Distributed objects and object-based middleware

INF 5040 autumn 2008

lecturer: Frank Eliassen

Why objekt-based distribution middleware?

- *Encapsulation*
  - natural unit of development of distributed applications
- *Data abstraction*
  - separation between implementation (class) and specification (interface)
- *Incremental development*
  - an object can be replaced by an alternative implementation
- *Extensibility*
  - can add new classes and objects
- *Inheritance of implementations and interfaces*
  - supports reuse of code and interface
- *Subtyping*
  - enables flexible selection of services in a distributed environment

Distributed objects - I

- Objects in a distributed program executes in different processes.
  - each object has a **remote interface** for controlling access to its methods and attributes that can be accessed from other objects in other processes located on the same or other machines
    - declared via an "Interface Definition Language" (IDL)
  - remote object
    - object that implements a remote interface
  - Remote Method Invocation (RMI)
    - method call from an object in one process to a remote object in another process

Distributed objects - II

- Remote objects have a unique identity: Remote Object Reference (ROR)
- Other objects that want to invoke methods of a remote object needs access to its ROR
- RORs are "first class values"
  - can occur as arguments and results in RMI
  - can be assigned to variables
- Remote objects are encapsulated by an interface
- Remote objects have a set of named attributes that can be assigned values
- Remote objects can raise "exceptions" as a result of method invocations

The type of a distributed object

- Attributes, methods and exceptions are properties objects can export to other objects
- These properties determine the type of an object
- Several objects can export the same properties (same type of objects)
- The type is defined once
- The object type is defined by the interface specification of the object

Exceptions

- Remote method invocations in a distributed systems can fail
- Exceptions are used to explain the cause of the failure to the method caller
- Failure of remote method invocations can be
  - generic
  - specific (application specific)
- Specific failure can be declared in object type specific exceptions

Declaration of remote methods

- A remote method is declared by its signature that consists of
  - a name
  - a list of **in**, **out**, and **inout** parameters
  - a return value type
  - a list of exceptions that the metod can raise

Example: CORBA/IDL

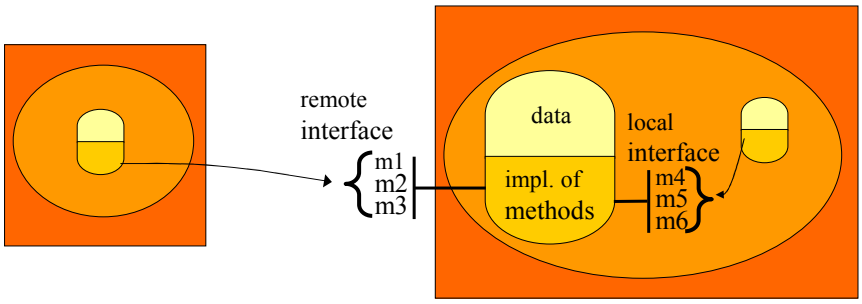
```
typedef enum {
        Goalkeeper, Defender, Midfielder, Striker
        } Position
interface Player {
        readonly string Firstname;
        readonly string Surname;
        readonly short Age;
        Position Role;
        Exception AlreadySelected{ };

        void select (in Date d) raises (AlreadySelected);
};
```

Remote method invocations

- A client object can request the execution of a method at a remote object
- Remote methods are invoked by sending a message (incl method name and arguments) to the remote object
- The remote object is identified by an object reference (Remote Object Reference - ROR)
- Clients must be able to handle exceptions that the method can raise

Fjernobjekt med fjerngrensesnitt



Subtyping of distributed objects

- object types are organised into a type hierarchy
- subtypes inherit attributes, exceptions, and methods from their supertypes

```
interface Club {
        readonly string name;
        readonly string streetaddr;
        . . .
};
interface FootballClub : Club {
        . . .
};

interface HandballClub : Club {
        . . .
};
```

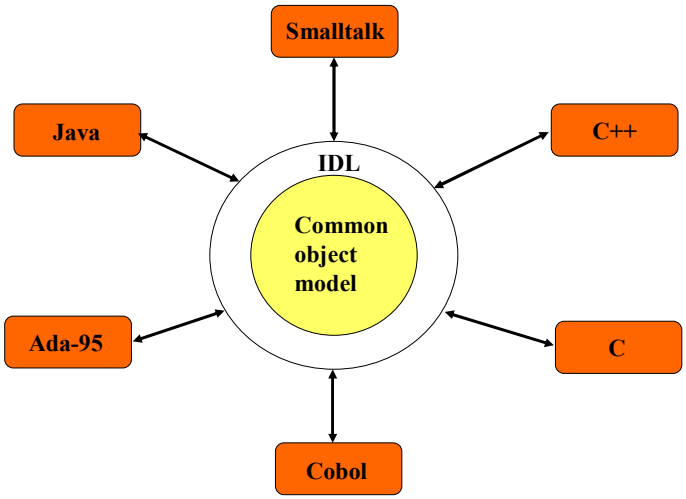

Language heterogeneity

- Some object-based middlewares allow interacting objects to be implemented in different programming languages
- Interoperability based on a common object model provided by the middleware
- Need for advanced mappings between different object implementation languages and the common object model

Interface Definition Language (IDL)

- Language for expressing all concepts in the object model of the middleware platform
- Requirement
  - must be independent of a specific programming language
  - need not be computationally complete
- Need for bindings (or language mappings) to different programming languages
- Example:
  - CORBA object model and different language bindings for CORBA/IDL

Common object model



The elements of the common object model

- Metalevel model for the type system of the middleware
- Defines the meaning of e.g.,
  - object identity
  - object type (interface)
  - operation (method)
  - attribute
  - method invocation
  - exception
  - subtyping/inhertance
- Must be defined generally enough to be mappable to most programming languages

Summary

- Distributed objects executes in different processes.
  - remote interfaces allow an object in one process to invoke methods of objects in other processes located on the same or on other machines
- Object-based distribution middleware:
  - midleware that models a distributed application as a collection of interacting distributed objects (e.g., CORBA, Java RMI)
  - some middlewares (as CORBA) allow objects in the same application to be implemented in different programming languages