Object Models for Distributed Systems

Motivation

- ***Distributed Systems consist of multiple components.***
- ***Components are heterogeneous.***
- ***Components still have to be interoperable.***
- ***There has to be a common model for components that expresses***
  - ***component states,***
  - ***component services and***
  - ***interaction between components.***

OO Approach to Distributed Systems

- *Components ⇔ objects.*
- *Visible component state ⇔ object attributes.*
- *Usable component services ⇔ object operations.*
- *Component interactions ⇔ operation execution requests.*
- *Component service failures ⇔ exceptions.*

Need for an object model

- *There are many different object-oriented approaches*
- *Distribution middleware must define object model that can serve as a common basis for heterogeneous components.*
- *What are the ingredients for an object model?*
- *We now introduce the OMG object model.*

Object

- *Has a unique identifier.*
- *May have many different references that refer to the object.*
- *Has a set of attributes whose names denote values.*
- *References may denote*
  - *equal objects*
  - *identical objects*
- *Is encapsulated by operations.*
- *May raise particular exceptions.*

Sample Objects

8987:Player
first = "Jürgen" *surname = "Klinsmann"* *age = 34* *role = Forward*

898:FootballClub
name = "Tottenham Hortspur FC" *adr = "White Hart Lane"*

Types and Distributed Objects

- *Attributes, operations and exceptions are properties objects may export to other objects.*
- *Multiple objects may export the same properties.*
- *Only define the properties once!*
- *Attributes and operations, and exceptions are defined in object types.*

Attributes

- *Attributes have a name and a type.*
- *Type can be an object type or a non-object type.*
- *Attributes are readable by other components.*
- *Attributes may or may not be modifiable by other components.*
- *Attributes correspond to one or two operations (set/get).*

Exceptions

- *Service requests in a distributed system may not be properly executed.*
- *Exceptions are used to explain reason of failure to requester of operation execution.*
- *Operation execution failures may be*
  - *generic or*
  - *specific.*
- *Specific failures may be explained in specific exceptions.*

Operations

- *Operations have a signature that consists of*
  - *a name,*
  - *a list of in, out, or inout parameters,*
  - *a return value type, and*
  - *a list of exceptions that the operation can raise.*

Example: Player

```
typedef enum {
  Goalie, Defender, Midfielder, Forward
} Position
interface Player {
  readonly string first;
  readonly string surname;
  readonly short Age;
  Position Role;
  Exception AlreadyBooked{};
  void book (in Date d) raises{AlreadyBooked};
};
```


Operation Execution Requests

- A client object can request an operation execution from a server object.
- Operation request is expressed by sending a message (operation name) to server object.
- Server objects are identified by object references.
- Clients have to react to exceptions that the operation may raise.

Subtyping

- *Properties shared by several types should be defined only once.*
- *Object types are organised in a type hierarchy.*
- *Subtypes inherit attributes, exceptions and operations from their supertypes.*
- *Subtypes can add more specific properties.*
- *Subtypes can redefine inherited properties.*

Subtyping Example

```
interface Club {
  readonly string name;
  readonly string street;
  readonly string city
};
interface FootballClub : SportsClub {
  ...
};
interface CricketClub : SportsClub {
  ...
};
```

Polymorphism

- *Object models may be statically typed.*
- *Static type of a variable restricts the dynamic type of objects that can be assigned to it.*
- *Polymorphism denotes the possibility of assignments of objects that are instances of the static type and all its subtypes.*

Polymorphism Example