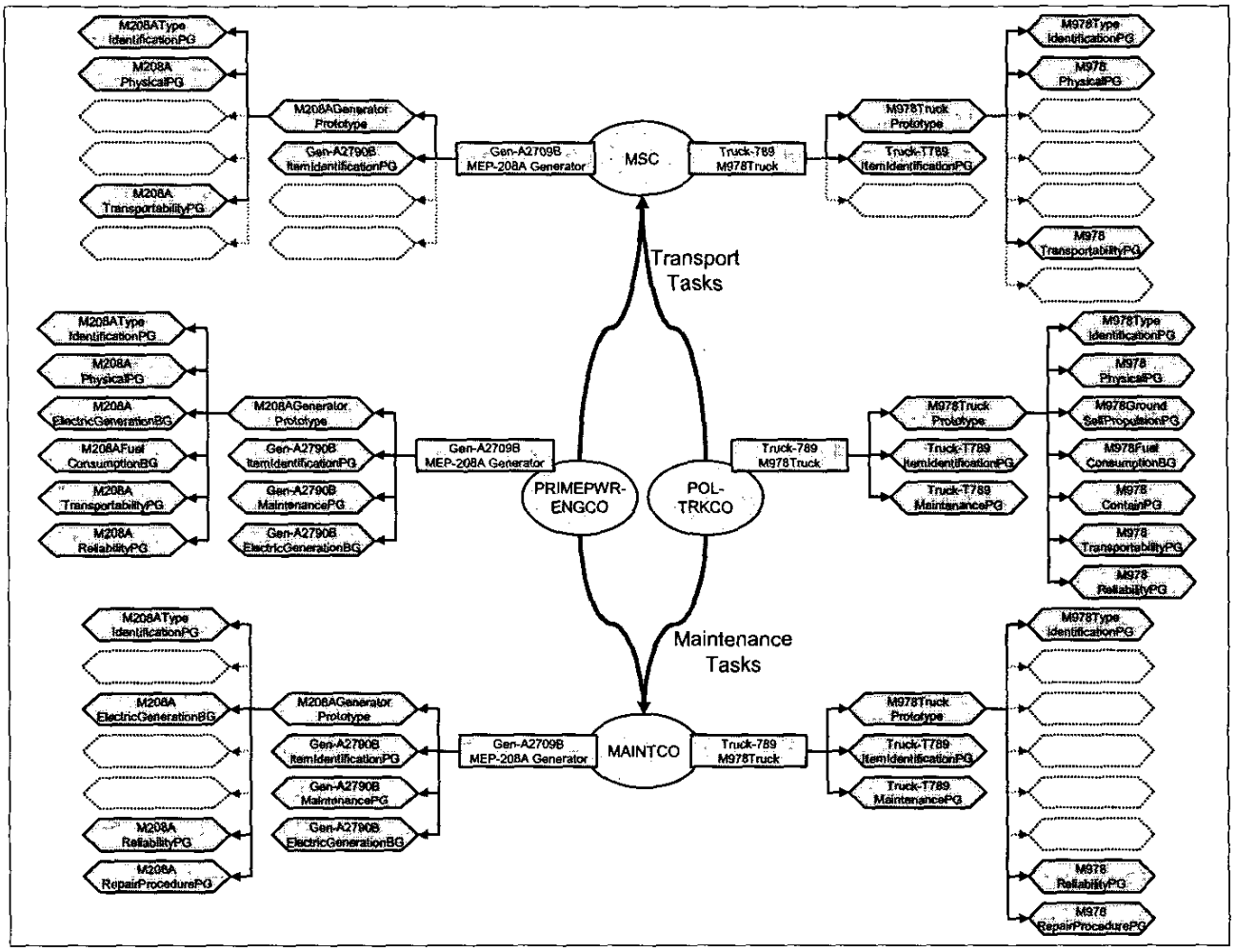

Figure 4 - Four Logistics Agents, with Multi-Resolutional Views of a Fuel Truck and an Electric Generator

There is a method on both the RootFactory and on the LDM, getPrototype() which first checks the prototype asset cache for the appropriate asset, and then invokes PrototypeProvider LDMPlugins until one is able to supply the right one. The LDM also has a set of methods which allow Plugins to create assets (especially Prototypes) without knowing all the properties which apply. These methods allow the creator of an asset to request that all PropertyProvider LDMPlugins be called to fill in whatever PropertyGroups they can.

The construction of an asset is generally initiated by either a notification that an agent has a new asset of its own to manage, or by receiving a reference to a new asset (typically as part of a task from another agent). In either case, the usual construction sequence of an asset is something like:

- A plugin (Plugin1) of an agent detects the reference to the new asset and decides it needs to create an "actual" asset of type T, because it must reason in some way about the asset
- Plugin1 asks the LDM to create a prototype asset of type T.
- The LDM looks in the prototype cache and fails to find anything matching T.
- The LDM invokes a prototype provider plugin (Plugin2).
- Plugin2 constructs a new asset prototype (Proto1) of the right class.
- Plugin2 asks the LDM to create and populate the properties of Proto1.
- The LDM invokes one or more property provider and behavior provider plugins (Plugin3 and Plugin4).
- Plugin3 adds a property group to the asset prototype (proto1) and returns.
- Plugin4 adds a behavior group to the asset prototype (proto1) and returns
- Plugin2 asks the LDM to cache the Proto1.
- Plugin1 asks the RootFactory to create an instance of

Proto1.

- RootFactory creates an instance of the asset (Asset1) which delegates to Proto1.
- Plugin1 adds Asset1 to the agent blackboard.

Thus when an agent receives a reference to a new asset type, the particular property groups and behavior groups it attaches to the asset prototype depend on the particular property provider and behavior provider plugins with which it is provisioned. By this means, each agent has control of its own view of the assets about which it reasons: assets which it owns and manages, assets for which it provides services, and assets which provide services to it.

A more detailed description of how the implementation of the Cougaar LDM interacts with the other components of the Cougaar architecture (agents, plugins, blackboard, and other components) is provided in the Cougaar Architecture Document (BBNT-2003a). Detailed usage patterns for implementing assets in the Cougaar LDM are described in detail in the Cougaar Developers' Guide (BBNT-2003b).

9. CONCLUSION

By using techniques of prototypes and delegation, we have developed a logical data model (the Cougaar LDM) for logistics that effectively factors an otherwise unwieldy representation problem. This factoring provides a basis for multi-resolutional representations of the entities in a large, agent-based logistics system; the information about any entity at each logistics agent can be limited to only that subset in which the agent has interest.

10. ACKNOWLEDGEMENTS

This work is sponsored by the US Defense Advanced Research Agency (DARPA) and is managed under DARPA's Joint Logistics Technology Office (JLTO).

11. REFERENCES

[1] Henry Lieberman, "Using prototypical objects to implement shared behavior in object-oriented systems," *Proceedings of OOPSLA '86, Object-Oriented Programming Systems, Languages, and Applications*, 214-223, November 1986, printed as SIGPLAN Notices, 21(11).

[2] Lynn Andrea Stein, Henry Lieberman and David Ungar, "A Shared View of Sharing: The Treaty of Orlando," *Object-Oriented Concepts, Applications and Databases*, Won Kim and Fred Lochovsky, eds., Addison-Wesley, 1988.

[3] Antero Taivalsaari, "Classes vs. Prototypes: Some Philosophical and Historical Observations," Journal of Object-Oriented Programming vol 10, nr 7 (Nov/Dec) 1997, pp.44-50.

[4] *Cougaar Architecture Document*, Version 10.0, http://cougaarforge.cougaar.org/docman/view.php/17/56/CAD_10_0.pdf, 1 February 2003.

[5] *Cougaar Developers' Guide*, Version 10.0. http://cougaarforge.cougaar.org/docman/view.php/17/57/CDG_10_0.pdf, 1 February 2003.

[6] MIL-HANDBOOK-633, Department of Defense, Handbook for Mobile Electric Power Engine Generator Standard Family General Characteristics, www.marcorsyscom.usmc.mil/sites/gtes/eps/DOCUMENTS/MILHDBK633.pdf, 20 February 1998.

[7] *HEMTT: Heavy Expanded Mobility Tactical Truck, M977 Series Truck*, http://peocscss.tacom.army.mil/pmHTV/pdf/hemtt_oshkosh.pdf, n.d.