1992

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER

THE UNIVERSITY OF ALABAMA

GUIDE TO OBJECT-ORIENTED ANALYSIS AND DESIGN

Prepared by: Harry C. Harrison

Academic Rank: Associate Professor

Institution and Department: Capitol College
Department of Mathematics and Computer Engineering

NASA/MSFC:

Office: Information Systems Office
Division: Systems Development and Implementation Office
Branch: Data Systems Branch

MSFC Colleague: Marcellus Graham

Introduction

Purpose

The purpose of this guide is to provide Marshall Space Flight Center personnel with guidelines for the use of object-oriented analysis and design and to describe how it can be accomplished within the framework of existing development directives, including the **Software Development Plan**. It is not intended as a detailed tutorial. The reader is referred to the Coad and Yourdon texts[1,2] in the References.

Overview

The term **object-oriented** has become a popular buzz word in the computer world today. Many say that it will be the dominant form of software development in the future. It is characterized by reusability of code, resulting in faster development and significantly lower development costs. In today's budget constrained world, it is critical to use the most cost-effective tool to develop software. NASA should be at the leading edge of this development.

While the object-oriented approach has great potential, it has been misused. Some vendors use this paradigm:

Object-oriented is good.
My Product is object-oriented.
Therefore my product is good.

Additionally, object-oriented analysis and design is still young. Standards are still not firm.

The object-oriented approach is not a panacea. It is not always the best choice. And when it is selected, the transition maybe difficult for many organizations. The object-oriented approach will require a radically new way of thinking about software development. Analysts will have to deal with a whole new language involving classes, inheritance, encapsulation, polymorphism, virtual functions, overloading, and a whole new model of programming.

Simplistically, in the object-oriented approach, the world is modeled in terms of objects that pass messages back and forth. The user does not need to know the details of how an object implements a message(data abstraction), but only what goes in and what comes back. He or she simply creates objects from a class and passes them messages.

The object-oriented anlysis and design approach presented here is based on the Yourdon/Coad textbooks[1] and [2]. Their approach appears to be the most widely accepted and is very clearly explained in these texts. These texts are highly recommended to any developer who plans to utilize OOA/OOD.

Terminology

attribute - a characteristic of an object, such as a name, size, part number, etc.

class - An abstract description of the data and behavior of a collection of similar objects.

data abstraction - defining high-level data types that provide a complete description of the system without a great amount of detail.

encapsulation - the tying together of data and functions into a single entity.

inheritance - the property of objects that gives them access to data and functions contained in a previously defined class.

polymorphism - literally, "many shapes". The exact way of implementing a service or function depends on the class that an object is in. You may have the same function name implemented differently in different classes.

service - a behavior that an object performs.

The Object-Oriented Approach

The object-oriented approach began with the programming language Simula. It then spread with other languages such as SmallTalk, and C++. The need for analysis and design approaches to support these languages led to object-oriented analysis(OOA), object-oriented design(OOD), and object-oriented programming(OOP). A language that supports object-oriented programming is known as an OOL. OOD is very closely related to OOA. OOD basically refines the model that was built in the OOA analysis. Both can be done without necessarily building the system with a language that supports object-oriented programming(OOP). There are still benefits to be gained from such an approach.

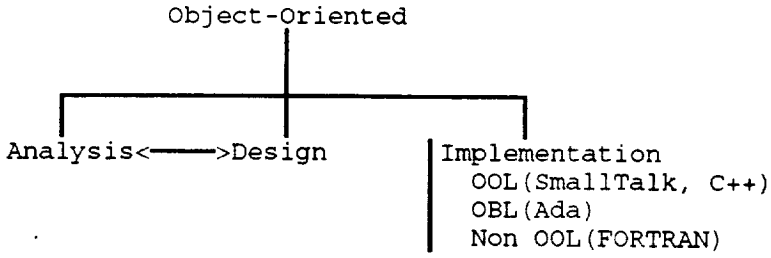

Figure 1 The Object-Oriented Approach

Object Diagrams

The heart of OOA/OOD is the object diagram. It consists of objects(rectangles with the object/class name, attributes, and services it provides listed inside. These objects have inheritance relations with classes, component relations when they are part of a larger system, and instance relationship when a certain number of the these objects are required for a system to perform. An object can pass message to other objects. These are the arrows. Objects can be grouped into subject areas to make the system easier to understand. Below is an object diagram that illustrates all of the symbols used.

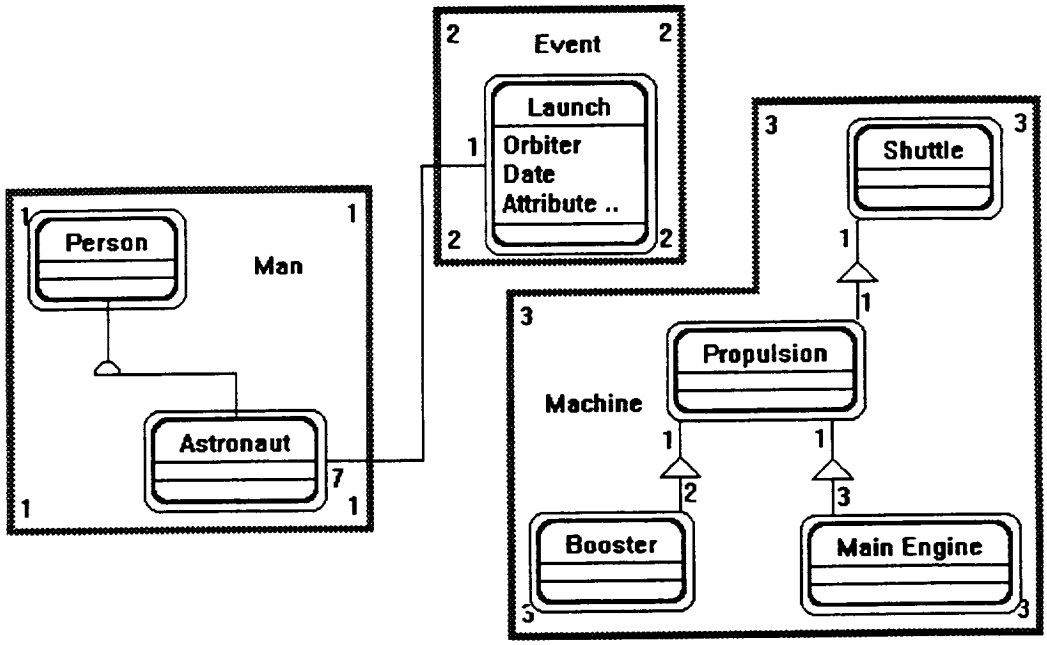

Figure 2 Object Diagram

The subject areas are Man, Machine, and Event. Inside Man is the class Person. A specialization of Person is Astronaut. The Astronaut class inherits information from person. Only the unique aspects of Astronaut need to be described, due to inheritance. There are 7 Astronauts required for a Shuttle mission. This is shown by an instance line. An event consists of a Shuttle Launch. A launch has an orbiter, a launch date, an STS number, etc. associated with it. The propulsion system consists of 3 main engines, 2 boosters, and a refueling tank. This is the component relationship.

Guidance for MFSC Software does not specify any particular development methodology. This report shows how charts such as the object diagram can be used in the development process.

Conclusions

Object-oriented analysis and design enable the developer to model the problem using objects, classes, attributes, and functions as the components. The combining of data and processing into a single entity(data encapsulation) protects the data and aids in making the software more reusable.

This approach is based on a model that is closer to reality. This facilitates communications with the user.

Although the approach is different from the classical waterfall approach used with structured techniques, the process can be adapted to fit within the guidelines of the SDP.

References

1. Peter Coad and Edward Yourdon, Object-Oriented Analysis, Yourdon Press, 1991.

2. Peter Coad and Edward Yourdon, Object-Oriented Design, Yourdon Press, 1991.