Handout 10

Object Design: adding implementation details

Design of Attributes and Services is relatively straightforward:
- attributes: choose data structures, specify data types
- services: choose algorithms to implement operations

Design of Associations

Associations are the “glue” of the model, they provide access paths between objects.

During design – formulate a strategy of implementing the associations.
To make optimal decisions – need to analyze the way an association is used in an application.

Conceptually, all associations are bi-directional.
Implementation-wise it is important to understand
1. in which way the association is traversed in the application,
2. how often will an association be used in each direction,
3. how often will an association be changed.

Example: Consider the Person-Company association from above, and the following Services

S1 : “Find the Company for which a given Person works”
S2 : “Find out who works for a given Company ”

and an update required when a given person quits working for a company.

Assume there are N Persons and M Companies

There are two categories of implementation: One-directional and two-directional.
Association:

One-directional implementation

Include an attribute in one of the participating classes to point at (reference) the associated object(s).

A)

How to implement S1 and S2?

How many objects have to be examined while implementing S1? S2?

B)

How to implement S1 and S2?

How many objects have to be examined while implementing S1? S2?

Pros: minimizes the storage cost
minimizes the update cost

Cons: backward traversal is expensive (requires searching many objects)

Useful for situations when traversal in one of the ways is relatively rare and can afford the overhead searching time.

---

1 A pointer or reference is a memory address of an object.
Two-directional implementation
A) If access in both directions must be fast, use pointer/reference attributes in **both** objects.

Pros: fast access both ways.
Cons: an update requires changes in two objects.
Useful when access outnumbers updates.

B) Implement as a distinct association-object independent of either class.
This is the only way to represent an association between predefined classes from a class library that **cannot** be modified (i.e. the pointer attribute cannot be added to class definition).

Association object – set of pairs of pointers/references to associated objects

Pros: Updates are localized in the association object.
Cons: Access is slow (max. number of pairs is N*M)

Useful for **sparse** associations, in which most objects of participating in association classes (Person, Company) do not participate
For faster access in both ways implement the association as **two hash tables**: one hashed by Person, the other—by Company name.

![Diagram of two hash tables](image)