Handout 8

Patterns – a new field that emerged from the object-oriented community.

[Riehle and Zullighoven] A pattern is the abstraction from a concrete that keeps recurring in specific non-arbitrary contexts.

[Coad] Pattern is a template of objects with stereotypical responsibilities and interactions.

[Fowler] A pattern is an idea that has been useful in one practical context and will probably be useful in others.

- Design patterns – useful architectural solutions
- Analysis (conceptual) patterns – useful conceptual models

Example of Analysis Patterns


Motivation: how to represent the many roles of an object?

Examples:
- companies may act as suppliers, distributors, customers, seller-agents, regulatory agencies
- people may do various things in a company, e.g. may have engineers, salesmen, directors, accountants

This is one of the most common situations in modeling: a group of objects that exhibit common behavior, but don’t all have common behavior. Sounds like a classic case for inheritance…but

Complications:
- An object may exhibit more than one bunch of behaviors (multiple inheritance? could lead to exponential number of subclasses)
- An object may take on new roles during its lifetime

Simple options:

1. Single role-class.
   When there are **no significantly different features** (from the system’s responsibilities prospective) between the engineers, salesmen, managers and accountants – use a **single role-class** with an attribute that differentiates one from another.
Advantages: simple!

Fear: in future the system will need to enable behaviors that are specific to particular roles. Have no fear: this representation is easily transformable into another one, when the need arises.

2. Separate role-classes.
When there are significantly different responsibilities (e.g. need to calculate total sales for a salesman, access manager’s budget, etc.) - use a generalization to capture commonalities, and specialization for each separate role.

Problem: arises when for instance “engineer John Smith adds some managerial responsibilities”. Creating a separate objects for John Smith –engineer and John Smith-manager results in loss of integrity (how do you tell if they represent the same person?)
More sophisticated solutions:

3. Role Objects
When an object during its lifetime may add roles (engineer becomes also a manager, then a retiree) – use a role object (actor-participant pattern).

Advantages: Can add/release roles to a person object and access them freely at any time. Adding a new role to the system has no impact on Person Class, neither on other roles. Disadvantage: two-step access to role-specific services. Consider getting the manager’s budget: first must find the manager role in the person’s role list, then ask that role for the budget.

Question: each Person object is connected to a (number of) generalized Person Roles. How does it distinguish which specific role is it?
Two solutions:
   a. add services such as isSalesman()? to PersonRole for each role, override it in each subclass, or
   b. add (a single) service hasType?(typename) to the Person Role, where typename can be “Salesman”, “Manager” or anything else; override it in each subclass.

With solution (a) – when a new role is added to the problem domain – must add another service isNewRole()? to PersonRole -- changing the interface.
With solution (b) – no changes in the interface are necessary.
4. Role Relationship.
Usually comes up when considering an organization with several different groups, where an employee may have roles in more than one groups.
Class PersonRole implements an association between a person and a group.
In this case not only need to know the roles a person is taking on, but also which role s/he plays in a particular group (e.g. School principal is a manager of the school faculty and a member of the school board)

Advantages: Can add role relationships at any time, keep the history of the changes. RoleRelationship may know the rules of which kind of people can have association with which groups (checking for conflict of interests, for instance).