Published in the Proceedings for the Inaugural Annual Conference on Systems Integration, March 12-14, 2003.

Modeling a System of Systems Using UML

Rob Cloutier, Lockheed-Martin Corporation, robert.j.cloutier@lmco.com
Andrew Winkler, Lockheed-Martin Corporation, andrew.j.winkler@lmco.com
John Watson, WOW IS, john.watson@lmco.com
Clay Fickle, Lockheed-Martin Corporation, clay.fickle@lmco.com

Abstract

The increased complexity of systems today has brought about the notion of "Systems-of-Systems". The traditional Systems Engineering approach to modeling complex systems is to perform a functional decomposition to determine the major functions of the system and once that is accomplished, to employ the same technique again to further break the larger functions into smaller ones. This approach tends to bias the system description to the engineer's perspective rather than to the user perspective and can lead to technologically sophisticated systems that may not meet user needs. In contrast, this paper presents a modern object oriented systems engineering (OOSE) approach to the problem of system of system specification and decomposition. The approach determines how the user expects the system to perform in the many different situations in which it will be used and creates a use case model capturing these scenarios and a class model capturing domain information and functionality. These two UML models provide sufficient information to create a partitioning of the system into subsystems, with each subsystem defined by a survey of its own set of use cases and domain information classes. This survey constitutes a high level specification of the subsystem. The surveyed subsystem use cases collaborate to accomplish the system level use cases and the technique for subsystem identification is iterative.

1.0 Introduction

Modeling a system of systems requires one to build multiple models as the design is detailed. It is important to understand the hierarchy of these models, and the relationship of the various models to one another. The top level model is the enterprise model, or the System-of-Systems model. This model represents the context in which the system being built resides. The next model to be created is the system model itself. The system model specifies the overall information and functionality of the system, and will allow the architect to begin to allocate functionality to the subsystems logically decomposed within the system. Finally, the subsystems are modeled, capturing the services, components and classes for each subsystem. Figure 1 represents this model hierarchy. It is important for the reader to recognize that depending on the complexity of the system, the subsystems may need to undergo further logical decomposition into smaller subsystems that might be referred to as segments, sub-segments, etc. The goal is to reach a manageable definition without undo complexity which can serve as a basis for design modeling.

The steps to be discussed in this paper are listed below. These steps are written from the perspective of extracting Subsystem models (use case and class survey) from a System model. These same steps can be followed to extract the System model from the Enterprise Model, however frequently in practice the System boundaries are predefined for the application. In this case, it is still very important to produce the enterprise model as a basis for extracting System use cases and for producing the initial set of domain classes.

The first two steps define the System model in detail. The next three steps extend the system model with the goal of identifying subsystems and specifying each subsystem with a use case survey and domain object model survey.

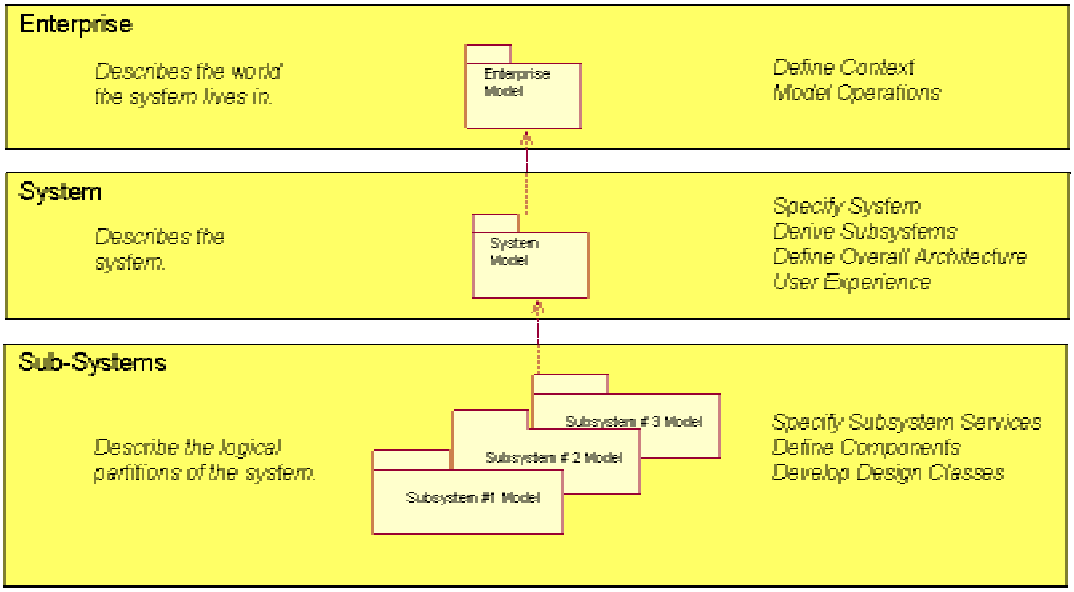

Figure 1 – Model Hierarchy

1. Define System model context
2. Develop System Use Case and Domain Class Models
   a. Survey System use cases
   b. Capture domain "language" in class model
   c. Detail System use cases using activity diagrams
   d. Enhance activity diagrams with domain object instances
3. Recapture activity diagrams as sequence diagrams using methods on domain objects
4. Group domain classes together to form Subsystem class survey based on class cohesion and loose subsystem coupling as defined by the sequence diagrams.
5. Identify sequence diagram sub-sequences to extract Subsystem use case survey.
6. Produce Subsystem context diagram based upon use case survey actors.

For the purposes of this paper the Enterprise model will be a naval surface ship operating within a naval battlegroup. The Navy Surface Ship is a System-of-Systems including Weapons System, Sensor System, Crew, and Command & Control System. For our sample problem the Command & Control System will be the System that needs to be developed. With the exception of the Crew, the existing systems and system interfaces must be maintained without change. Figure 1a represents the Systems within the naval ship. We will show how to use the Enterprise model to produce a survey of the use cases, domain classes and context diagram for the Command & Control System. We will then discuss how the same steps could be followed to identify a set of Subsystems of the Command & Control Software System.

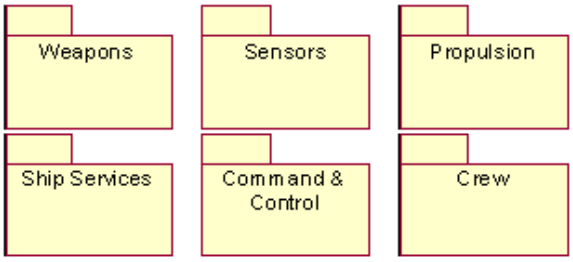

Figure 1a – Systems within the Navy Surface Ship System-of-Systems

2.0 Define Enterprise Context

This approach requires one to first understand the System-of-Systems, the enterprise, in which the System will operate. This is done by creating a context diagram of the enterprise. The enterprise context diagram is an anatomy diagram in which the enterprise to be modeled is placed in the center of the diagram with the external systems connected through links to the

enterprise. Figure 2 represents a simple enterprise context diagram. It creates the boundary of the system.

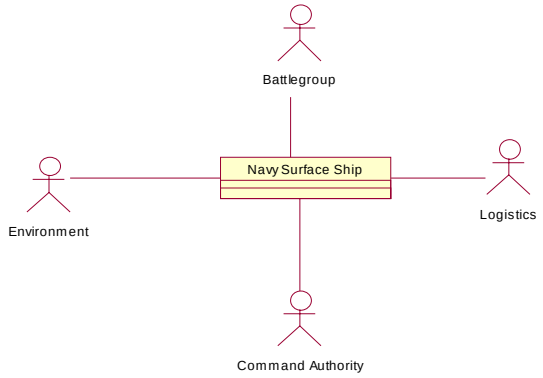

Figure 2- Enterprise Context Diagram

2.1 Develop Enterprise Use Case and Domain Object Models

The development of the Enterprise Use Case Model and Domain Object Model are tightly coupled and highly iterative activities. But, the process "begins" by developing a survey of the Enterprise use cases that are relevant to the Command & Control System. Then the use cases are detailed using activity diagrams and the Domain Object Model is built. Finally, the activity diagrams are enhanced with domain object flows.

2.2 Enterprise Use Case Survey

A use case survey is a list of use cases where each use case is described in a few sentences and the primary actors for the use case are identified. Developing the Use Case Survey, using UML use case notation, can be done in a variety of ways, but we have found it is best accomplished through interviews with domain experts/subject matter experts. Alternately, the domain experts can be the ones actually creating the models. The goal is to capture what the primary actors need from the Enterprise or what the Enterprise needs to do for them, and how the secondary actors need to contribute to meet this goal. Each of these associated sets of needs or primary actor goals becomes a use case. Examples of this in the context of this paper include Perform Air Self Defense and Re-supply Ship, represented in Figure 3.

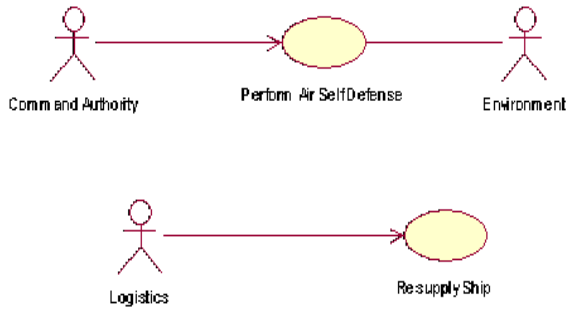

Figure 3- Enterprise Use Cases

2.3 Domain Object Model

A Domain Object Model is a set of logically grouped Classes and their relationships captured on class diagrams, where each class represents a significant piece of domain information. At this point in the process, the classes do not have methods and may not have attributes, but each class has a clear, written definition. The classes include real world information the users depend on to perform their tasks which are represented in the use cases. Information can take many forms both external and internal to the system. Examples of this external information in the context of the navy ship include publications, documents, electronic messages, hand written logs. Examples of internal information include Tactical Picture other entities being tracked by the sensors, plans, and engagement policies. We have found interviews with domain experts an excellent source for the identification of classes.

The DOM provides three major benefits. They are:

1. A mechanism to couple what may seem to be a loosely related set of use cases by virtue of the information's persistence
2. Provide the framework for the development of analysis and design classes later in development.
3. Precise definition of the System vocabulary from the user perspective

In our example the Systems comprising the Enterprise Model are known, so each internal class is assigned to the System that "owns" the information. Emphasis is placed on identifying external classes and classes owned by the C2 System.

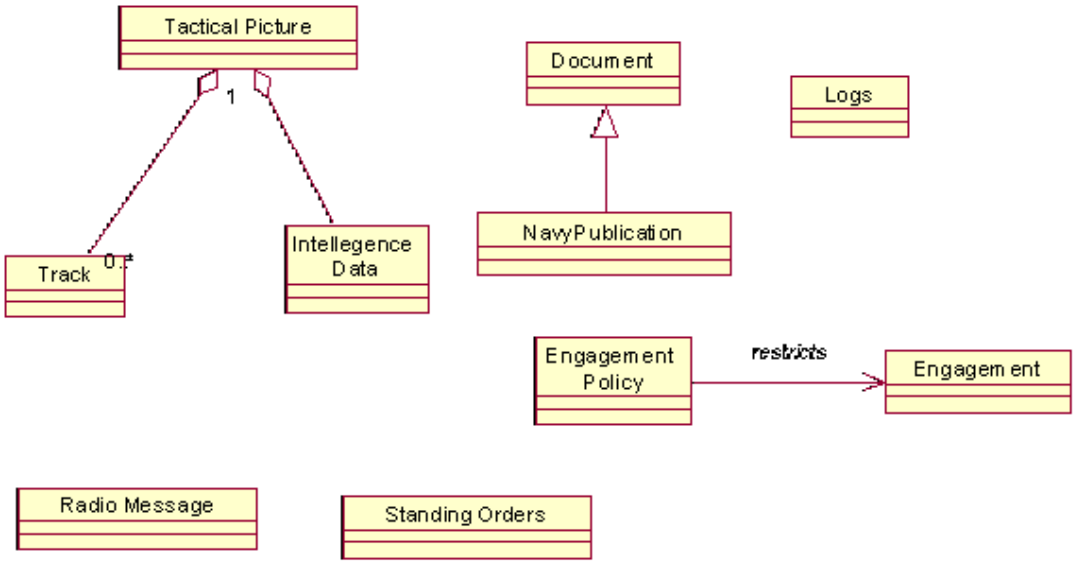

Figure 4 – Domain Object Model

For detailed reasons that will be noted later, add a proxy class representing each System. Figure 4 is an example of a DOM.

2.4 Detail Use Cases with Activity Diagrams

An activity diagram details a use case by capturing its various scenarios. The activity diagrams are made up of objects, activities, and event relationships that capture the interactions between the Systems and the external actors. It is important to note that activity diagrams are not data flow diagrams, as activity diagrams show the flow of events not the flow of data. An example of an activity diagram for "Perform Air Self Defense" is found in Figure 5.

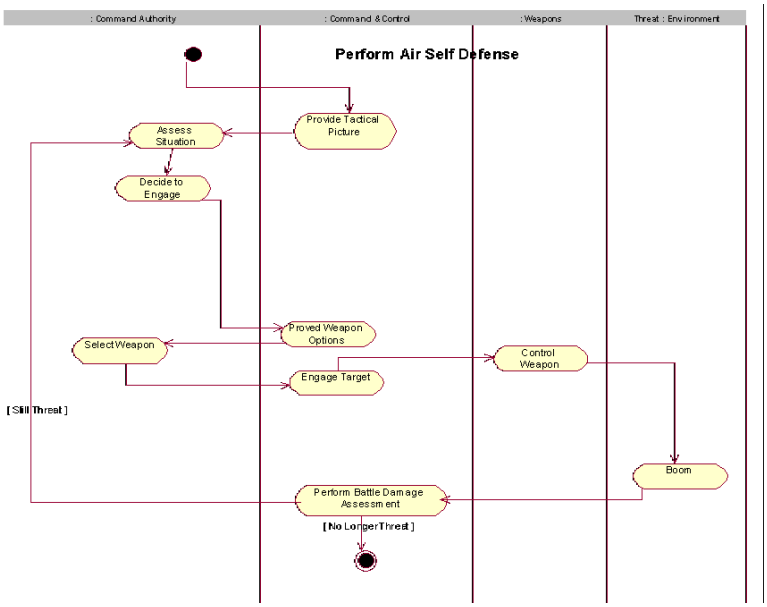

Figure 5 – Activity Diagram

In our example, create an activity diagram for each use case showing the interaction of the System-of-System systems, paying particular attention to the C2 System since this is the system we are building. Since the set of Systems that compose our Enterprise Model are known, the activity diagrams are "white box" diagrams, meaning that a swim lane is added for each system that participates in the use case. Since we will be detailing only the C2 System, the focus is placed on that system. As with the use case survey and DOM, domain experts are a vital source of information for the development of the activity diagrams.

2.5 Add Object Flows to Use Case Activity Diagrams

It was mentioned earlier in this paper that the DOM and the Activity diagrams are developed together and iteratively. As activities are identified in the activity diagrams, the user needs to ask what information is used during this activity. If the class already exists an object instance of that class is added to the activity diagram and placed in the swim lane that owns the information.

Since in our example problem we are only detailing the C2 System, emphasis should be placed on the activities in that swim lane. If the class doesn't exist, a new class is added to the DOM.

Figure 6 is an example of an Activity diagram with objects. Flow arrows show when objects are used or modified in association with an activity. Figure 6 represents the same activity diagram in Figure 5 with objects included.

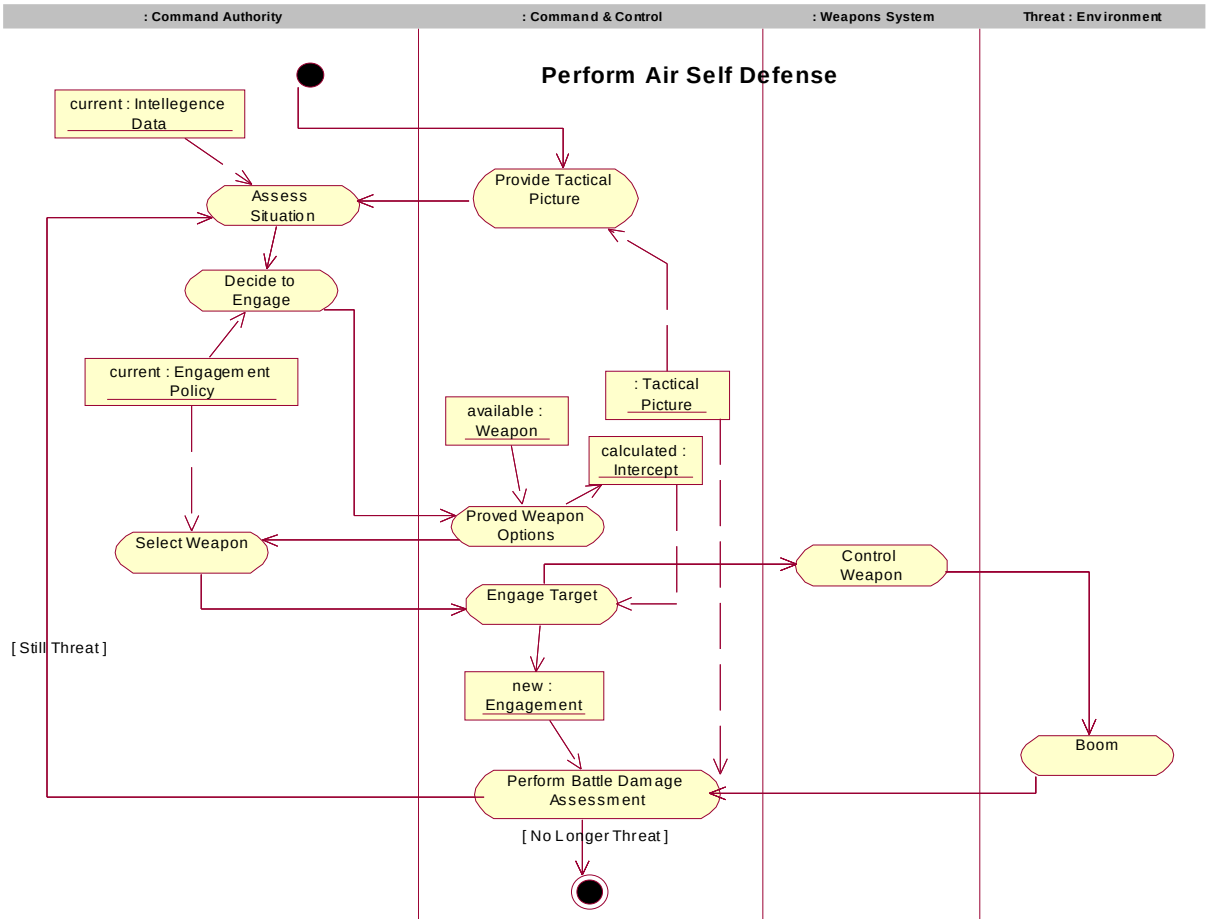

Figure 6- Activity Diagram with Objects

3.0 Develop Command and Control System Use Case and DOM

At this point in our example we have detailed the Enterprise (System-of-System) level use cases that are germane to the Command & Control System and are ready to begin to extract a context diagram, use case survey and domain object model survey for the C2 System. The first step is to produce one or more sequence diagrams for each of the enterprise use cases. In the process methods will be added to a subset of the DOM classes and some of these will constitute the C2 System Class Survey. The Sequence diagrams will be studied to extract a use case survey for C2 System and the actors for these use cases determine the C2 System context diagram.

3.1 Sequence Diagrams

At least one sequence diagram should be created for each use case. The number of sequence diagrams per use case will increase with the complexity of the use case scenarios. In the sequence diagram, the modeler defines the collaboration between the Command and Control domain objects located in the swim lane of the system of interest and other systems and actors, to satisfy the activities associated with a particular use case scenario. As the operations are defined, they are assigned to the appropriate domain class, proxy or actor. A careful description of the actions performed by each method should be written, as they are frequently reused in multiple sequence diagrams. The interaction with other subsystems begins to identify the interface dependency of the C2 System. Figure 7 Shows and example of a sequence diagram generated for the Command and Control system based on the activity diagram in Figure 6. In this example, the other Systems (e.g. Weapons) are not detailed and methods are simply added to the proxy class for the System.

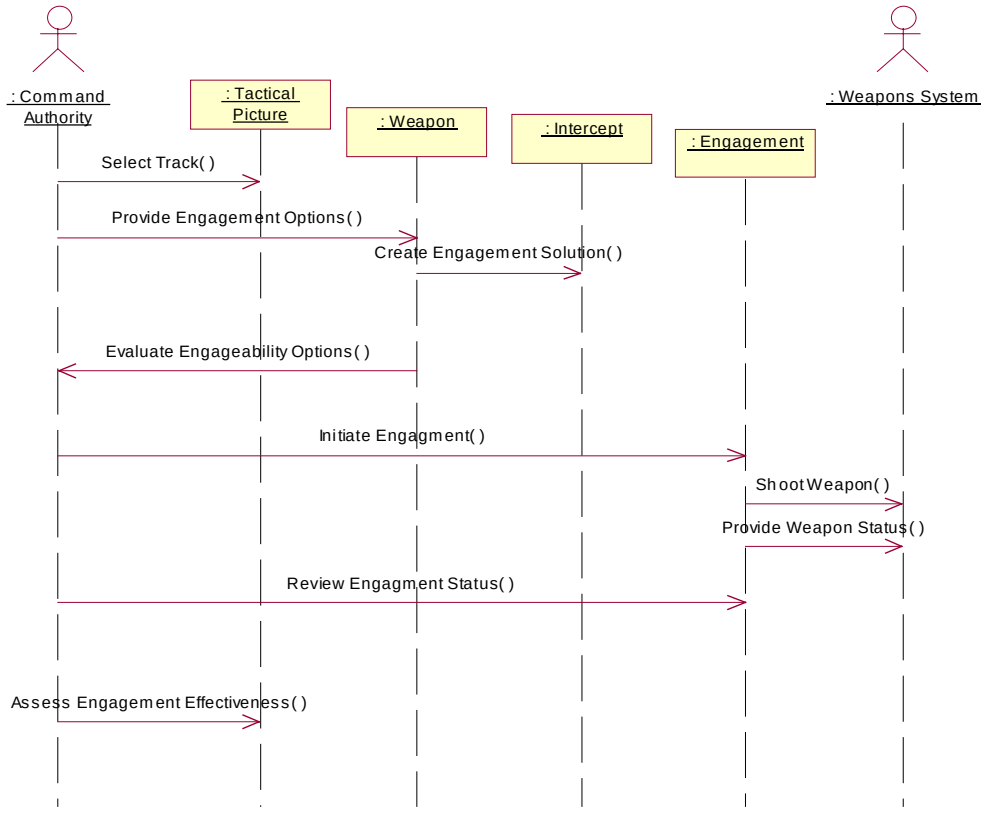

Figure 7- Sequence Diagram

3.2 C2 System Domain Object Model Survey

As the sequence diagrams are developed methods are added to DOM classes, which have already been partitioned by System. Since our emphasis is on the C2 System, most of our DOM classes are associated with it, but every System has at least one class, namely a proxy class. As noted above, for all Systems except the C2 Systems the methods are added to a System proxy class. But for the C2 System, methods are added to classes associated with the C2 System. It is important to note that not necessarily all classes will have methods. Those classes that have methods and have been allocated to the C2 System constitute the C2 System DOM Survey. Since each operation has been precisely defined, these operations define the functionality performed by the C2 system. Figure 8 is an example of this survey.

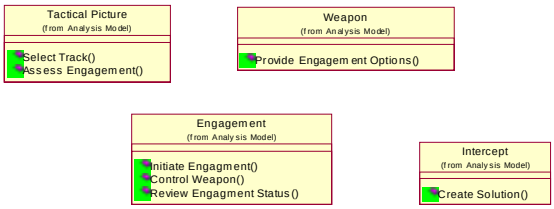

Figure 8 – C2 System DOM Class Survey

3.3 C2 System Use Case Survey

The sequence diagrams not only serve as a vehicle to define the System functionality via their methods, but also serve as a tool for identifying System use cases. The sequence diagram is composed of objects associated with the C2 System and proxy objects for the other systems. From the perspective of the C2 system, any proxy object used in a sequence diagram is an external actor. Thus the sequence diagrams show interactions between the C2 System and its external actors. Each coherent set of interactions becomes a use case, and the specific sub-sequence diagram that encompasses the interactions is used as the basis for the use case survey description.

There will frequently be a many to many relationship between Enterprise Level Use cases and the use cases in the System Use Case Survey. Figure 9 is an example of a Use Case Survey for our C2 system.

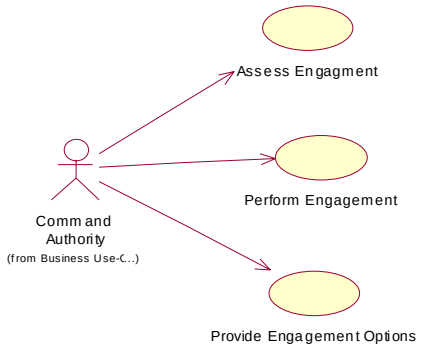

Figure 9 – Use Case Survey

3.4 C2 System Context

After the use case and class surveys are developed the context diagram for the C2 system should also be developed. This is done by reviewing the set of actors from the use case survey and will consist of some or all of the Enterprise Actors and some or all of the other Enterprise Systems. Figure 10 show the context diagram for our sample C2 system.

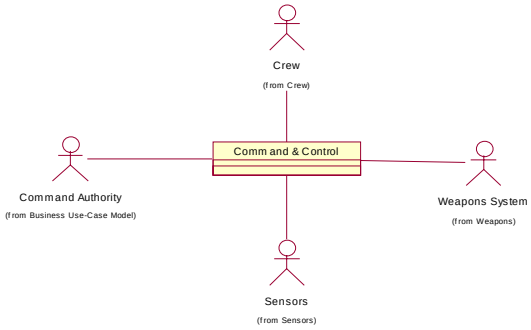

Figure 10 – C2 Context Diagram

4.0 Transitioning to the Next Level

At this point, the following artifacts in the enterprise model have been developed for the C2 System:

- A C2 Context diagram
- A C2 Use case survey
- A C2 Class survey

These artifacts provide the framework for our next level model – the C2 system model. In fact, they serve as a specification for the C2 System. The methods on the DOM class survey specify the functionality of the C2 System. Depending on the project objectives, "shalls" may even have been used to specify each method's functionality. The C2 Use Case Survey specifies how the C2 System is used and can serve as a test scenario specification. The C2 Context Diagram specifies the external interfaces for the System. The sub-sequence diagrams associated with each use case in the use case survey and the operations added to the external actors (System Proxies) add further detail to the external interface definition.

At this point the same process repeats with the goal of discovering the subsystems of the C2 System. The primary distinction from what has been discussed so far is how the new subsystems are identified, since the Systems were assumed to be a given in our Enterprise model. In the work so far, we exploited our knowledge the Systems by producing "white box" activity diagrams, which means we had a separate swim lane for each relevant System. This allowed us to focus the activity diagram enhancements on the C2 swim lane and to use System proxies for all Systems except C2 in the sequence diagrams. As DOM classes were identified, they were assigned to the System (or External) package that owned them. In the case where the Subsystem are not known apriori, all the activity diagrams are necessarily "black box", which means we will only have a swim lane labeled "C2 System". This swim lane is still detailed with object flows and sequence diagrams are produced as before. The Subsystems are defined by looking for cohesive collections of classes having methods and grouping them together to form a class survey for each subsystem. The functionality associated with those classes through their operations specifies the functionality provided by that subsystem. The goal of the decomposition is to get a set of loosely coupled subsystems with little or no redundant functionality.

5.0 Conclusion

This paper has presented an object oriented systems engineering approach to the problem of System-of-System specification and decomposition. A user centric process for detailing the functional requirements for a system and extracting a specification for subsystems of the system has been presented. The approach is model based and uses a variety of UML diagrams including context, use case, class, activity and sequence diagrams.

The model based, graphical nature of the approach brings with it several benefits. Graphical models facilitate requirements elicitation, review and verification by the System users, which has been referenced as a key part of the process. A well organized model framework facilitates the addition of new capability or modification of existing capability. The hierarchical and graphical nature of the models provides an excellent means for new project members to explore and learn the details of the System-of-Systems. Current modeling tools allow teams of engineers to work collaboratively on the System-of-System models and provide linkage between the models.

Modeling of Systems-of-Systems still requires traditional systems engineering activities to occur. For example, performance models and timing budgets still need to be developed using traditional systems engineering tools and methods such as simulation models. It is our view that this approach is complimentary to these vital activities. As data interchange standards such as AP-233 are adopted, it is expected that the boundaries between the techniques discussed in this paper and other systems engineering disciplines and tools will become less pronounced.

The diagrams used in the approach are all consistent with the UML 1.4 standard. That standard includes additional diagrams, such as deployment diagrams and state diagrams, which were not included in this paper. The UML 2.0 standard is in the works and is expected to be released in 2004. That standard will include extensions specifically targeted for Systems Engineering and it is our understanding that these extensions will only enhance the utility of the approach discussed here.

The approach presented in this paper focuses on the behavioral aspects of a System-of-System. There are also non-behavior requirements (e.g. "ilities") that are a critical

part of the System definition. The authors are currently exploring techniques for representing these non-behavioral aspects of the system in UML and for allocating them to subsystems.

References

Bahill, Terry and Daniels, Jesse, Using Objected-Oriented and UML tools for Hardware Design: A Case Study, Systems Engineering (6) (2003) 28-48.

Cantor, Murray, Rational Unified Process for Systems Engineering RUP SE1.0, A Rational Software White Paper TP 165, (8/01).

Friedenthal, Sanford A., Object-Oriented System Engineering, Proceedings of the Lockheed-Martin Systems Engineering and Software Symposium, (July 1997).

Rob Cloutier works for Lockheed-Martin Corporation[1]. He holds a B.S. from the United States Naval Academy, and an M.B.A. from Eastern University. He has 20 years experience in systems engineering, software engineering, and project management in both the commercial and DoD industries. Rob can be reached at: robert.j.cloutier@lmco.com.

Andrew Winkler works for Lockheed-Martin Corporation[1]. He holds a B.S. and M.S. in Physics from the University of Vermont. He has over 8 years experience in systems, software and manufacturing engineering. Andrew can be reached at: andrew.j.winkler@lmco.com

John Watson works for WOW-IS and currently is providing contract and consulting work to Lockheed Martin[1]. He has over twenty years of software and systems engineering experience in a large spectrum of telecommunication systems and navy command and control domains. John can be reached at: john.watson@lmco.com

Clay Fickle works for the Lockheed-Martin[1] Advanced Technology Laboratories. He has over twenty years of software and systems engineering experience in airborne radar, communication satellite and navy command and control domains. Clay can be reached at: clay.fickle@lmco.com

[1] Lockheed-Martin Corporation, NESS&SS, 199 Borton Landing Road, Moorestown, NJ 08057