The Object Model for a Product Based Development Process

Pavel Hruby
Navision Software A/S, Frydenlunds Allé 6, 2950 Vedbaek, Denmark
E-mail: ph@navision.com, homepage: www.navision.com

Abstract

The traditional workflow process model is typically illustrated with a graph of activities, tasks, deliverables and techniques. From an object-oriented perspective, every identifiable concept can be depicted with an object, and from this same perspective, evolution can be demonstrated with object interactions. There are at least two ways to describe the software development process using an object-oriented model. The first, an activity-based approach, uses activities as objects, tasks as object operations and deliverables as operation postconditions. The second, a deliverable-based approach, uses deliverables as objects, tasks as object operations and activities as object states. Techniques are considered the implementation of operations in both cases. The article will:

a) compare activity-based and deliverable-based object models of development processes and
b) describe our experience with a deliverable-based object model that we used for about one year when we developed an application used for financial management.

The Workflow Model

This section defines the terminology used throughout this article. *Tasks* (in this article) are small behavioral units that usually result in a deliverable. Some examples of tasks are: construction of a use case model, construction of a class model and writing of code. *Techniques* are formulas for performing tasks, for example, functional decomposition, using CRC cards, and programming in Smalltalk. *Deliverables* are final or intermediate products resulting from software development, for example, a use case model, a class model, or source code. *Activities* (in this report) are units that are larger than task units. Activities typically include several tasks and deliverables. Some examples of activities are requirements analysis, logical design and implementation.

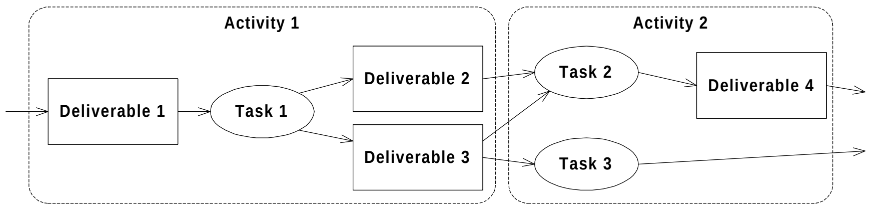

Fig. 1. Workflow model of a development process.

The Object-Oriented Model – with Activities as Objects

In an activity-based approach, the activities are modeled as objects, and tasks are modeled as their object operations. Project deliverables are postconditions of object operations. A good example of the activity-based approach is the OPEN Project Specification [2]. In this model, we see the following potential problems:

1. When setting up a new project, it can be difficult to determine the appropriate set of tasks leading to the delivery of the completed product. First of all, it would require a detailed knowledge of the entire method, and secondly, such a set of tasks would strongly depend on various project characteristics, for example, the project size.
2. Because the list of activities and tasks is determined by the method, using modified activities or tasks requires a modification of the process model.
3. The model is not fail-safe. If an activity or task is omitted in the project plan, there is typically no warning that something is missing.

The Object-Oriented Model – with Deliverables as Objects

In a deliverable-based approach, the deliverables are modeled as objects. These objects are instances of deliverable classes that determine behavior, attributes, semantics and relationships.

Deliverables have two kinds of *methods*. They have one or more *constructors*, which are one or more tasks used to create the deliverable, and *quality-assurance* methods which are, for example, completeness and consistency checks.

Deliverables have numerous *attributes*: name, kind, description, references to other deliverables, project, subsystem, increment identification, responsible developer and other attributes such as who created and modified the deliverable and when.

The attributes *kind* and *name* taken together are the key that uniquely identifies the deliverable in the data dictionary. The attribute *description* typically contains a UML diagram, a table or a text. The choice of a suitable representation is left up to the judgement of the developer and depends on the specific situation.

Each design increment is described by a single deliverable referred to as "Increment Context", that corresponds to "Task" in Microsoft Project used for project planning. "Increment Context" deliverables exists throughout the entire life cycle of the increment (from requirement analysis to implementation and testing. This is an important factor that makes management of incremental development easier.

Note that design deliverables may relate to more than one increment context documents. For example, different developers may work with the same class model in the scope of different increments, or more typically, class model can be reused within different increments.

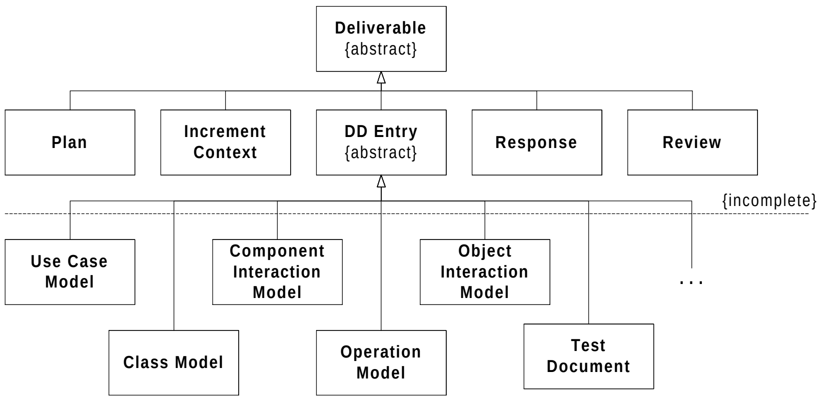

Fig. 2. Inheritance diagram showing the classes of deliverables used in the development process. The incompleteness of the inheritance tree allows for flexibility throughout the process and for the exact matching of project deliverables to different kinds of processes. The class DD Entry has abstract constructor and abstract quality criteria defined in derived classes.

Experience with the Deliverable-Based Process Model

We have used the described process model in several projects since the summer of 1996. Using the object model above, we have developed a process [4], based on the Fusion method [1], extended of the use cases and requirements analysis and made minor modifications to match the UML notation and incremental development. We used Lotus Notes as a repository for project deliverables as well as the data dictionary.

The main benefit of using these tools was flexibility – in notation, in the kinds of deliverables in the repository and in the possibilities provided for modifying the process according to the size and character of the increment.

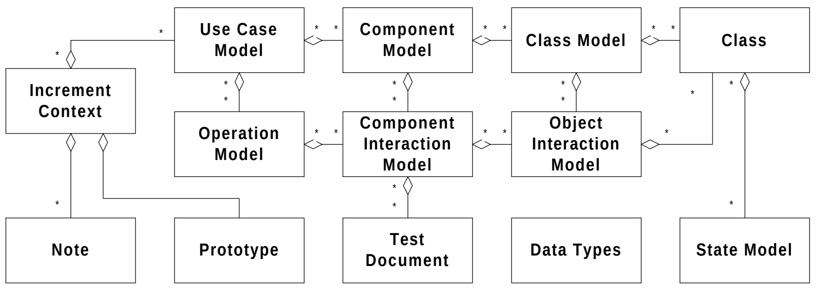

Fig. 3. Structure of typical references between analysis and design deliverables. In principle a deliverable can have bidirectional link to any other document or documents, to enable maximal flexibility.

After half a year, we had about 350 documents in the repository with the following distribution: increment context, 36%; note, 15%; use case model, 10%; component interaction model, 14%; component model, 4%; operation model, 5%; object interaction model, 3%; class model, 6%; class, 5%; state model, 1% and data types, 1%. The relatively large number of increment context documents can be explained by the fact that some minor increments were sufficiently defined by their increment context documents together with note documents and did not need to aggregate a full set of analysis and design deliverables.

Benefits of a Deliverable-Based Object Model as Compared to an Activity Based Object Model.

1. *It is easier to define a set of deliverables than a set of activities*. In a small project, it is often sufficient to produce only context, plan, source code and user documentation. As the project grows, the number of kinds of deliverables increases and always reflects project state and possible specific demands.
2. *The model is flexible*. The final product is not always a code. It could be another process, such as testing or version-control process. The deliverable-based object model can be easily adopted for different kinds of projects, just by defining the methods and attributes of the deliverable classes.
3. *The model provides good management support for incremental development*. The increment is described by a single document that remains throughout the increment's life cycle.
4. *The model is robust and consistent*. In an activity-based object model, an activity might be omitted during project planning, but in a deliverable-based model, the deliverable that is necessary for the development cannot be omitted because of the constructor and quality methods of related deliverables.

Conclusions

We have described a deliverable-based object-oriented model for the description of a software development process. Its main artifacts are deliverables, modeled as objects with constructor and quality-assurance methods and with a number of specific attributes. We have described our experience with the model and identified it as more flexible, easier to use in general, and more robust than an activity-based model.

References

[1] Coleman, D. et al.: Object-Oriented Development: the Fusion method, Prentice Hall 1994
[2] Henderson-Sellers, B., Graham, I., Younessi, H.: The OPEN Process Specification, Swinburne University 1997
[3] Hruby, P.: The Object-Oriented Model for a Development Process, OOPSLA'97
[4] Hruby, P.: The Fusion Process from an Object-Oriented Perspective. Submitted to the Fusion Newsletter.