3. The Object Oriented Analysis and Design (OOA/OOD) method from Coad & Yourdon

3.1 Background ideas

Coad & Yourdon mention 7 key motivations and benefits in favor of OOA/OOD instead of using traditional analysis methods.

These motivations and benefits are:

* Tackle more challenging problem domains
* Improve analyst and problem domain expert interaction
* Increase the internal consistency across analysis, design and programming
* Explicitly represent commonality between classes and objects
* Build specifications resilient to change
* Reuse OOA, OOD and OOP results
* Provide a consistent underlying representation for analysis, design and programming

According to Coad & Yourdon the main result of Object Oriented Analysis and Design (OOA/OOD) comes from a reduction of complexity of a problem and the system's responsibilities within it. The OOA/OOD method is based on a number of general principles for managing complexity. According to Coad & Yourdon the Object Oriented approach is the only way to provide all these principles.

The Object Oriented approach is based on a uniform underlying representation (Classes and Objects) which according to Coad & Yourdon leads to a number of major implications:

* No major difference between analysis and design notations
* No 'transition' from analysis to design
* No waterfall model has to be followed, spiral and incremental are also possible
* There are still different skills and strategies needed for analysts and designers
* There is a uniform representation from OOA to OOD to OOP (OO programming)

For Coad & Yourdon an object oriented approach consists of classes, objects, inheritance and communication with messages.

3.2 Approach

The OOA part of the method consists of five major activities:

1. Finding Class & Objects
2. Identifying Structures
3. Identifying Subjects
4. Defining Attributes
5. Defining Services

It is mentioned at several places in the OOA/OOD method that these steps are not sequential steps.

According to these major activities, an OOA model that is built during Analysis consists of five layers:

1. A Subject layer
2. A Class & object layer
3. A Structure layer
4. An Attribute layer
5. A Service layer

The OOD part adds to the five layers four different components that have to be designed for these layers:

1. Human Interaction Component
2. Problem Domain Component
3. Task Management Component
4. Data Management Component

These design steps are also not sequential steps.

3.3 Concepts of OOA/OOD

Object	An abstraction of an entity (tangible or intangible) about which information has to be kept; an encapsulation of Attribute values and their exclusive services.
Class	A description of one or more Objects with a uniform set of Attributes and Services, including a description of how to create new objects in the class.
Attribute	An attribute is some data (state information) for which each Object in a Class has its own value.
Class-&-Object	A term meaning "a Class and the Objects in that Class"
Subject	Subjects are a mechanism for partitioning large, complex models. Subjects are also helpful for organizing work packages on larger projects, based upon initial OOA *investigations*.
Service	A Service is a specific behaviour that an object is responsible to exhibit.
State	The State of an Object is the value of its Attributes
Transition	Transition is the change of a state
Condition	If/precondition/trigger/terminate action
Text Block	Text
Loop	While/Do/Repeat/trigger/terminate action

Relationships between Objects

* Whole-Part Structures

  In Whole-Part relationships one object (the object representing the Whole) is decomposed of other objects (the Parts). There can be three variations of the whole-part relationship:

  1. Assembly-Parts: A car has wheels, an engine,...
  2. Container-Contents: A box consists of a number of nails
  3. Collection-Members: An organization has a number of analysts

  Whole-Part relationships may be associated with specific amounts or ranges, like the cardinality concept in ER modelling.

* Instance connections (Association relationship)

  An instance connection is a connection that one object needs with other objects in order to fulfill its responsibilities.

  For example: An object Person works at an object Office

  For Instance Connections an amount or range can also be denoted.

* Message Connections

  A message connection models the processing dependency of an Object, indicating a need for Services in order to fulfill its own responsibilities (services).

  For example: For fulfilling the responsibility (service) "video tape return" the Object Store Assistent needs the Service "acces" from the object Rental.

Relationships between Classes

* Generalization-Specialization (Gen-Spec) Structures (Inheritance relationships)

  Gen-Spec relationships between classes define an inheritance hierarchy for classes that are specializations of other classes. A class may both inherit from one superclass (single inheritance) or from more superclasses *(multiple inheritance)*.

  For example: a Diesel Engine is a specialization of an Engine.

3.4 Operations and communication

The types of services the OOA method provides are:

* Algorithmically simple services like Create, Connect, Access and Release
* Algorithmically complex services:
  * Calculate services to calculate a result from the attribute values of an object.
  * Monitor services to monitor an external system or device.

In the OOA method objects communicate by sending messages. The sender object "sends" a message, that is "received" by the receiver object that takes some action and returns a result to the sender object. In the OOA diagram this communication is denoted by Message Connections.

3.5 Techniques

The result of applying OOA/OOD results in one main OOA diagram consisting of the five layers mentioned in section 3.2:

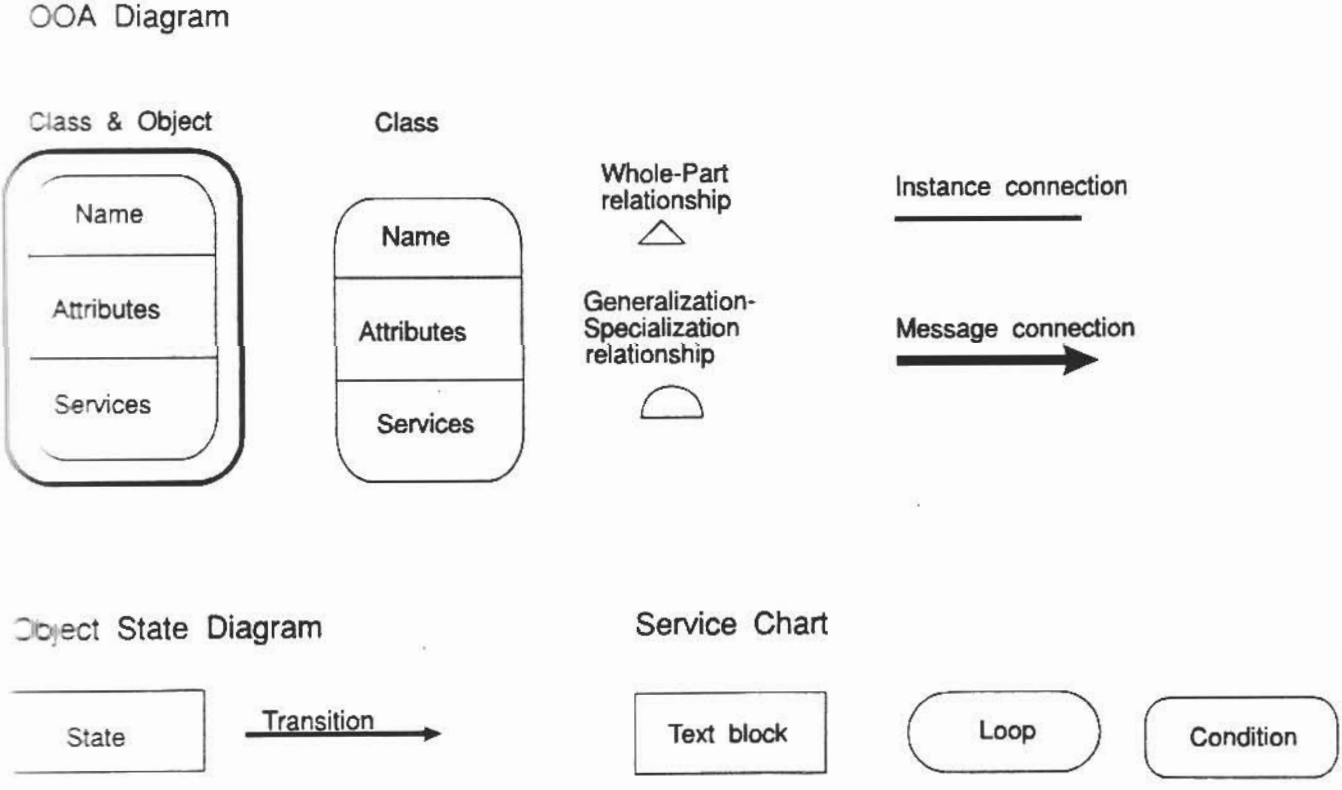
1. Subject layer as a partitioning mechanism
2. Class & Object layer to capture Classes and Objects
3. Structure layer to capture inheritance and whole-part structures
4. Attribute layer to capture attributes and instance connections between Class & Objects
5. Service layer to capture methods and message connections between Class & Objects

The dynamic behavior of Classes can be captured in Object State Diagrams, a restricted form of State Transition Diagrams. The algorithms that have to be applied for services can be described by Service Charts, which are a kind of flowcharts.

The connection between a Service from a Service Chart and the States from an Object State Diagram can be established by a Service/State Table.

Object State Diagrams, Service Charts and Service/State Tables are not described very well or extensively in the OOA/OOD books.

3.6 Graphical notation OOA/OOD



3.7 Description of the OOA/OOD method

For the steps in the OOA/OOD method it is an important point to check previous OOA results in the same and similar problem domains. This is a key point because reuse is one of the main issues in the object oriented approach.

It is also important that the steps described below do not have to be followed in the given sequential order. For large systems it is often better to refine the problem domain into several preliminary subjects, and then start with identifying Class & Objects.

Object Oriented Analysis (Activity 1)

Identifying Class & Objects (Activity 1.1)

The first step in finding Class & Objects is studying the problem domain (activity 1.1.1).

Finding potential Class & Objects (activity 1.1.2) can be done by looking at:

* Structures
* Other systems
* Devices
* Things or events remebered
* Roles played
* Operational procedures
* Sites (Physical locations)
* Organizational units

When potential Class & Objects are found and named (activity 1.1.3) they have to be challenged against the following criteria (activity 1.1.4):

* Needed remembrance
* Needed behaviour
* (Usually) multiple attributes
* (Usually) more than one object in a class
* Always-applicable attributes
* Always-applicable services
* Domain-based requirements
* Not merely derived results

After this is done the Class & Objects can be added to the OOA diagram (activity 1.1.5).

Identifying Structures (Activity 1.2)

There are two kinds of structures: Gen-Spec and Whole-Part structures.

To find Gen-Spec structures (acitivity 1.2.1), each class has to be considered as a Generalization and the next questions have to be asked:

1. Is it in the problem domain?
2. Is it within the problem's responsibilities?
3. Will there be any inheritance?
4. Will the Specialization/Generalization meet the Class & Objects criteria?

Next, each class has to be considered as a Specialization and the above questions have to be asked again.

For identifying Whole-Part structures (activity 1.2.2) the next appearances are possible:

* Assembly-Parts
* Container-contents
* Collection-members

Each object has to be considered as a whole and as a part and the next questions have to be asked:

1. Is it in the problem domain?
2. Is it within the problem's responsibilities?
3. Does it capture more than just a status value?
4. If not: include attribute within the whole
5. Does it provide a useful abstraction?

After identifying Gen-Spec and Whole-Part structures multiple structures can be identified (activity 1.2.3). Multiple structures include various combinations of Gen-Spec, Whole-Part structures or both. After identifying multiple structures the structures can be added to the OOA diagram (activity 1.2.4).

Identifying Subjects (Activity 1.3)

Selecting possible Subjects (activity 1.3.1) is done by promoting the uppermost class in each structure upwards to a Subject. Then promote each Class & Object not in a structure upwards to a Subject.

Refining Subjects (activity 1.3.2) is done by searching for minimal interdependencies and minimal interactions between Class & Objects in different Subjects. Next the Subjects can be constructed (activity 1.3.3) and added to the OOA diagram (activity 1.3.4).

Identifying Attributes (Activity 1.4)

To find attributes (activity 1.4.1) the next questions have to be asked from the perspective of an object:

1. How am I described in general?
2. How am I described in this problem domain?
3. How am I described in the context of this system's responsibilities?
4. What do I need to know?

5. What state information do I have to remember over time?
6. What states can I be in?

In defining Attributes the Gen-Spec structure has to be kept in mind, attributes have to be placed as high as possible in the Inheritance Structure (activity 1.4.2).

After identifying attributes the Instance Connections between objects can be identified (activity 1.4.3).
This is done by adding connection lines for each object reflecting mappings within the problem domain. The Gen-Spec stuctures have to be examined to put the Instance Connections at the right level.

The identified Attributes and Instance Connections have to be checked for several special cases (activity 1.4.4).
These cases are:

* Attributes with a 'non-applicable' value
* Class & Objects with only one Attribute
* Attributes with repeating values
* Many-to-many Instance Connections
* Instance connections between Objects of a single Class
* Multiple Instance Connections between Objects
* Additional needed Istance Connections
* One connecting Object having a special meaning

After checking these cases the Attributes have to be specified, and if needed additional constraints have to be specified for an Attribute and the Attributes and Instance Connections can be added to the OOA diagram (activity 1.4.5).

Identifying Services (Activity 1.5)

The first step in defining Services is identifying the Object States (activity 1.5.1) by describing the States and Transitions in an **Object State Diagram.**
Then the required Services can be identified for each Class & Object (activity 1.5.2). For the algorithmically complex Services like Calculate and Monitor a **Service Chart** can be drawn to depict the algorithmical behavior of the Service. If needed **Service/State Tables** can depict the connection between Services and States.
Simple Services are not denoted in the OOA diagram.

After identifying Services Message Connections can be identified (activity 1.5.3).
For identifying the Message Connections the next questions have to be asked for each object:

1. What other Objects do I need services from?
2. What other Objects needs Services from me?

Then follow each Message Connection and repeat these questions.
Next the Services can be specified and Services and Message Connections can be added to the OOA diagram (activity 1.5.4).

Prepare Documentation (Activity 1.6)

The last step of the OOA part of the method is putting the OOA documentation together.
This activity consists of:

- Drawing of the complete OOA diagram (activity 1.6.1)
- Specification of Class & Objects (activity 1.6.2)
- Addition of supplemental documentation if this is needed (activity 1.6.3) which can consist of:
  - Table of critical threads of execution
  - Additional system constraints
  - Services/State tables or diagrams

Object **Oriented Design (Activity 2)**

Designing the Problem Domain Component (Activity 2.1)

By designing the problem domain component the Object Oriented Analysis is carried into Object Oriented Design. The OOA results are improved, and additions to it are made during designing the problem domain component.
The first step is searching for previous Designs and Classes that can be reused (activity 2.1.1).
To use so called off-the-shelf Classes there may be the need for some changes in the OOA results. If possible the changes have to be kept small.
Next Problem Domain specific Classes can be grouped together (activity 2.1.2) by adding a Root Class as mechanism for grouping Classes. Also Generalization Classes have to be added to establish a protocol, the naming of a common set of Services (activity 2.1.3).
With regard to the programming language that will be used for implementation it can be necessary to change the OOA Gen-Spec structures to accomodate the level of inheritance the programming language offers (activity 2.1.4). Coad & Yourdon give guidelines for moving a multiple Inheritance structure into single Inheritance structures, and how to flatten single Inheritance structures in zero-Inheritance, although this is not recommended, because of the possibilities inheritance offers.
The next step is changing the Problem Domain Component to improve performance aspects (activity 2.1.5). Speed can be improved by measuring the speed of actual code, the perceived speed

of the system can be approved by caching interim results.

Support of the Data Management Component (activity 2.1.6) can be achieved in several ways: each Object can store itself, or it can be saved by the Data Management Component. For both approaches specific Attributes and Services (or new Classes) have to be added to the Problem Domain Component. A third approach is the use of an Object Oriented Database Management System (OODBMS) which takes care of storing Objects.

For convenience during design and programming, lower-level components can be isolated in seperate Classes (activity 2.1.7). The last step of designing the Problem Domain Component is reviewing and challenging the additions that have been made to the OOA results (activity 2.1.8).

Designing the Human Interaction Component (HIC) (Activity 2.2)

For the Human Interaction Component prototyping is recommended.

Designing the Human Interaction Component starts with classifying the people who will be the users of the system by 'putting yourself in their shoes' (activity 2.2.1).

Classification can be done on the next criteria:

* Skill level
* Organizational level
* Membership in different groups (for example staff or customer)

Each defined category of people has to be described, inlcuding a task scenario (activity 2.2.2). The next things have to be written down for each:

* Who is it?
* Purpose
* Characteristics (like age, education etc.)
* Critical success factors (needs/wants and likes/dislikes/biases)
* Skill level
* Task Scenarios

After doing this a command hierarchy can be designed for the system (activity 2.2.3) by studying existing human interaction systems.

To refine this hierarchy the next guidelines have to be considered:

* Ordering of services
* Chunking whole-part patterns and breadth/depth guidelines
* Minimizing number of steps to get a service done

After this is done the detailed interaction can be designed (activity 2.2.4).

Criteria to use are:

* Consistency in human interaction
* Minimizing of number of steps
* Giving meaningfull timmely feedback to users
* Use small steps to accomplish something

* Provide 'Undo' functions
* Don't rely on human memory storage for remembering things
* Keep learning time and effort small
* Pleasure and appeal of system to users is important

To check if the Human Interaction Component meets this criteria there is a need for Prototyping (activity 2.2.5). After the detailed interaction is found to serve the needs, the human interaction Classes have to be designed (activity 2.2.6), including windows, fields, graphics and selectors. Whenever possible, standard graphical user interfaces (GUI) should to be used.

Designing the Task Management Component (Activity 2.3)

The first activity is looking if there is a need for tasks in the system (activity 2.3.1).

The next kinds of systems demand a need for tasks:

* Systems with data acquisition and control responsibility for local devices
* Human interaction with multiple windows that run simultaneously
* Multi-user systems
* Multisubsystem software architectures
* Single processor systems needing tasks to communicate and coordinate
* Multiprocessor systems
* Systems needing communication with other systems

If not needed, tasks should not be designed, because they increase complexity.

There are several kinds of tasks that can be identified (activity 2.3.2):

1. Event-Driven Tasks that trigger upon an event (for example a mouse-click)
2. Clock-Driven Tasks that are triggered on a specified time interval
3. Priority tasks and Critical Tasks

When three or more tasks are needed it is recommended to add a special coordination task.

After defining the tasks these must be challenged against a number of criteria (activity 2.3.3) to keep the number of used tasks at a minimum and to keep the tasks understandable.

After this each task can be defined (activity 2.3.4) by specifying:

* What the task is
* How the task coordinates
* How the task communicates

Designing the Data Management Component (Activity 2.4)

First an approach for the Data Management Component has to be chosen (activity 2.4.1); a choice has to be made wether Flat Files, a Relational Database Management System or an Object Oriented Database Management System will be used.

According to this choice different aspects have to be considered like identification, normalization etc. For the chosen approach the possible Data Management Tools have to be assessed using a list

of criteria (activity 2.4.2).

The last step is designing the Data Management Component for the chosen approach and tool(s) (activity 2.4.3). This consists of designing the data layout and designing the corresponding services. How this has to be done depends on the chosen approach, Flat File, RDBMS or OODBMS.

Coad and Yourdon give a number of guidelines for Object Oriented Design to measure the quality of the design. These guidelines include the next topics:

* Coupling
* Cohesion
* Reuse
* Additional criteria:
    * Clarity of design
    * Generalization-Specialization depth
    * Simplicity of Classes & Objects
    * Simplicity of protocols
    * Simplicity of services
    * Volatility of the design
    * System size

To evaluate the design the authors suggest Walk-throughs and evaluation of the design with Critical Success Factors.