

“E-Film Hiring” Project

HW3. From Requirements to Design

Deliveries: Design Model (Design Class Diagrams, Interaction Diagrams, Statechart Diagrams)

THIRD MACRO-ACTIVITY: OBJECT DESIGN

To review the previous macro-activities, relationships between some software development artifacts are listed :

- The *use case* suggests the system events that are explicitly shown in system sequence diagrams.
- Details of the effect of the system events in terms of changes to domain objects may optionally be described in *system operation contracts*.
- The *system events* represent messages that initiate interaction diagrams, which illustrate how objects interact to fulfill the required tasks, that is the use case realization.
- The *interaction diagrams* involve message interaction between software objects whose names are sometimes inspired by the names of conceptual classes in the Domain Model, plus other classes of objects.

Why Object Design?

The requirements and object-oriented analysis has focused on learning to *do the right thing*; that is, understanding the goals for our system, and related rules and constraints. The following design macro-activity will stress *do the thing right*; that is, skillfully designing a solution to satisfy the requirements.

During object design, a logical solution based on the object-oriented paradigm is developed. The heart of this solution is the creation of **interaction diagrams**, which illustrate how objects collaborate to fulfill the requirements. Drawing interaction diagrams is a reflection of making decisions about the software system design.

In parallel with drawing interaction diagrams, **design class diagrams** can be drawn. These summarize the definition of the software classes (and interfaces) that are to be implemented in software and will be added to the classes inspired by the domain model. These artifacts are part of the **Design Model**. In the followings we will use a linear approach, for sake of simplicity and clarity, but you have to concurrently carry out both activities.

Keep in mind that principles of responsibility assignment and design patterns are the fundamental knowledge for object design!

First Step: Identifying interactions for use-case realization

After identifying requirements, creating a domain model, and identifying system operations you should add methods to the software classes, and define the messaging between the objects to fulfill the requirements.

Responsibility is "a contract or obligation of a class". Responsibilities are related to the obligations of an object in terms of its behavior. Basically, these responsibilities are of the following two types:

- knowing
- doing

Doing responsibilities of an object include:

- doing something itself, such as creating an object or doing a calculation
- initiating action in other objects
- controlling and coordinating activities in other objects

Knowing responsibilities of an object include:

- knowing about private encapsulated data
- knowing about related objects
- knowing about things it can derive or calculate

Responsibilities are assigned to classes of objects during object design. A responsibility is not the same thing as a method, but methods are implemented to fulfill responsibilities.

For assigning responsibility use typically five fundamental patterns:

1. Information Expert
2. Creator
3. High Cohesion
4. Low Coupling
5. Controller

There are others (Polymorphism, Fabrication, Indirection, Protected Variations etc.) but it is worthwhile mastering these five first because they address very basic, common questions and fundamental design issues.

Procedure

1. Design the interaction (sequence or collaboration) diagrams for each use-case and system operation identified in the previous homework. Use also system operation contracts when they are available. Do not hesitate to introduce new classes when needed:
 - Introduce a controller for each use case: a use case will be managed by at least one controller and each controller manages at least one use case.
 - Introduce graphical interface classes such that each time when an actor interacts with the system he/she does through a GUI.
2. Give reason for your design decisions by indicating the used patterns.
3. Complete the static model with the classes created in this step.

Second Step: Designing Design Class Diagrams (DCDs)

During the completion of interaction diagrams for use-case realizations, it is possible to identify the specification for the software classes (and interfaces) that participate in the software solution, and annotate them with design details, such as methods.

Although this presentation of DCDs *follows* the creation of interaction diagrams, in practice they are usually created in parallel. Many classes, method names and relationships may be sketched out very early in design by applying responsibility assignment patterns, prior to the drawing of interaction diagrams.

Bear in mind the rapport between the Domain Model diagrams and the design class diagrams from the Design Model!

The Domain Model does not illustrate software classes, but may be used to inspire the presence and names of some software classes in the Design Model. During interaction diagramming or programming, the developers may look to the Domain Model to name some design classes, thus creating a design with lower representational gap between the software design and our concepts of the real domain to which the software is related. **A design class diagram (DCD)** illustrates the specifications for software classes and interfaces (for example, Java interfaces) in an application. Typical information includes:

- classes, associations and attributes
- interfaces, with their operations and constants
- methods
- attribute type information
- navigability
- dependencies

In contrast to conceptual classes in the Domain Model, design classes in the DCDs show definitions for software classes rather than real-world concepts.

Procedure

1. During interaction discovery identify those classes that participate in interactions. These can be also found by scanning all the interaction diagrams and listing the classes mentioned: the design classes discovered while designing realizations of use-cases should be summarized in DCDs.
2. Draw a class diagram for these classes and include the attributes previously identified in the Domain Model that are also used in the design.
3. Identify the methods of each class by analyzing the interaction diagrams. In general, the set of all messages sent to a class X across all interaction diagrams indicates the majority of methods that class X must define. A message to a multiobject is interpreted as a message to the container/collection object itself.
4. Adorn each end of an association with a role, if needed, in the DCDs and with a navigability arrow. Navigability is a property of the role that indicates that it is possible to navigate uni-directionally across the association from objects of the source to target class. Navigability implies visibility. The usual interpretation of an association with a navigability arrow is attribute-like visibility from the source to target class. The required visibility and associations between classes are indicated by the interaction diagrams. Common situations suggesting a need to define an association with a navigability adornment from A to B are the followingsata:
 - A sends a message to B.
 - A creates an instance B.
 - A needs to maintain a connection to B.