PACKAGE - INTRODUCTION

Package is the UML mechanism for grouping things. It can be used to:

- group semantically related elements;
- define a “semantic boundary” in the model;
- provide units for parallel working and configuration management;
- package is used to provide an encapsulated namespace within which all names must be unique.

Analysis packages contain:

- use cases
- analysis classes
- use case realizations

A package syntax is shown in any of the form as in Figure:1.

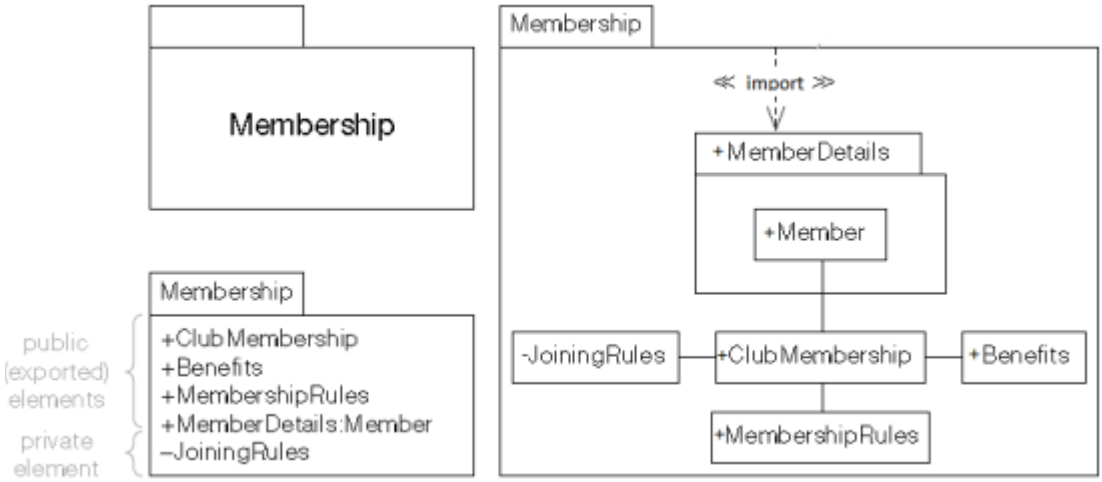

Figure:1 Package syntax

Visibilities of elements inside a package

An element with public visibility are visible to elements outside the package and they are exported by the package and are represented by pre-pending ‘+’.

An element with private visibility are completely hidden inside the package and are represented by pre-pending ‘-‘.

Standard package stereotypes

<<framework>> – if the package contains model elements that specify a reusable architecture.

<<modelLibrary>> – if the package contains elements that are intented to be reused by other packages.

A package define automatically an ***encapsulated namespace*** which means within the package boundary all elements name are unique. A ***qualified name*** is the representation by which an element inside a package is accessed. It is indicated with the help of '::'. for eg:- a class Librarian inside a package users which again is inside in Library is accessed as Library:: Users:: Librarian

Nested packages

Packages may be nested inside other packages to any depth.

Two possible representations of nested packages are shown in Figure:2. First one is commonly used.

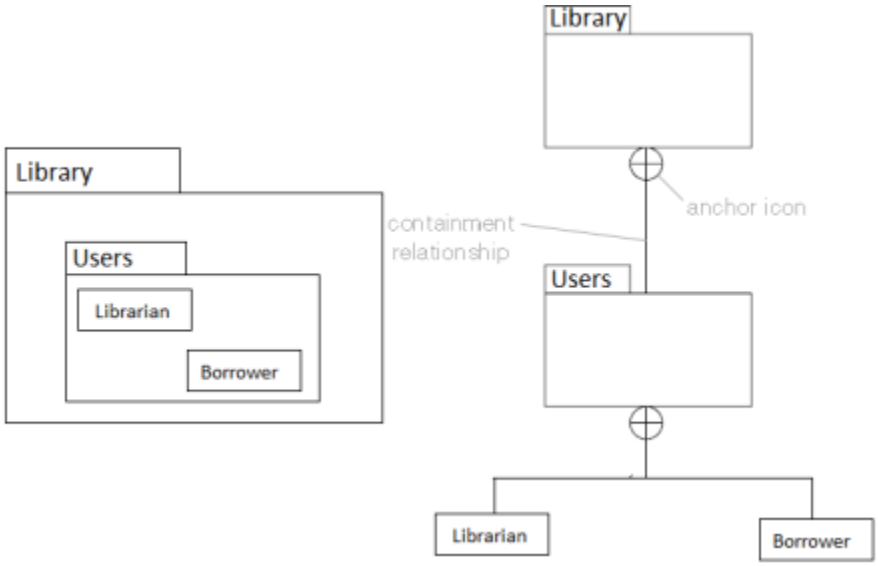

Figure:2 Nested Packages

Package dependencies

A dependency relationship indicates that one package depends in some way on another. Different dependency are explained below.

«use»- This means that an element in the client uses a public element in the supplier in

some way i.e the client depends on the supplier. If a package dependency is shown without a stereotype, then «use» should be assumed. Shown in Figure:3.

Figure:3 Use dependency

«import»- Public elements of the supplier namespace are added as public elements to the client namespace. Elements in the client can access all public elements in the supplier using unqualified names. Shown in Figure:4.

Figure:4 import dependency

«access»- Public elements of the supplier namespace are added as private elements to the client namespace. Elements in the client can access all public elements in the supplier using unqualified names. Shown in Figure:5.

Figure:5 access dependency

«trace»- «trace» usually represents an historical development of one element into another more developed version – it is usually a relationship between models rather than elements.A complete UML model can be represented by a package with a small triangle in its top right hand corner. Shown in Figure:6.

Figure:6 trace dependency

«merge»- Public elements of the supplier package are merged with elements of the client package. used only in meta modeling, not used in OO analysis and design. Shown in Figure:7.

Figure:7 merge dependency

Transitivity

Transitivity means that if there is a relationship between thing A and thing B and a relationship between thing B and thing C, then there is an implicit relationship between thing A and thing C. «access» dependency is not transitive and «import» dependency is transitive. This is illustrated in Figure: 8.

Figure:8 package-transitivity

Lack of transitivity in «access»means that:
public elements in package C become private elements in package B;
public elements in package B become private elements in package A;
Therefore elements in package A can’t see elements in package C.

Package generalization

In package generalization, the more specialized child packages inherit the public and protected elements from their parent package. Child packages may add new elements,

and may override elements in the parent package by providing an alternative implementation with the same name. This is repreasented in Figure: 9.

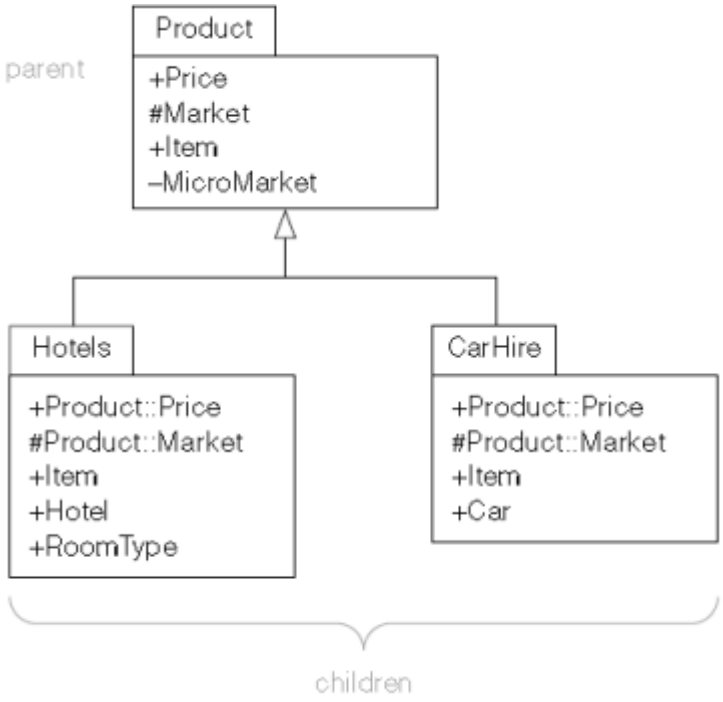

Figure: 9 Package Generalization

Architectural analysis

Architectural analysis partitions related classes into analysis packages, and then layers the packages.Represented in Figure: 10.

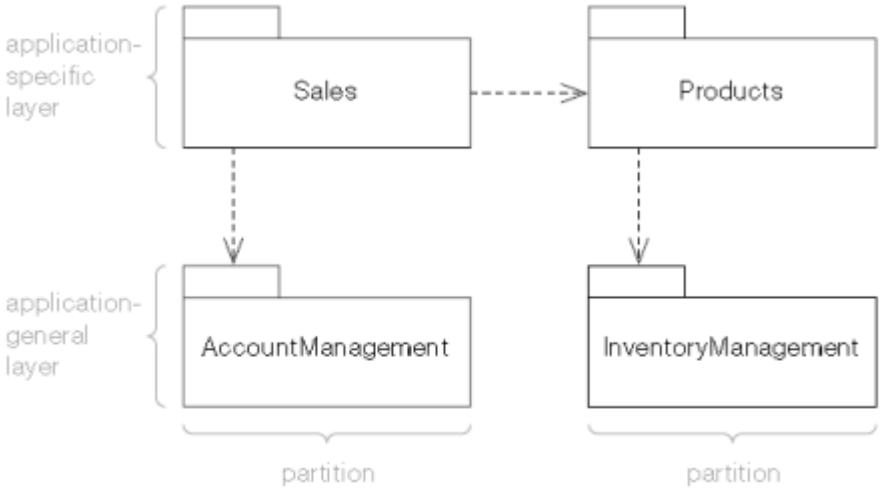

Figure: 10 Architectural analysis

One of the goals in architectural analysis is to try to minimize the amount of coupling in the system. It can be done in three ways:

- minimize the dependencies between analysis packages;
- minimize the number of public and protected elements in each analysis package;
- maximize the number of private elements in each analysis package.

Systems that exhibit a high degree of coupling are typically complex and difficult to build and maintain.

Finding analysis packages

Analysis packages are found by identifying groupings of model elements that have strong semantic connections. They are often discovered over a period of time as the model develops and matures. The keys to good package structure are high cohesion within a package, and low coupling between packages. A package should contain a group of closely related classes.

Cyclic package dependencies

Avoid cyclic dependencies in the analysis package model.If package A depends in some way on package B, and vice versa, there is a very strong argument for just merging the two packages and this is a perfectly valid way of removing cyclic dependencies.This is shown in Figure: 11.

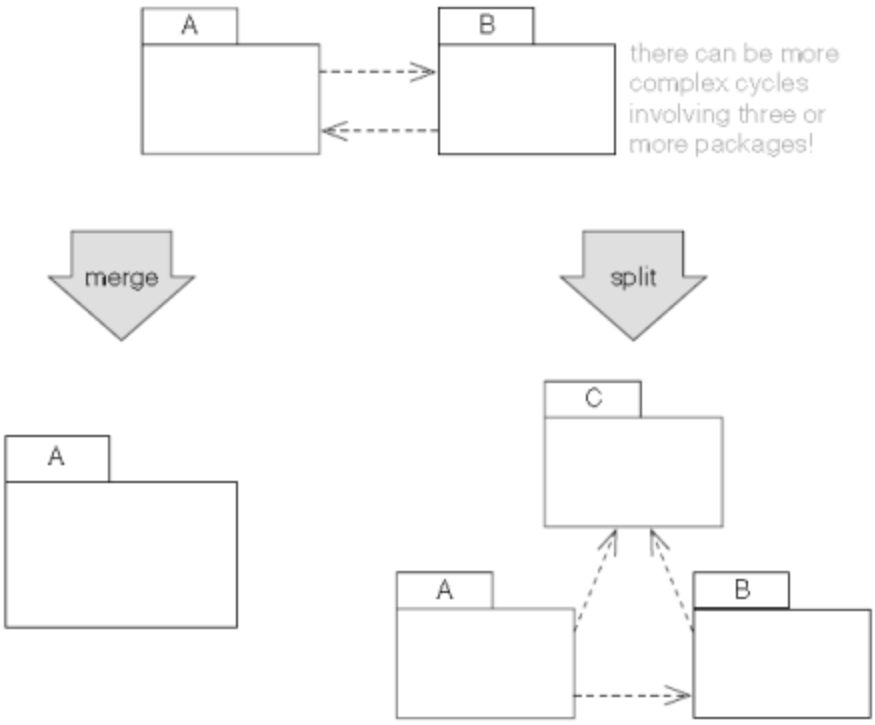

Figure:11 Cyclic package dependencies

Suppose we have a very simple model with one class in package A and another class in package B. If the class in package A has a bidirectional relationship with the class in package B, then package A depends on package B, but package B also depends on package A – we have a cyclic dependency between the two packages. The only ways to remove this violation are to refine the relationship between A and B by making it unidirectional(split), or to put the two classes in the same package(merge). Try to factor the common elements out into a third package C as shown in figure above.

Source : http://praveenthomasln.wordpress.com/2012/03/10/packages-s6-it/