

410-TD-003-001

Object Modeling Technique Tutorial for the ECS Project

August 1995

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

<u>Paul Fingerman /s/</u>	<u>8/3/95</u>
Paul Fingerman, Systems Engineer EOSDIS Core System Project	Date

SUBMITTED BY

<u>Edward Lerner /s/</u>	<u>8/3/95</u>
Edward Lerner, Science and Communications Development Office (SCDO) Manager EOSDIS Core System Project	Date

Hughes Information Technology Corporation
Landover, Maryland

This page intentionally left blank.

Abstract

Object oriented methodology is a development paradigm that organizes a system as a collection of objects, each of which has data structure and behavior, and which has meaning within the context of the problem that is being modeled. This document provides a brief introduction into the Object Modeling Technique tool as used in the CSMS/SDPS design specifications.

Keywords: object, modeling, techniques, OMT, CDR, class, attribute, review, guide, event, trace

This page intentionally left blank.

Contents

Abstract

Contents

1. Introduction

1.1	Purpose	1-1
1.2	Organization	1-1

2. Object Modeling Technique Tutorial

2.1	OMT	2-1
2.2	OMT Diagram Tutorial	2-2
2.3	Event Trace Diagram Tutorial	2-4
2.4	State Transition Diagram Tutorial	2-4

Figures

2-1	Object Model Diagram Notation.....	2-2
2-2	Example of Object Model Diagram	2-3
2-3	Example of Event Trace	2-5
3-4	Example of State Transition Diagram.....	2-6

Acronyms and Abbreviations

This page intentionally left blank.

1. Introduction

1.1 Purpose

This tutorial is provided to personnel participating in the Critical Design Review (CDR) of the EOSDIS Core System (ECS). This document provides the reviewer with the background necessary to interpret Object Modeling Technique (OMT) figures used frequently throughout the design data package.

1.2 Organization

This paper is organized as follows:

Section 1 provides the reader with pertinent information regarding the purpose and organization of the document.

Section 2 provides the tutorial, including sample figures and explanatory text by which the user may become familiar with the tool and its output.

Questions concerning distribution or control of this document should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Corporation
1616 McCormick Drive
Landover, MD 20785

This page intentionally left blank.

2. Object Modeling Technique Tutorial

2.1 OMT

Object-oriented methodology is a development paradigm that organizes a system as a collection of objects, each of which has data structure and behavior and which has meaning within the context of the problem that is being modeled. The methodology being used on the ECS Program is the Object Modeling Technique (OMT) set forth by Rumbaugh, et al in the book Object-Oriented Modeling and Design. In object-oriented methodologies, the analysis and design are developed in terms of graphical models. The foundation of OMT is the object model, in which the complete static structure of the system is captured.

The following material provides a tutorial on how to read an OMT object model. The tutorial is in the form of a walk-through of a sample model. Although the sample does not use all of the available notation, it uses most of the notation that will be seen in models that have been constructed for ECS. Before starting the walk-through, the following definitions have to be understood.

- **Object:** An abstraction of something in the problem at hand, characterized by a unique name, distinct properties, and well defined behavior.
- **Class:** A group of objects with the same meaning, properties (*attributes*), behaviors (*operations*), and relationships (*associations*) with other objects.
- **Generalization:** Objects can be generalized into a more generic object class. For example, guides, program descriptions, and general system descriptions could be generalized into a common class called documents. The document class is then called the parent class of guides, program descriptions, and general system descriptions.
- **Attribute:** a named property of a class, describing data values held by each object in the class. Classes describe the data property (e.g., color). Each object holds a value (e.g., green) for each attribute defined for the class to which the object belongs.
- **Operation:** a part of the behavior of a class. Collectively, all of a class' operations define the things that objects of the class can do.
- **Link:** a physical or conceptual connection between object instances -- an instance of an *association* (see the next definition).
- **Association:** a group of links with common structure and common meaning -- a set of potential links.
- **Aggregation:** The model also recognizes a specific kind of relationship, called Aggregation. It indicates that objects of one class (the aggregate) are composed of objects belonging to other classes (the components).

The following design document uses several types of modeling diagrams:

- Object Model Diagrams depict the classes of objects which make up a design, their attributes and operations, and how they are related to each other. In essence, the concepts presented in the above list are presented diagrammatically (see Section 2.2).
- Event Trace Diagrams depict a sequence of events that occur in a scenario. At the preliminary design level, scenarios are concerned with the interactions that take place among objects. Events, therefore, represent messages which are sent from one object to another (see Section 2.3).
- State Transition Diagrams are used occasionally in the design to show how the messages affect the internal state of an object, and in particular, the events which cause state transitions (see Section 2.4).

2.2 OMT Diagram Tutorial

Figure 2.2-1 shows the notation used by the ECS Object Models. The rectangular boxes in the model denote classes. Each box, shown in full detail, consists of three sections. The name of the class fills the top section, its attributes go in the middle section, and its operations in the bottom section. Sometimes in high level drawings, only the top section of the box, showing the class name, is shown. A class may be the generalization of several other classes. In Figure 2.2-1, the "Parent Class" is the generalization of two other classes, each called a "Derived Class." Derived classes always include the attributes and operations provided by their parent classes. The diagrams, therefore, only show any additional attributes or operations which the derived class may have.

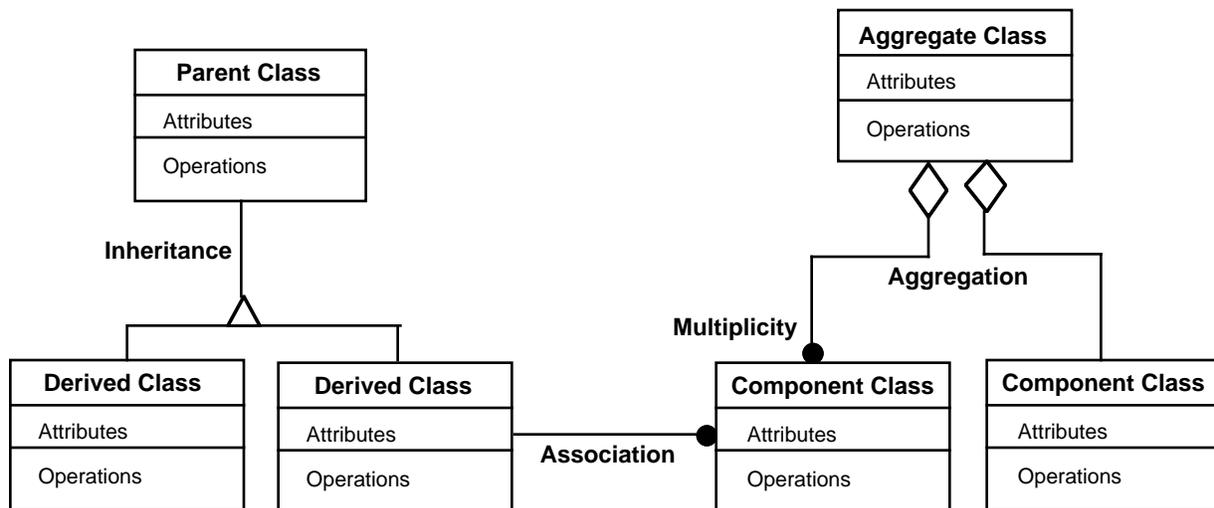


Figure 2.2-1. Object Model Diagram Notation

Figure 2.2-1 also shows that there are two classes, each called a "Component Class", have been aggregated into another class, called the "Aggregate Class". There may be design rules which determine how many components of each class an aggregate may have. This is shown by providing an indication of the "Multiplicity" in the diagram. In Figure 2.2-1, the left component

The diagram shows several types of resources, each a derived class. For example, strings (represented by "PIString") may be associated with several computers and several network disks. A computing platform ("PIComputer") consists of several components, namely several cpu ("PICpu"), disks ("PILocalDisk") and main memory ("PIRam"). It may also use several network disks ("PINetworkDisk"), and those disks might be attached to several computers.

2.3 Event Trace Diagram Tutorial

Figure 2.3-1 shows an example of a event trace diagram, again taken from the Production Planning CSCI. The example shows a scenario in which a previously created plan is activated (the example has been chosen for its simplicity). The first even occurs in the user interface, when production planning staff selects a previously created plan and activates it. Correspondingly, the diagram shows an arrow from the Planning User Interface to the Plan.

The software associated with the Plan object now will perform a number of operations, such as verifying that the plan being activated is indeed valid, noting the differences between the currently active plan and the new one, and creating new processing requests (or updating existing ones) to implement the new plan. The result of this activity is a series of messages to the "DPR" class. Each instance of this class represents a pending job and has associated with it any information which must be provided as input to the processing CSCI when the PGE is initiated through a request for data processing (hence the name of the class). Each message creates a new instance of a pending job, or updates a currently existing pending job.

Concurrently, the planning CSCI monitors the inputs for which there are currently waiting jobs (i.e., for which there is processing in the currently active plan). In essence, each of the pending jobs waits for its inputs to become available. When they are, the pending job (i.e., the corresponding instance of "DPR") is sent to the Processing CSCI. In the object design, interfaces with external CSCI are typically represented as "Interface Classes". The "Processing Planning Interface" is such an interface class. In the Planning CSCI object model, it is the target of the DPR message.

2.4 State Transition Diagram Tutorial

Figure 2.4-1 shows an example of a state transition diagram. The rectangular box used to depict object classes this time is used to show the possible states of the object. In the case of a production plan ("PIPlan") there are two possible states:

- The plan can be a "candidate plan", i.e., one that has been fully planned but has not been activated. A candidate plan is created with the "create plan" command. This is the initial state of a plan. This is indicated in the diagram by showing that there is no previous state (i.e., with a filled black circle).

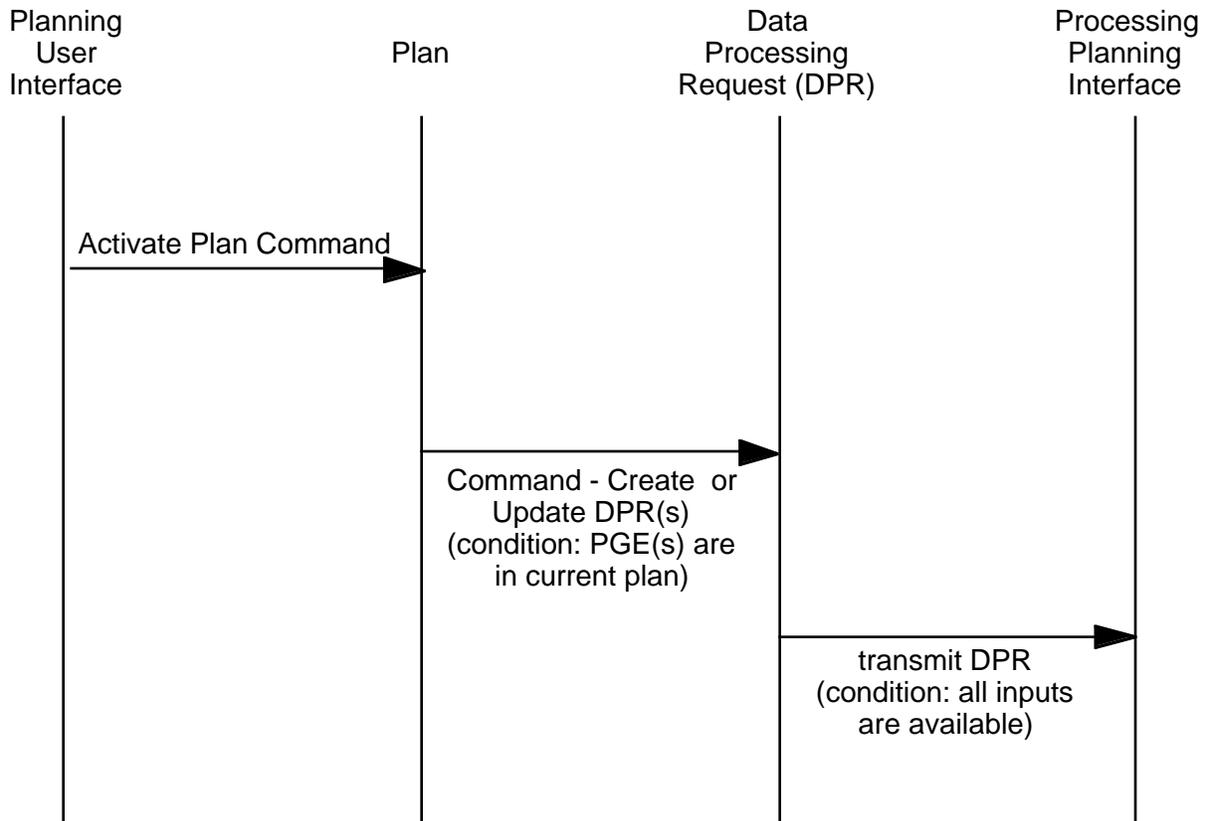


Figure 2.3-1. Example of an Event Trace Diagram

- A candidate plan can be made an "active plan" via an "activate plan" command. In this state, the plan receives alerts for on-demand production requests and data arrival notifications (DAN). It instructs DPR objects to transmit the corresponding data processing request message when the job is ready (as discussed in the scenario in Section 2.3). This is the final state of a plan. The plan is terminated when another plan is activated, or when this plan is canceled. The final state of an object is indicated by the symbol shown to the left of Activate Plan.

The diagram shows that during the creation of the candidate plan, the plan object interacts with a number of other objects: with "Data Processing Requests" to obtain information about the PGE in the plan; with "Resource Management" to determine the availability of resources needed by those PGE, and with the data server (represented by the "Data Server Interface" class) to store the candidate plan. Successful creation of the plan (or failure) also display a response on the planning user interface.

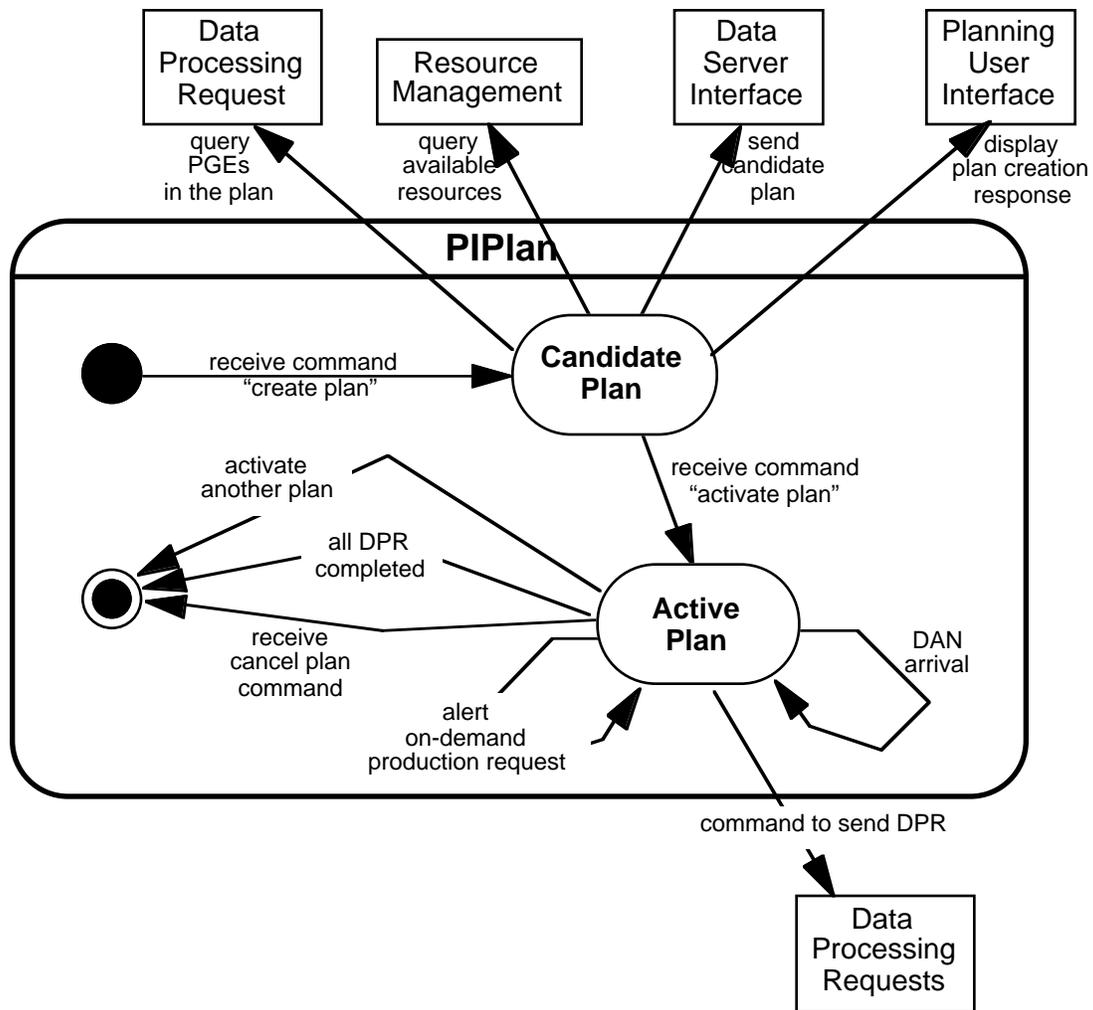


Figure 2.4-1. Example of a State Transition Diagram

Acronyms and Abbreviations

CDR	Critical Design Review
CSCI	computer software configuration item
ECS	EOSDIS Core System
OMT	Object ModelingTechnique

This page intentionally left blank.