Designing Distributed Objects

Motivation

- ***Many will have experience with designing local objects that reside in the run-time environment of an OO programming lang.***
- ***Designing distributed objects is different!***
- ***Explain the differences.***
- ***Avoid some serious pitfalls***

Local vs. distributed Objects

- *References*
- *Activation/Deactivation*
- *Migration*
- *Persistence*
- *Latency of Requests*
- *Concurrency*
- *Communication*
- *Security*
- ➡ *Several Pitfalls are lurking here*

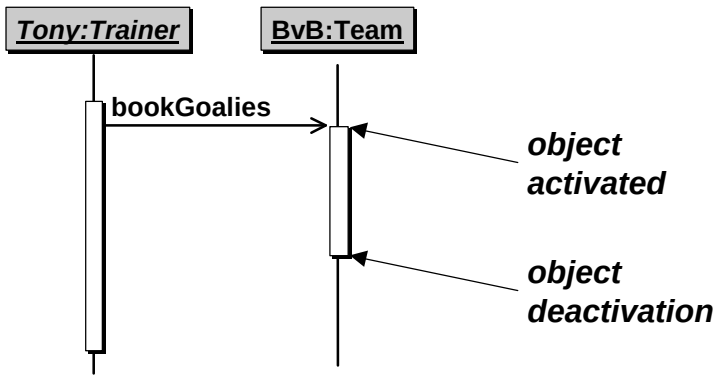
Object References

- *References to objects in OOPL are usually pointers to memory addresses*
  - *sometimes pointers can be turned into references (C++)*
  - *sometimes they cannot (Smalltalk,Java)*
- *References to distributed objects are more complex*
  - *Location information*
  - *Security information*
  - *References to object types*
- ➡ *References to distributed objects are bigger (e.g 350 bytes with Orbix).*

Activation/Deactivation

- *Objects in OOPL are in virtual memory between creation and destruction.*
- *This might be inappropriate for distributed objects*
  - *sheer number of objects*
  - *objects might not be used for a long time*
  - *some hosts might have to be shut down without stopping all applications*
- *Distributed object implementations are*
  - *brought into main memory (activation)*
  - *discarded from main memory (deactivation)*

Activation/Deactivation (cont'd)



Activation/Deactivation (cont'd)

- *Several questions arise*
  - *Repository for implementations*
  - *Association between objects and processes*
  - *Explicit vs. implicit activation*
  - *When to deactivate objects*
  - *How to treat concurrent requests*
- *Who decides answers to these questions?*
  - *Designer*
  - *Programmer*
  - *Administrator*
- *How to document decisions?*

Persistence

- *Stateless vs. statefull objects*
- *Statefull objects have to save their state between*
  - *object deactivation and*
  - *object activation*

  onto persistent storage
- *Can be achieved by*
  - *externalization into file system*
  - *mapping to relational database*
  - *object database*
- *To be considered during object design*

Object Lifecycle

- *OOPL objects reside in one virtual machine.*
- *Distributed objects might be created on a different machine.*
- *Distributed objects might be copied or moved (migrated) from one machine to another.*
- *Deletion by garbage collection does not work in a distributed setting.*
- *Lifecycle needs attention during the design of distributed objects.*

Latency of Requests

- *Performing a local method call requires a couple of hundred nanoseconds.*
- *An object request requires between 0.1 and 10 milliseconds.*
- ➡ *Interfaces of distributed objects need to be designed in a way that*
  - *operations perform coarse-grained tasks*
  - *do not have to be requested frequently*

Example: Iteration over a Sequence

- *Java*

Vector
+size():int +elementAt(i:int):Object ...

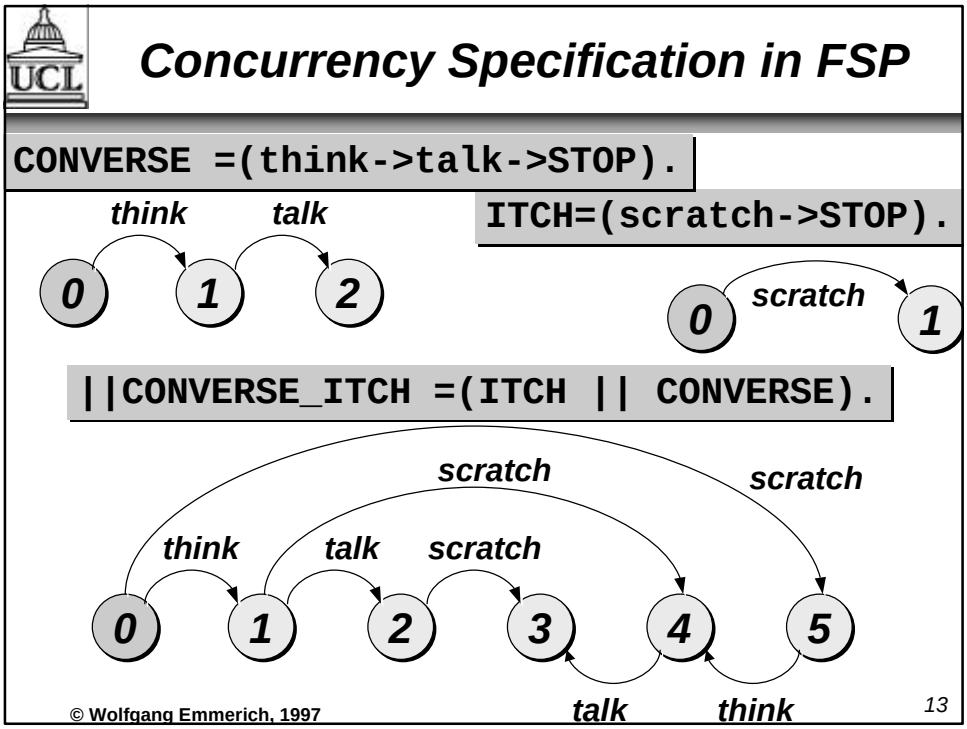
- *Distributed Objects*

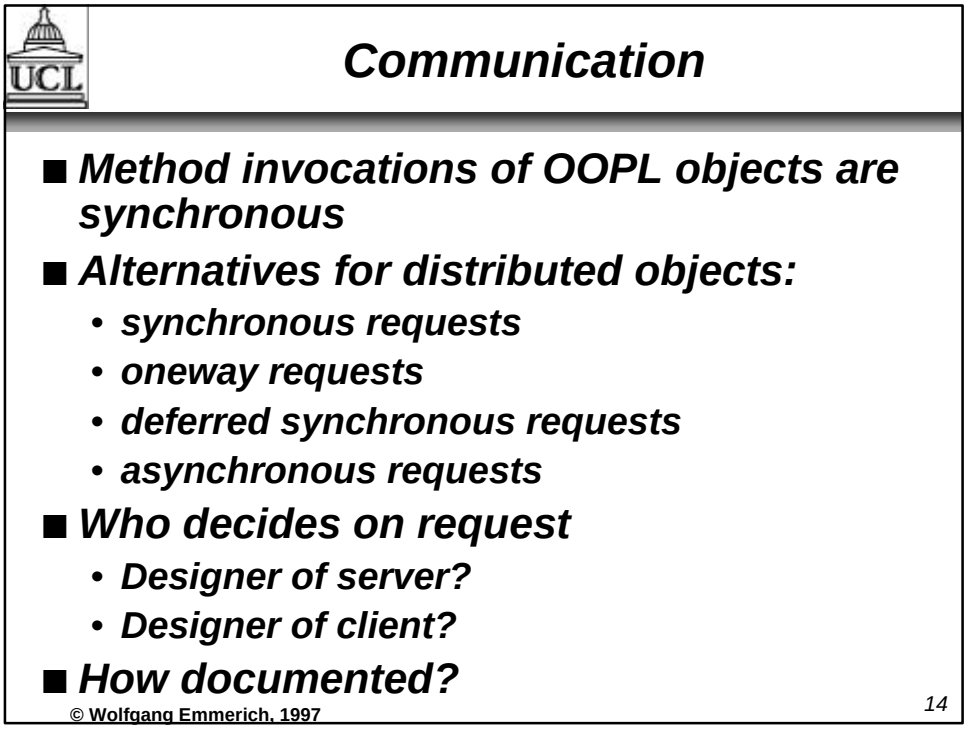
List
+long list (in how_many:long, out l:sequence<object>, out bi:Iterator i)

Iterator
+next_one(out o:Object): boolean +next_n(in how_many:long, out l:sequence<object>):boolean

Concurrency

- *Execution of OOPL objects is often sequential*
- *Execution of distributed objects is always concurrent*
- *Concurrency between*
  - *processes*
  - *within objects*
- *How to model concurrency*
  - *Hoare's CSP*
  - *Milner's CCS*
  - *Magee & Kramer's FSP*


Concurrency Specification in FSP
CONVERSE =(think->talk->STOP).
think
talk
0
1
2
ITCH=(scratch->STOP).
0
scratch
1
||CONVERSE_ITCH =(ITCH || CONVERSE).
scratch
scratch
think
talk
scratch
0
1
2
3
4
5
talk
think
© Wolfgang Emmerich, 1997
13


Communication
■ Method invocations of OOPL objects are synchronous
■ Alternatives for distributed objects:
• synchronous requests
• oneway requests
• deferred synchronous requests
• asynchronous requests
■ Who decides on request
• Designer of server?
• Designer of client?
■ How documented?
© Wolfgang Emmerich, 1997
14

Security

- *Security in OO applications can be dealt with at session level.*
- *OOPL Objects do not have to be written in a particular way.*
- *For distributed objects:*
  - *Who is requesting an operation execution?*
  - *How can we know that subject is who it claims to be?*
  - *How do we decide whether or not to grant that subject the right to execute the service?*
  - *How can we prove that we have delivered a service so as to make the requester pay*