

# Conveyance Estimator Ontology: Conceptual Models and Object Models

**Xiaoshan Pan, PhD**  
**Senior Software Architect**  
**CDM Technologies, Inc., San Luis Obispo, California, USA**

## Abstract

This paper proposes the construction of a Conceptual Model as a logical step prior to the preparation of the Object Model of an ontology to facilitate the design and development of software systems in which a high-level internal representation of context supports some intelligent capabilities. The intent of the Conceptual Model is to be expressive for human interpretation utilizing descriptions that are readily understood by laypersons, subject matter experts, and software developers who may be concerned with only a particular portion of the software system. The intent of the Object Model (as a subset of an ontology) is to be expressive for machine interpretation. The author argues that the existence of a Conceptual Model not only serves as an effective communication vehicle among the various stakeholders in a software development project, but also facilitates the development of the Object Model component of an ontology.

**Keywords:** Conceptual Model, Object Model, ontology, representation, software development, UML, Unified Modeling Language

## 1. Prolog

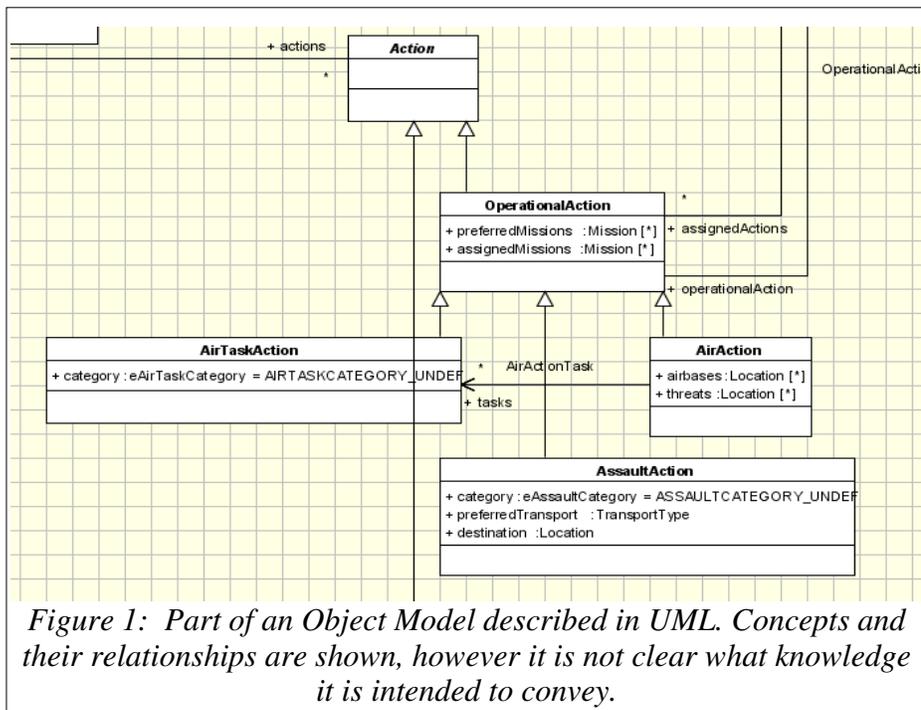
In the beginning, there was the *World*. And then humans came. Humans studied the things that they perceived as being in the *World*, which they called *Ontology*. *Ontology* helped humans communicate with each other, share and reuse knowledge. Later, humans developed computers and the software systems running on the computers that helped to solve problems about specific *aspects* of the *World*, called *Domains*. In the process, humans learned that a computer system needed to have a *representation* of a *domain* to develop and specify problem-solving strategies for the *domain*. Such a *representation* was called by different names depending on the technology employed, such as *Variables* and *Data Structures*. Popular contemporary terms include *Object Model*, or *Ontology*.

## 2. Introduction

In the context of the computer and information sciences, an ontology is the knowledge representation of a domain in the world, and its purpose is to enable "... knowledge sharing and reuse" (Gruber 2008). Practically speaking, an ontology or the knowledge about a domain comprises three elements:

1. the concepts (i.e., objects with attributes) used to describe the domain, which provides shared meanings to those working within the domain;
2. the relationships and constraints among the concepts; and
3. the behaviors or business rules typically describing scenarios about the interactions among the concepts in the domain.

Although an Object Model might be targeted as the vehicle of the knowledge representation for a domain when constructing a computer software system, this target has not yet been able to fully function as an ontology. The reason is simple but not obvious – while an Object Model is able to represent the concepts, the relationships among concepts, and the constraints among the concepts (to some extent), the technology has not reached maturity where it can fully describe the behaviors (i.e., the business rules) that relate to the concepts. In software development, business rules are typically implemented separately from the Object Model. For example, a system built with a three-tier architecture contains a Presentation Tier, an Information Tier, and a Logic Tier. Business rules are implemented in the Logic Tier (e.g., agents) and the Object Model resides in the Information Tier. Therefore, as a medium to represent knowledge, an Object Model alone is not sufficient. In other words, an Object Model is not equivalent to an ontology (Figure 1).



*Figure 1: Part of an Object Model described in UML. Concepts and their relationships are shown, however it is not clear what knowledge it is intended to convey.*

Another reason why Object Models and ontologies are not equivalent is that they can have different scopes. An ontology generally is built for capturing, sharing and reusing domain knowledge, and it can be applied to many areas as long as these areas have need for this knowledge. An Object Model is often a partial implementation of an ontology, meaning only a

subset of the concepts and relationships are implemented. The choice of these concepts and relationships is determined by the specific functional requirements and use-cases of a software system.

Since an Object Model is usually tied to a specific implementation and described in a formal language such as the Unified Modeling Language (UML), the words (or symbols) used in the model are most meaningful to the modeler himself, and can be difficult to interpret<sup>1</sup> by other users (e.g., the users who might implement the business rules of the system). This is particularly the case when modeler's intentions and knowledge are not clearly documented and accessible to others. Therefore, as a medium providing shared meaning and understanding among human users, an Object Model is not an ideal choice<sup>2</sup>. Practice has shown that the more complex and expressive an Object Model is the more difficult it is to comprehend and thus to gain general acceptance.

The preceding discussion suggests that it is necessary to recognize the limitation of what an Object Model can represent functionally, and its insufficiency as an *ontology*. How then should an ontology be specified? The answer depends on who the ontology users are. If the users are software systems, then an ontology, in principle, can be represented by an Object Model, together with the business logic describing the behavior of the objects as defined in the model (possibly implemented as software agents). However, if it becomes difficult to retrieve and reuse knowledge provided in this way, then the purpose of an *ontology* is undermined. If the users are humans, including the clients who requested the system, domain experts, system architects, and component developers, a Conceptual Model may be more appropriate for presenting an ontology.

A Conceptual Model contains the domain concepts and their relationships described in plain English words. The relationships must be sufficiently flexible to describe not only normal UML relationships (e.g., inheritance, aggregation, composition, etc.) but also the behavior of the concepts as well. For example, Figure 2 shows a Conceptual Model about the Martian atmosphere. The model is composed of the following elements:

1. the concepts as used by the domain users, such as “Protective Ozone layer”, “Nitrogen”, and “Composition”;
2. the relationships among the concepts, such as “Composition includes Nitrogen, Argon, Carbon Dioxide”; and
3. the behavior of the concepts (i.e., describing the interactions among the concepts). For example, “Greenhouse gas absorbs some heat re-radiated by surface” and “General circulation of atmosphere leads to surface erosion”.

One can easily construct a Conceptual Model for the purpose of capturing, preserving, and sharing domain knowledge, which is precisely what an ontology is supposed to do. Therefore, it

---

<sup>1</sup> Complex and expressive object models can easily lead to different interpretations or misinterpretations.

<sup>2</sup> Note that this does not pose a problem to non-human users. An Object Model by far is still the most effective medium to provide shared understanding about a domain among multiple software systems.

seems reasonable to suggest the following propositions for the development of complex software systems<sup>3</sup>:

- 1) Subdivide the traditional development of an ontology into two distinctive processes: the development of a Conceptual Model; and, the development of a corresponding Object Model.
- 2) The Conceptual Model is best suited for capturing, preserving, and sharing domain knowledge among human users, while the Object Model is best suited for providing a shared representation about the domain in terms of objects and their relationships among different system components and/or different systems.
- 3) An Object Model is an implementation (or partial implementation) of its corresponding Conceptual Model in supporting some specific use-cases and system functionalities.

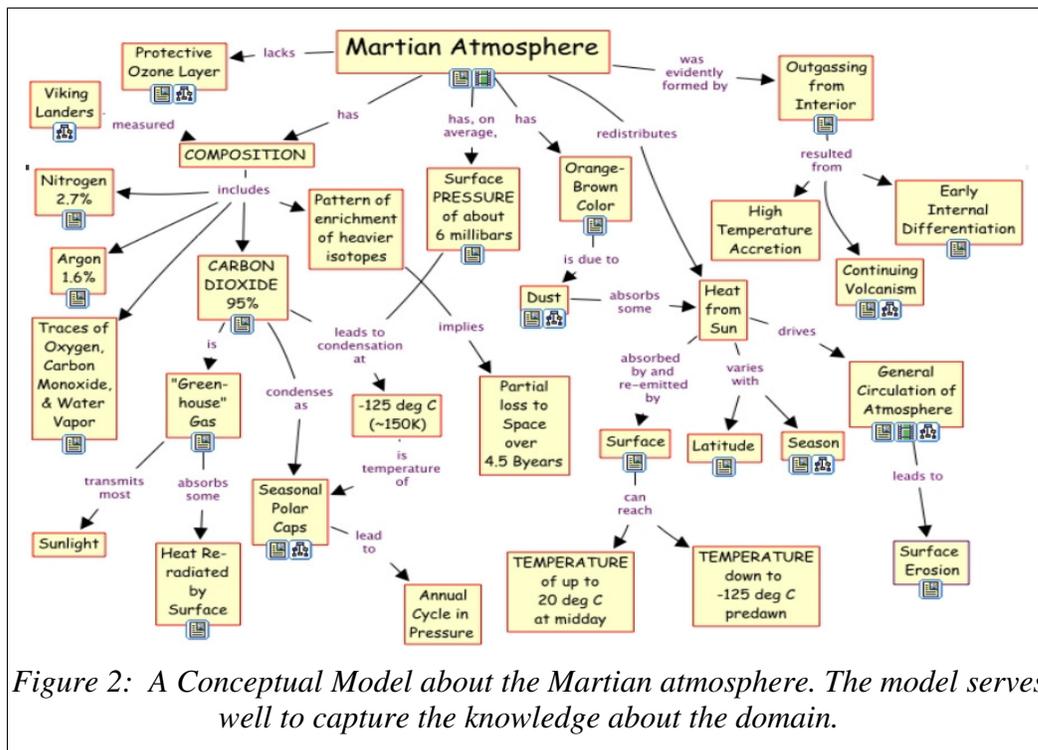


Figure 2: A Conceptual Model about the Martian atmosphere. The model serves well to capture the knowledge about the domain.

In this paper the author will refer to the development of the ontology for a Conveyance Estimator software system as a case study, to describe how the proposed idea was implemented in a specific project context. The objective of the Conveyance Estimator (CE) project is to provide capabilities for the assessment of logistical resource requirements—determining the approximate number of pallets, containers, aircraft, railcars, or trucks required to transport a given mix of

<sup>3</sup> These propositions may be necessary in developing all but the very simplest systems.

supplies and equipment. At the micro level, the validity and accuracy of such estimation is largely dependent on the procedure used to optimally load a single container, or a conveyance, which is a complex problem in itself<sup>4</sup>. At the macro level, because different conveyance types represent different sets of constraints with significantly different cargo capacities (e.g., tens of thousands of items for a ship and less than a hundred items for a truck), it is reasonable for such an estimate to be made at both a fine-grained and a coarse-grained level.

### 3. Conceptual Model

The Conceptual Model for the Conveyance Estimator consists of a set of graph-like concept maps. Concept maps are graphical tools for organizing and representing knowledge (Novak and Canas 2008). The top level of the model is a “root” map (Figure 3), which defines a list of high level concepts and their relationships that shape the project, such as “Problem Statement”, “Objectives”, and “Conveyance Estimator Problem”. Other concept maps are the expansions of these high level concepts.

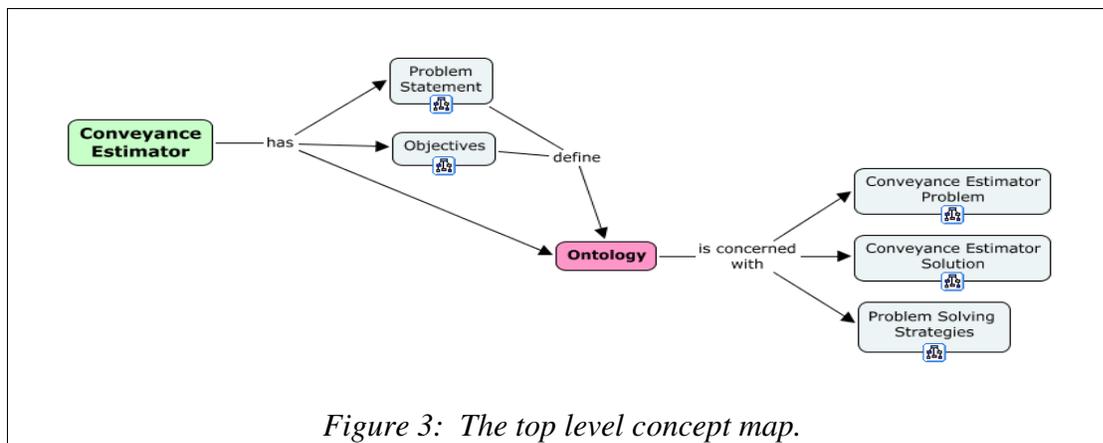


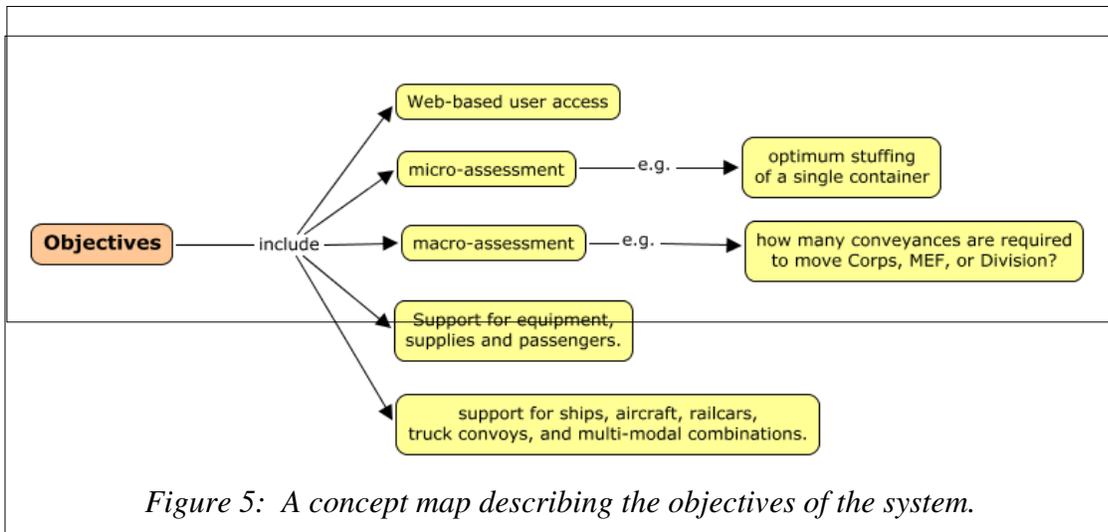
Figure 3: The top level concept map.

The structure of this Conceptual Model is simple: each model is a collection of concept maps. A concept map is comprised of nodes and arrows. Both nodes and arrows can be attached to words or symbols. By convention, nodes represent concepts, and arrows represent relationships. Since verbs are freely used to describe relationships among concepts, such a graph can be easily used to describe the behavior of the concepts. Importantly, the model is presumably comprehensive to laypersons. For example, the top level concept map (Figure 3) can be read as “Conveyance Estimator has Problem Statement, Objectives, and Ontology”, and “Ontology is concerned with Conveyance Estimator Problem, Conveyance Estimator Solution, and Problem solving Strategies.”

<sup>4</sup> Optimally stowing items into a three-dimensional container represents a bin packing problem, which is a combinatorial NP-hard problem within the realm of computational complexity theory.

Each concept within a concept map can be further expanded as another concept map. For example, the concept “Problem Statement of Conveyance Estimator” is further expanded as another concept map as shown in Figure 4. Similarly, any concept in Figure 4 can be expanded as well. For example, we could create concept maps to explain “equipment” and “supplies” (but that has not been done here since their meanings appear to be understandable.)

Figure 5 contains a concept map describing the objectives of the system, which include “Web-



based user access”, “micro-assessment”, “macro-assessment”, and so on.

Figure 6 describes the input of the Conveyance Estimator system. The input includes “requirement” and “planning options”, and “requirement” is further defined as lists of “available conveyances” and “cargo items”, and “conveyances” can be “pallet”, “container”, “truck”, “railcar”, “ship”, and “aircraft”, and so on.

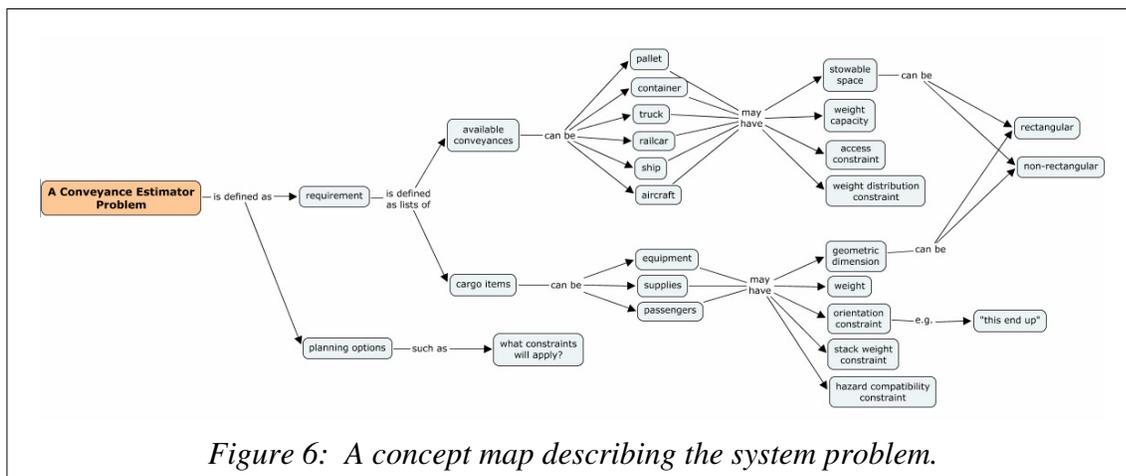


Figure 7 contains a concept map describing the kind of output that the Conveyance Estimator system produces. For example, one part of the output is a list of “packed conveyances” that holds the following information: “constraints applied to the package”, “cargo items in the package”, “packaging sequence”, and “meta info”.

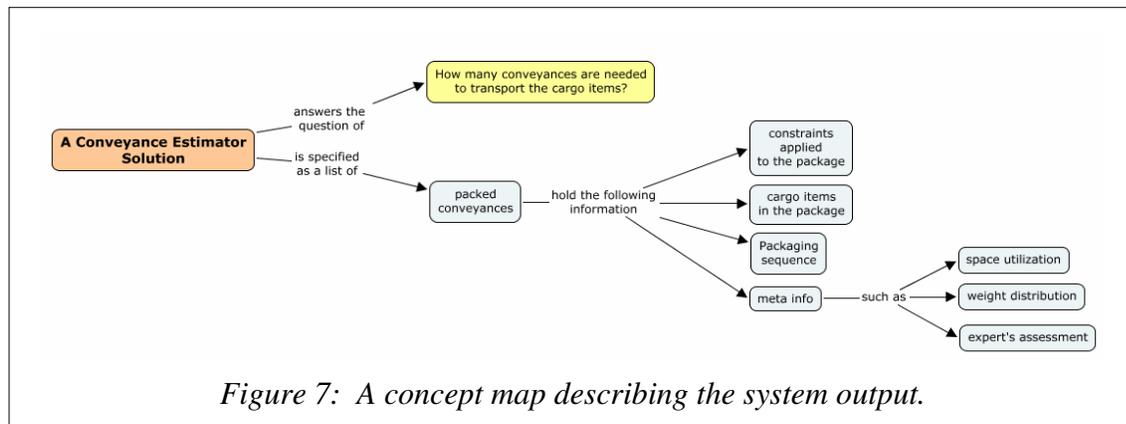
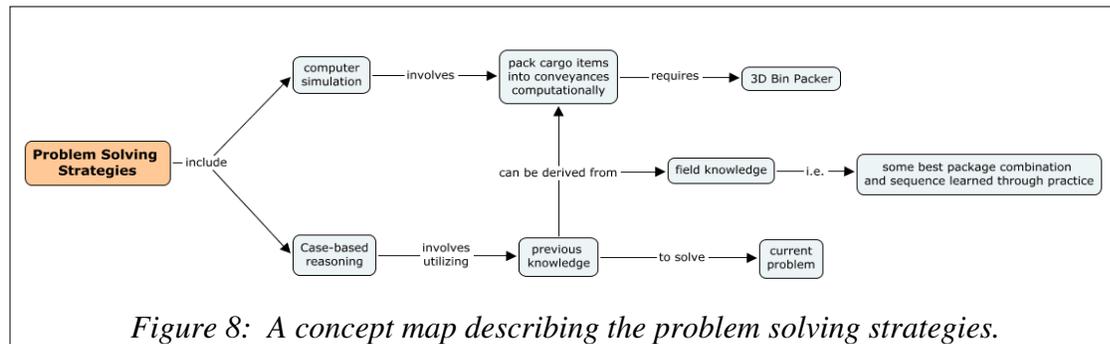


Figure 8 contains a concept map describing the possible “problem solving strategies” for utilization in the development of the Conveyance Estimator system, which include “computer simulation” and “case-based reasoning”.



#### 4. Object Model

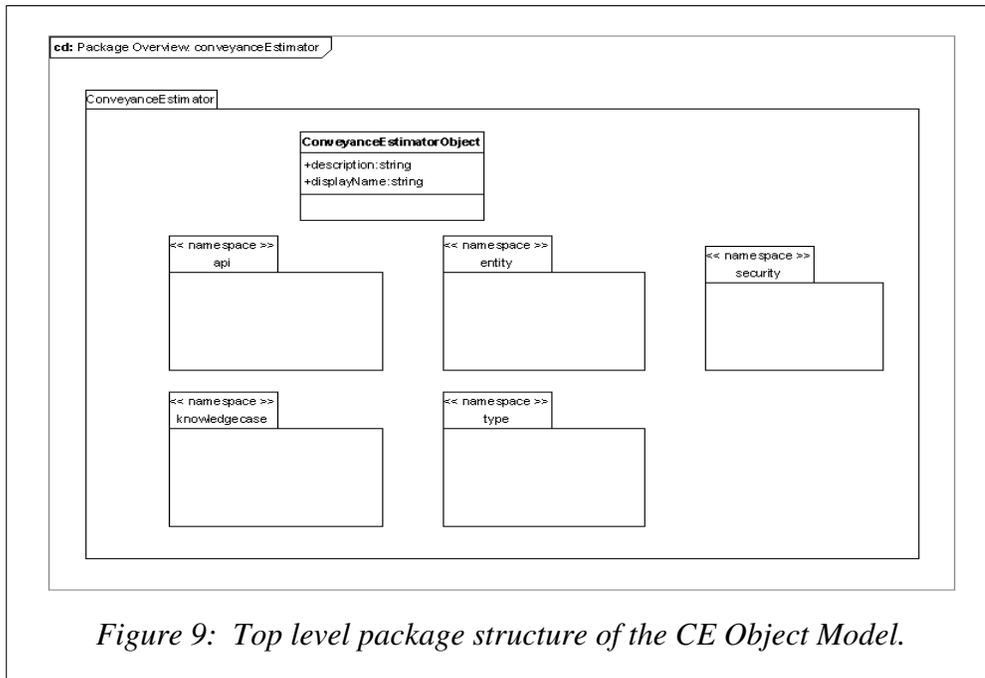
By first looking at the Class Diagrams of the Conveyance Estimator (CE) Object Model<sup>5</sup>, the direct correspondence between them and the Conceptual Model as described in Section 2 may not be obvious for the following reasons:

1. While the Conceptual Model describes the high-level domain knowledge that the developers have gained through interaction with the customer, the domain experts, and the project advisory committee, the Object Model contains only the subset of concepts and relationships that are directly supportive of the limited functional requirements of the system. For example, in the concept map shown in Figure 6, an “available conveyance” is described as either “pallet”, “container”, “truck”, “railcar”, “ship”, or “aircraft” type; however, in the Object Model as shown in Figure 10, there are only three types of bins (i.e., conveyances) – Pallet, Container, and Truck. This is because the developers decided that the first version of the Conveyance Estimator system would support only pallet, container and truck stowing. Therefore, adding other types such as “railcar” and “ship” was not necessary. Moreover, because the system will not initially support “railcar” stowing, adding a type “railcar” to the Object Model could confuse the developers (e.g., a GUI developer might create a report for “railcar” even though it is not supported by other components in the system).
2. The Conceptual Model and the Object Model follow different naming conventions. In the Conceptual Model, the words/sentences attached to each node and each arrow are plain English words/sentences. This is done for the sake of clarity, readability and ease of understanding. In the Object Model it is possible to use plain English words for some cases, but not for all cases<sup>6</sup>. For example, in the Conceptual Model as shown in Figure 6, an “available conveyance” has “stowable space”, which can be either “rectangular” or “non-rectangular”; however, when that information is translated into the Object Model it becomes: a “BinType” (i.e., “available conveyance”) has “widthLimit”, “lengthLimit”, and “heightLimit”. This is because the modeler has decided that: (1) a “stowable space” will be treated as being “rectangular” in the system; and, (2) the width, length, and height of a “stowable space” will be called “widthLimit”, “lengthLimit”, and “heightLimit” in the model.
3. While the Conceptual Model is described in a non-constrained, very flexible manner, the Object Model must be described in a formal language such as UML. For example, an Object Model is organized in terms of “name space”, “package”, “class”, “attribute”, “data type”, “generalization relation”, “aggregation relation”, and so on. Such formality is necessary so that the model descriptions are precise enough to be processed by software. However, there are no such restrictions in the Conceptual Model that is processed by human users.

---

<sup>5</sup> Please refer to the documentation of the CE Object Model at:  
<\\datafish\jmadss\ConveyanceEstimator\doc\index.html>

<sup>6</sup> In UML, one convention could be that white spaces are not allowed as part of a symbol.



*Figure 9: Top level package structure of the CE Object Model.*

4. While the Conceptual Model holds no other concerns than trying to describe the domain knowledge, the Object Model does involve some specific implementation concerns. For example, the package structure design of the CE Object Model (Figure 9) is largely based on how the system will be built and interfaced with other systems. For example, the CE Object Model has five first-level packages – “api”, “entity”, “type”, “knowledgecase”, and “security”; the packages “entity” and “type” are for holding the objects that exist in the domain; the package “api”<sup>7</sup> is for holding the objects that will interface with other systems; and, the package “knowledgecase” is for holding the objects that implement the “Case-Based Reasoning” strategy.

<sup>7</sup> “api” originally refers to Application Program Interface; here it is a placeholder for the objects that will be interfacing with other system, considering that CE system will be developed as a web-based service.

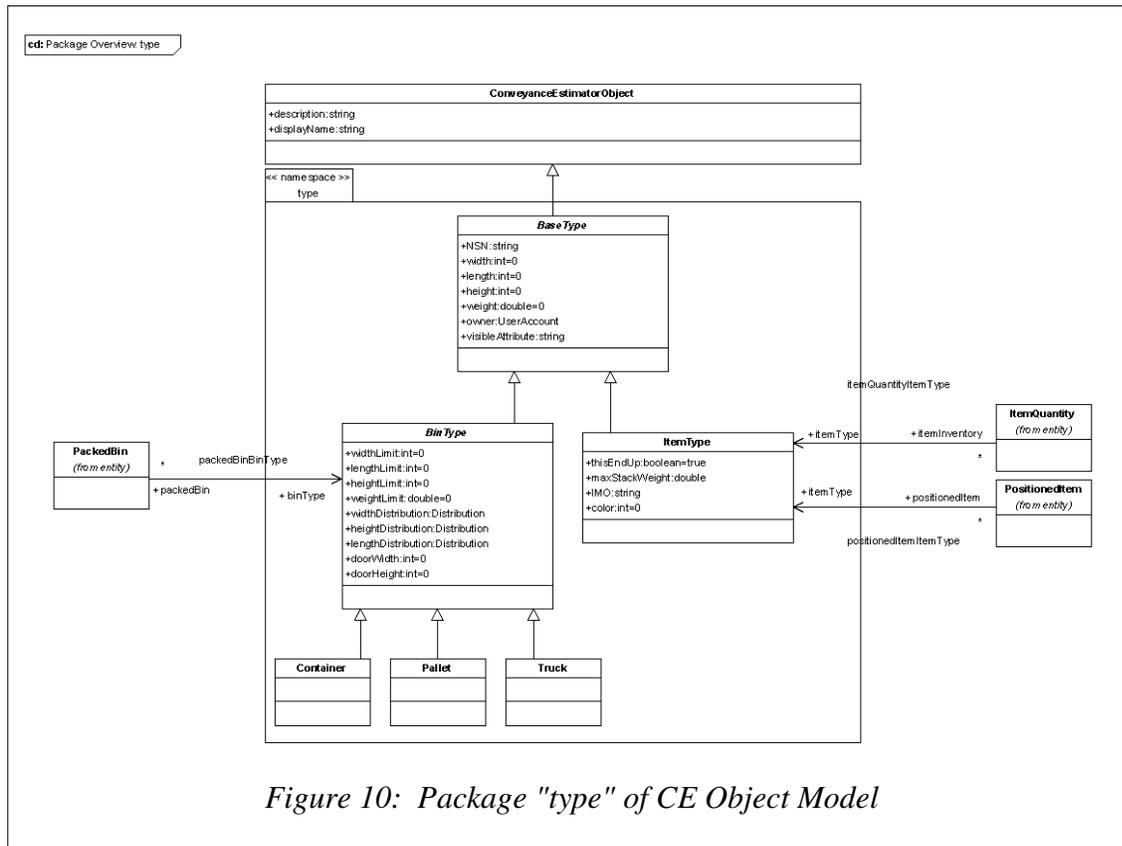


Figure 10: Package "type" of CE Object Model

With all of the differences mentioned above, the two models are still very much coupled, due to the fact that the Object Model was developed utilizing the CE Conceptual Model as a guideline.

## 5. Summary

Based on the experience gained in the development of the Conveyance Estimator Ontology, developing a distinct Conceptual Model prior to the development of an Object Model has shown to be beneficial in the following ways:

1. The Conceptual Model helps the modeler to capture (or describe) the domain knowledge more quickly and comprehensively.
2. During the development of the Conceptual Model, the modeler gains a better understanding of the domain concepts, the business rules, and the functional requirements, which will later help to clarify what needed to be built into the Object Model. Such clarity led to the smooth development of a clean and efficient Object Model for the CE system.

3. While the modeler can make the Conceptual Model as expressive as desired, the corresponding complexity is not necessarily reflected in the Object Model if it is not required to support the functional requirements of the system. Therefore, the Object Model users, such as component developers, do not have to deal with such complexity and can focus on component implementation in a straightforward way by working with a much simpler model.
4. The separation of the Conceptual Model from the Object Model appears to be a plausible solution to the on-going frustration between ontology developers and functional component developers. Such a separation makes it possible to satisfy the needs of both. The Object Model can be kept clean and efficient, with its evolution largely driven by the system's functional requirements and use-cases, while the Conceptual Model can grow as complex and expressive as needed to capture and preserve the domain knowledge.
5. A Conceptual Model serves as a means to understand its corresponding Object Model (especially in the case of complex and non-intuitive models), which directly facilitates the reusability of the Object Model across different applications.

There are many ways of utilizing a Conceptual Model in support of software project development. Because the general purpose of the model is for organizing and representing knowledge, depending on the types of knowledge it works with, a Conceptual Model can represent system architecture design, process flow, organization design, and many others aspects of a project, in addition to representing domain knowledge<sup>8</sup>. However, as far as the purpose of an ontology is concerned, in this writing the author has focused on its function in capturing, representing, and reusing domain knowledge.

Furthermore, because it would be most useful if a Conceptual Model can automatically synchronize with its corresponding Object Model<sup>9</sup>, or vice versa, the next step in this research study will include exploring the possibility of auto-processing a Conceptual Model into a more structured and formalized model (even into an Object Model directly). A recent release of CmapTools (IHMC 2008) has support for building formalized ontologies utilizing concept maps. For example, CmapTools provides a set of predefined relation types for a user to choose from, such as “actives”, “are”, “type of”, “at least”, and “exact opposite of”, and the semantics of these types can be defined in a machine processable format. In addition, the tool allows a concept map to be exported to and imported from standard XML files, which provides ready opportunities for tools to be created for processing a Conceptual Model automatically.

## 6. References

IHMC (2008). *IHMC CmapTools*. Available at: <http://cmap.ihmc.us/>

---

<sup>8</sup> For example, the concept map as shown in Illustration 3 describes the high-level input and output of the CE system, which is a reflection of the design of the system architecture.

<sup>9</sup> Currently the synchronization between a Conceptual Model and its corresponding Object is performed manually.

Gruber, T. (2008). "Ontology" *Encyclopedia of Database Systems*, Ling Liu and M. Tamer Özsu (Eds.), Springer-Verlag.

Novak, J. and Canas, A. (2008). *The Theory Underlying Concept Maps and How to Construct and Use Them*, Florida Institute for Human and Machine Cognition. Available at: <http://cmap.ihmc.us/Publications/ResearchPapers/TheoryCmaps/TheoryUnderlyingConceptMaps.htm>