Chapter

# 73 *From Analysis to Design*

**This chapter describes the differences between analysis and design activities. The aspects to cover when you are moving from object models to SDL, are also described.**

# From Analysis to Design – Overview

The task of moving from the analysis model of a system to a design of the system is a creative step that involves many design decisions to be taken. The reason is that the purpose of the analysis model is to give an abstract understanding of the system to be built, and an understanding of the concepts that are needed in order to describe a solution of the problem the system is to solve. The purpose of the analysis model is not to give a precise definition of **how** the problem is to be solved or to decide how the architecture and reuse issues are taken care of. These aspects are the purpose of the design activities. Both the system design and the object design are focussed on these type of questions that are necessary to solve in an efficient way in a development project.

The SOMT method introduces a specific concept to emphasize the task of moving from one model to another. It is particularly useful when moving from the analysis model, with its abstract view of the problem, to the design model, with its constructive precise definition of the system that solves the problem. This is the *Paste As* concept that was introduced in "Implinks and the Paste As Concept" on page 3666.

During design the idea is to encapsulate the creative design action taken when moving from the analysis model to the design model into one visible action. From a pragmatic point of view the designer just takes an analysis object and pastes it as a design object. However by doing this simple action the designer documents a number of the design decisions that are involved in the design activity.

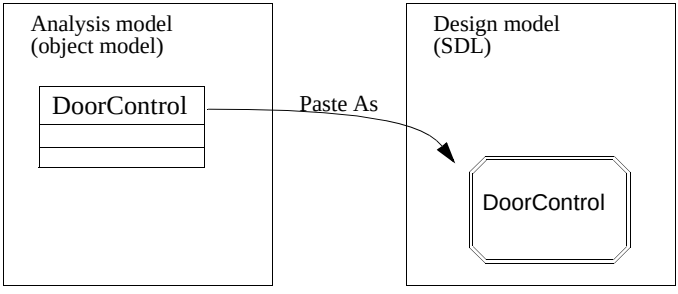


*Figure 645: The Paste As concept*

One important aspect of the paste-as concept is the relation to traceability links between different models. When performing the paste-as ac-

tion the designer not only creates design objects. He also creates a link between the analysis object and the design objects. This link defines in a precise and compact way the design decisions taken. The links are a vital part of the project documentation that is automatically created by the tools. The links can be inspected when understanding the system and can be used for a number of purposes, e.g.:

- Verification that all analysis concepts are implemented
- "What if" analysis, what happens if one analysis object is changed because of changed requirements

# Analysis vs. Design

When comparing analysis to design there is always a question of where the borderline between the activities are. In practise the distinction between analysis and design is somewhat arbitrary. From a practical point of view the important decision is not to define exactly what to call a specific activity, but rather to decide where to document a decision: in the analysis documentation or in the design model. The most important input to guide this decision is to consider the purpose of the different models. The analysis model has a focus on the architecture and most important concepts in the system, abstracting away from implementation details. It is used to provide a means to understand the application as efficiently as possible. The SDL design, on the other hand, is a complete definition of all necessary details.

In the context of mapping object models to SDL in SOMT the trade-off between analysis and design, essentially is a matter of where to document a decision:

- In the object model
- In the SDL model

As an example of this type of question that either can be described in the analysis model or in the design model, consider the issue of the traversal direction of an association, i.e. is the association a one-way or two-way association.

If the choice is to be made in the object model a notation or convention is needed to make it possible to add the extra information. For the one-way/two-way associations a possible notation could be to use an arrow head to denote the direction of the association. Note that this is only an example. SOMT does not recommend this convention.

If the choice is to be documented in the SDL model there is a choice of how the designer expresses his/her choice:, either

- During the Paste-As
- Directly in the SDL model

The result is in both cases the same: The design choice is documented in the combination of the implink (that defines the relation to the analysis model) and in the SDL model (that defines the details of the design choice).

For the one-way/two-way association the Paste-As alternative would imply that the designer interactively entered information as a parameter to the Paste-As of a class to a SDL data type that defined weather an association would be visible in the SDL data type. The association would only be visible if the direction was *from* the given class (or it was a two-way association).

If the one-way/two-way design choice was to be expressed directly in SDL the designer would manually either include or not include the association in the SDL data type corresponding to the class.

SOMT makes a trade-off between the different alternatives to accomplish an as smooth as possible transition from the analysis model to the design model. The SOMT mechanism is based on the following:

- The information that is naturally found in an analysis model is used as far as possible to deduce the mapping to SDL.
- The Paste-As functionality is defined to encapsulate the major design choices, like if an object will be mapped to a process type, struct data type or to some other concept. Paste-As also provides a predefined default choice for all other, minor design choices that have to be made. The intention is that the default settings should be useful in many cases, but in general some of the default choices will have to be modified during the design process.
- The designer will make the final design decisions directly in SDL. This implies that the SDL model is the artifact that documents most of the design choices.

The rationale behind this mechanism is mainly to be able to fulfil the objective of the analysis and design models as good as possible. The analysis model should define the high-level architecture and provide a sound basis for understanding the application as efficiently as possible.

To facilitate this the analysis model should be as abstract as possible while still containing all relevant information. The implication is that design decisions that are not needed to understand the application as far as possible should be kept in the design model and that abstractions should be used as much as possible in the analysis.

# Active vs. Passive Objects

One choice to be made when moving from analysis to design is to decide whether an analysis object is to be implemented as an active object or a passive object. An active object is one that has its own thread of control and that can exhibit an autonomous behavior without any other object acting upon it. An active object can be viewed as executing in parallel with other active objects. In a distributed system the active objects may execute on different hardware. In other cases the scheduling mechanisms and operating system creates an abstract interface on which the active objects are executing. One important implication of this is that when two different active objects interact, e.g. when invoking operations, this interaction is a communication between two parallel possibly distributed applications over some communication medium. The communication may be synchronous, e.g. in a client-server relation between objects; the client requests a service from the server and waits for a response from the server before continuing, or asynchronous, e.g. when peer entities are performing a common task using a specific protocol.

A passive object on the other hand does not have its own thread of control. It changes state only when it is being operated upon by another object. Typically a passive object is mainly an information container used in the system to store information about the outside world and the internal state. If the information handled by a system is complex then an entire structure of passive objects may be needed to model it. Often, particularly in distributed applications, there is a relation between the active objects and the passive objects in the sense that an active object "owns" a passive object. The passive object is part of the same thread of control as the active object and it is localized in the same place in the system structure. An example of this might be an active data base server object that handles the control of the access and operations on a data base defined by a number of passive objects. This is very similar to the notion of "container classes" that are used as abstractions of the classes that they contain.

# Reuse Issues in the Design Models

Another aspect of objects is if a particular object class is subject to reuse or not. The reuse may be on different levels of ambition. The lowest level of reuse is when an object is reused within one subsystem in an application. This has no major implication on how to package the object definition. Since one subsystem is most often developed by a small group of individuals there are no special requirements on the packaging of the object. The next level is when an object class is found to be useful in different subsystems implemented by different development teams. This situation makes things a bit more complex, since the object definition now probably should be grouped together with other similar objects and put in a special package. This package is used by many of the development groups and there may even be a special team devoted to the development of this support package. In any case it needs its own version control, interface definitions etc.

The third level of reuse is when an object is found so general and useful that it can be used in more than one project for more than one application. In this case there is obviously an even stronger need for a packaging mechanism, since the object together with other related objects now must be seen as a separate product.

Designing an object that is suitable for reuse requires special care: already early in the design it is necessary to identify that an object might be suitable for future reuse. Then the object needs to be specially designed to make it as general (and reusable) as possible. Careful thinking is also required to identify the parts of the object that later might have to be redefined when reused. These parts have to be marked “virtual” in SDL. A rule of thumb is that it is worthwhile to design a reusable object when it is expected that the object can be reused at least three times.

# Mapping Object Models to SDL

When moving from an analysis view of an application to a design of the application, one of the major tasks is to define the relation between analysis concepts and design concepts. There are two different ways to view this: either the design is seen as an elaboration and refinement of the analysis model or the design can be seen as a transformation of the analysis model. In practice both viewpoints are useful. When creating the design a transformation is performed from the object model developed in the analysis activity to the SDL design models. The concepts in the object model are mapped to suitable concepts in the SDL domain. However, when this mapping has been done the analysis model is used as an abstract view of the application where only the details relevant from an analysis perspective is present and the design is viewed as an elaboration and refinement of the analysis model.

When mapping object models to SDL there are several aspects to cover:

- Mapping the structure of object models to the structure of SDL
- Mapping the interfaces of objects to interface definitions in SDL
- Mapping the classes in the object model to SDL definitions
- Mapping existing state charts to SDL process definitions

The structure of an object model of interest for the mapping to SDL are of two kinds:

- The aggregation structure in the class diagrams
- The module structure that defines how the complete object model is divided into different modules

The aggregation structure in the object model is in general in SDL represented by the block hierarchy as described in "Architecture Definition" on page 3755. The actual mapping between the object model and SDL involves creating the block structure based on the aggregation structure and is further discussed in "Mapping Aggregations of Active Objects" on page 3777.

The module structure of the analysis model is one of the inputs to the definition of the design module structure. In SDL the major code structuring concept is the package and in general it is recommended to keep a simple mapping from the module structure of the analysis model to the packages in the SDL design. This is further discussed in "Design Module Structure" on page 3757.

In SDL the interfaces of processes, blocks etc. are separated from the description of behavior or implementation of the block/process. This implies that in general there are two different mappings from classes in the object model to SDL, one to the interface definition and one to the process/block definition. Interfaces in SDL are defined using signals or remote procedure calls as described in "Static Interface Definitions Using SDL" on page 3761 and the mapping to these concepts is outlined in "Mapping Object Models to SDL Interface Definitions" on page 3762.

The mapping of the classes in the object model to the corresponding SDL behavior definition is an issue of object design and is depending on if the object is an active or passive object. Active objects are mapped to SDL processes (or process types) and passive objects to data types. These different mappings are discussed in "Mapping Object Models to SDL Design Models" on page 3774.

# Summary

When moving from analysis to design, several decisions has to be made. These decisions should be documented and it should be possible to view the impact of such decisions. A way to achieve this is the Paste As functionality that produces links that could be traversed in order to do "what if" analysis or to do consistency checks between requirements and design.

When we start to design from our analysis models, several possible SDL mappings exist for entities in the object model and the use cases. The important decisions that have to be made concern issues like reuse and determining if objects should have autonomous behavior or not (active/passive objects).