UNIT 2

Hierarchical Object Oriented Design (HOOD)& Object Modeling Technique (OMT)

HOOD

Hierarchical Object-Oriented Design (HOOD) has been developed for the European Space Agency as a design/notation method for Ada. Robinson introduces it in this way: "The top-down hierarchical decomposition approach is not new. After all, this is the method used with data flow diagrams starting from a context diagram which shows all the external interfaces with one central process. This process is then decomposed into other processes with data flows and control flows interacting between them, with consistency checks between levels. In the same way, the purpose of HOOD is to develop the design as a set of objects which together provide the functionality of the program."

The main process in HOOD is called the Basic Design Step. Robinson describes it admirably: "A Basic Design Step has as its goal the identification of child objects of a given parent object, and of their individual relationships to other existing objects, or the refinement of a terminal object to the level of the code. This process is based on the identification of objects by means of object-oriented design techniques."

The phases can be summarised as follows:

1. Problem definition.

The context of the object to be designed is stated, with the goal of organising and structuring the data from the requirement analysis phase. This is an opportunity to provide a completeness check on requirements and traceability to design.

1.1 Statement of the problem - the designer states the problem in correct sentences which provides:

- a clear and precise definition of the problem;
- the context of the system to design.

1.2 Analysis and structuring of requirement data - the designer gathers and analyses all the information relevant to the problem, including the environment of the system to be designed.

2. Development of solution strategy.

The outline solution of the problem stated above is described in terms of objects at a high level of abstraction.

3. Formalisation of the strategy.

The objects and their associated operations are defined. A HOOD diagram of the proposed design solution is produced, allowing easy visualisation of the concepts and further formalisation. There are five subphases in the formalisation of the strategy:

3.1 Object identification.
3.2 Operation identification.
3.3 Grouping objects and operations (object operation table).
3.4 Graphical description.
3.5 Justification of design decisions.

4. Formalisation of the solution.

The solution is formalised through:

- formal definition of provided object interfaces
- formal description of object and operation control structures.

1.3.2 Notation

The main diagram used for describing the structure of a system is the HOOD object diagram, which shows a static view of the structure in the hierarchical object oriented design. The symbols used are as follows:

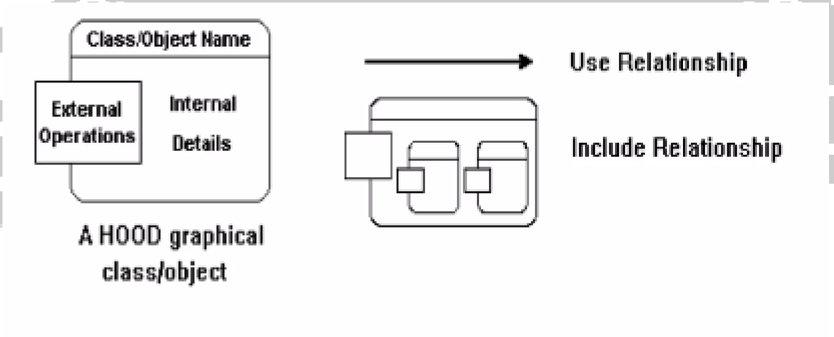

Figure - A Key to the HOOD Diagram notation

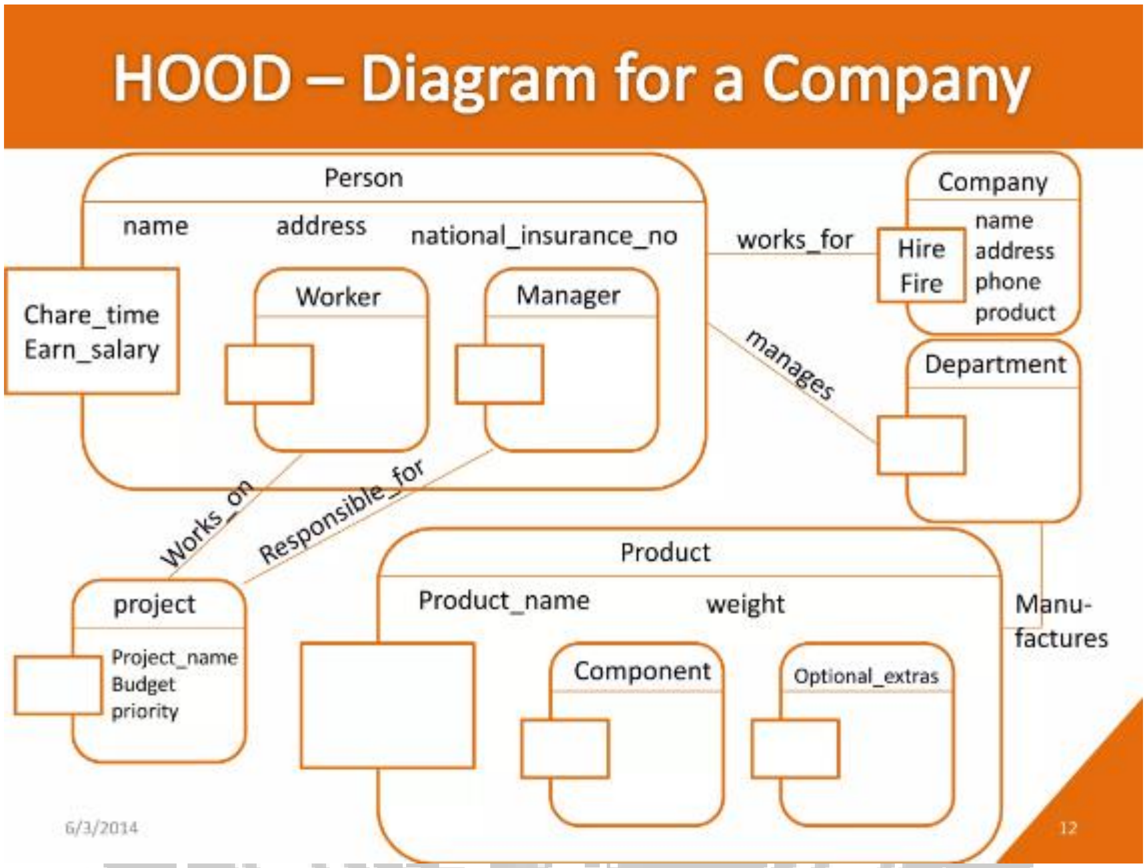

Figure - A HOOD Diagram for a Company

1.3.3 Analysis of Method

HOOD is heavily geared towards an implementation in Ada. This is obviously ideal for Ada developers, but can be considered to limit its usefulness for any other programming language environment. Jacobson et al. [Jaco92] say: "The method gives the basic design step, but does not give any help in finding the appropriate object structure. Actually HOOD does give strong support for containment structures, but does not for other structures such as use or inheritance structures."

One of the major criticisms of HOOD is its lack of support for some of the more object-oriented techniques available. Support for inheritance is very poor, as evidenced by the lack of any notation to graphically represent it. The method is perhaps more object based than truly object- oriented.

1.3.4 Support for reuse

HOOD has no specific consideration of reuse in its process. Robinson [Robi92c] suggests that the HOOD principles of abstraction, information hiding, locality and modularity support reusability, because of

" - applicability to the real world and to data entities,
- an object may be general and therefore be reused in a new configuration".

The only suggestion Robinson gives of designing with reuse is a very vague reference when describing how to produce class objects. "A class object has to be designed separately from the main design as a root object, and either:

- pre-exists from a previous project
- from a library of classes
- has been developed to represent an Ada generic package
- it is developed specially for the project as it is seen as a general purpose object that can be usefully used in several parts of the design. This is a bottom up aspect of design."

Tools available

Name of Tool Company Price Platform HOOD-SF Virtual Software Factory Ltd. Unknown Unknown Select Select Software Tools Unknown Unknown GraphTalk Rank Xerox Unknown Unknown HOOD Toolset IPSYS Software Unknown UNIX, DOS STOOD Techniques Nouvells d'Informatique Unknown UNIX, RISC, X-windows

The Object Modelling Technique (OMT)

1.4.1 The Method

The Object Modelling Technique provides three sets of concepts which provide three different views of the system. There is a method which leads to three models of the system corresponding to these views. The models are initially defined, then refined as the phases of the method progress. The three models are:

The object model, which describes the static structure of the objects in a system and their relationships. The main concepts are:

- class

- attribute
- operation
- inheritance
- association (i.e. relationship)
- aggregation

The dynamic model, which describes the aspects of the system that change over time. This model is used to specify and implement the control aspects of a system. The main concepts are:

- state
- sub/super state
- event
- action
- activity

The functional model, which describes the data value transformations within a system. The main concepts are:

- process
- data store
- data flow
- control flow
- actor (source/sink)

The method is divided into four phases, which are stages of the development process:

1) Analysis - the building of a model of the real world situation, based on a statement of the problem or user requirements. The deliverables of the analysis stage are:

- Problem Statement
- Object Model = Object Model Diagram + data dictionary
- Dynamic Model = State Diagrams + Global Event Flow Diagram
- Functional Model = Data Flow Diagram + constraints

2) System design - the partitioning of the target system into subsystems, based on a combination of knowledge of the problem domain and the proposed architecture of the target system (solution domain). The deliverables of the system design stage are:

- System Design Document: basic system architecture and high-level strategic decisions

3) Object design - construction of a design, based on the analysis model enriched with

implementation detail, including the computer domain infrastructure classes. The deliverables of the object design stage are:

- Detailed Object Model
- Detailed Dynamic Model
- Detailed Functional Model

4) Implementation - translation of the design into a particular language or hardware instantiation, with particular emphasis on traceability and retaining flexibility and extensibility.

1.4.2 Notation

The main diagram used for describing the structure of a system is the object model, which gives a model of the class structure specified in the object oriented design. These are the symbols that are used:

1.4.3 Analysis of Method

The OMT is perhaps one of the most developed of the object-oriented design methods, both in terms of the notation that it uses, and the processes which are recommended for developing an object oriented system. In describing the technique, Rumbaugh et al. [Rumb91] claim that the method attempts to show how to use object-oriented concepts throughout the whole of the software development lifecycle. The inclusion of sections discussing analysis, design, and implementation in both object-oriented and non-object-oriented languages help to validate this claim.

The concept of using three views to represent a system presented in the OMT is potentially very powerful, but can also be considered to be very complex. It is unclear whether the views are to be developed independently of each other, or whether knowledge of one model should be used to influence construction of another. It would seem logical to ensure that the concepts in the different views are properly defined and interrelated, or much confusion could occur at the design stage of the development where the models are to be integrated.

Walker [Walk92] notes two main areas in which the OMT method is lacking. First, "there is no serious attempt to address the systematic reuse of software or design components. Secondly, the management of the software development process, including the establishment of suitable metrics for the measurement of progress and quality, is almost totally neglected." He feels, however, that, in terms of a technical approach to object-oriented software development, "Rumbaugh et al. have clearly established the current state of the art."

It is interesting to note that Rumbaugh is now advocating the inclusion of Jacobson's use cases (see section 1.8.1) as a 'front-end' to the OMT method [Rumb94]. He says, "We feel that use cases fit naturally on the front end of the published OMT process and supplement the existing user-centered features of the method." He goes on to qualify his recommendation of use cases, feeling that "a combination of direct domain analysis and use cases is an effective approach to starting analysis, rather than depending on use cases alone."

According to a recent survey conducted by the Object Management Group, OMT is currently the most popular of the standardised object oriented development methods (in house methods were the most used, with OMT a close second). One of the reasons for its success may be the variety of tool support available for the method, some of which are included in the table below.

1.4.4 Support for reuse

Rumbaugh et al. consider only the reuse of code, with very little practical advice. "Reuse a module from a previous design if possible, but avoid forcing a fit. Reuse is easiest when part of the problem domain matches a previous problem. If the new problem is similar to a previous problem but different, the original design may have to be extended to encompass both problems. Use your judgment about whether this is better than building a new design."

In addition, Rumbaugh et al. discount the utility of defining classes in isolation for general reuse. "Planning for future reuse takes more foresight and represents an investment. It is unlikely that a class in isolation will be used for multiple projects. Programmers are more likely to reuse carefully thought out subsystems, such as abstract data types, graphics packages, and numerical analysis libraries." Their suggestions on 'style rules for reusability' concentrate on methods, and, on the whole, reiterate the standard practices of structured programming, emphasising such styles as encapsulation with minimal coupling and high cohesion.

Tools available

Name of Tool Company Price Platform SelectOMT Select Software Tools 495 Windows OMTool Martin Marietta $995 Unknown StP/OMT Interactive Development Environments $12,000 AIX