Asynchronous Method Invocation

- Bisherige Möglichkeiten eines Methodenaufrufs in CORBA:
  - "synchron" (insbes. bei Rückgabewerten; analog zu RPC)
  - "verzögert synchron" (Aufrufer wartet nicht auf das Ergebnis, sondern holt es sich später ab)
  - "one way" (Aufrufer wartet nicht) mit "best effort"-Semantik ("fire and forget")

gedacht war an UDP-Implementierung; Semantik (z.B. Fehlermeldung bei Misslingen?) implementierungsabhängig

---

- Neu: Asynchronous Method Invocation (AMI)
  - bisher eher umständlich mit mehreren Threads simuliert
- Zwei Aufruftechniken bei AMI:
  - (1) Callback
    - Client gibt dem Aufruf eine Objektreferenz für die Antwort mit
    - Callback-Objekt kann sich im Prinzip irgendwo befinden
    - Kommunikations-Exceptions werden im Callback-Objekt ausgelöst
  - (2) Polling
    - Client erhält sofort ein Objekt zurück, das er für Polling oder zum Warten auf Antwort nutzen kann

Time-independent Invocation (TII)

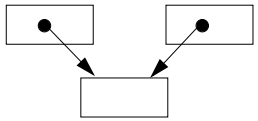
- Teil des Messaging Services: Aufruf von Objekten, die nicht aktiv sind oder zeitweise nicht erreichbar sind
- Aufruf-Nachrichten werden von zwischengeschalteten "Router Agents" verwaltet
  - Store and Forward-Prinzip
  - Router Agent beim Client ermöglicht disconnected operations
  - Router Agent beim Server kann dessen Eingangsqueue verwalten
- "Interoperable Routing Protocol" sorgt dafür, dass Router Agents verschiedener Hersteller interagieren
- Quality of Service (QoS) steuerbar (als "Policy")
  - z.B. max. Round Trip-Zeit: dadurch müssen Router Agents die Nachrichten nicht beliebig lange aufbewahren
  - oder z.B. Aufrufreihenfolge: Soll Router seine gespeicherten Aufträge zeitlich geordnet oder nach Prioritäten oder... ausliefern?

- Beachte: QoS ist ein gewichtiger Name für ein einfaches Prinzip ohne feste Garantien

Objects by Value

- Bisher war nur *Referenzübergabe* möglich
  - um es Objekten zu gestatten, Methoden anderer Objekte aufzurufen, konnten bisher Objektreferenzen als Parameter übergeben werden
  - Objekt selbst bleibt aber "am Platz", Aufruf wird also immer als Fernaufruf über das Netz geschickt
  - ferner kommt es zum gelegentlich unerwünschten Aliasing-Effekt: unbedachte Rückwirkungen auf das "Originalobjekt"

- Bei *Wertübergabe* wird das Objekt serialisiert und im Adressraum des Empfängers eine Kopie angelegt
  - Marshalling des Objektzustandes (d.h. der *Daten*)
  - auf Empfängerseite existiert eine Factory, die das Objekt als Kopie (mit eigener Identität) erzeugt
  - was geschieht bei Alias-Zeigern bzw. Zyklen bei der Serialisierung komplexer Strukturen?



- Bei heterogenen Umgebungen: Wie transportiert man das *Verhalten* des Objektes zum Empfänger?
  - es handelt sich um ausführbaren Code (für welche Plattform?)
  - kann von verschiedenen Sprachen (C, Java,...) erzeugt worden sein
  - einfach bei Java auf beiden Seiten: Bytecode ist unabhängig vom Maschinentyp durch die JVM in gleicher Weise interpretierbar
  - ansonsten muss sich die Factory beim Empfänger den Code besorgen (aus lokaler Bibliothek, übers Netz...)

- Leider können jedoch keine normalen CORBA-Objekte per Value übergeben werden, nur sogen. "valuetypes"
  - neues Konstrukt der IDL (→ für Anwender dadurch kompliziert)
  - Wertübergabe beliebiger Objekte wäre zu aufwändig

Real-Time CORBA

- Ziel: Vorhersagbares Ende-zu-Ende-Verhalten (insbesondere beim entfernten Methodenaufruf)
  - sowohl für "Hard Real-Time"
  - als auch für "Soft Real-Time" (mit nur statistischen Aussagen)

- Setzt zugrundeliegendes Realzeit-BS voraus

- Einige Mechanismen:
  - Prioritäten
  - Timeouts für Aufrufe
  - Multithreading (mit geeignetem Scheduling), Threadpools
  - private (statt gemultiplexte) Verbindungen
  - austauschbare Kommunikationsprotokolle

- Anwendungsbereiche z.B.:
  - verteilter Telefonswitch
  - Prozessautomatisierung
  - ...

CORBA und Java

- Java ist weit verbreitet (als "Internetprogrammier-sprache") und in gewissem Sinn eine "Konkurrenz"
  - allerdings hegemonial im Sinne von "überall Java / JVM"

- Ziel: Interoperabilität durch Zusammenführung von Java RMI und CORBA IIOP

  - Remote Method Invocation: Entfernter Methodenaufruf mit Transport (und dabei Serialisierung) auch komplexer Objekte
  - auch Code (Bytecode) wird übertragen und remote ausgeführt
  - damit kann z.B. ein Proxy eines Servers (z.B. ein Druckertreiber) jeweils aktuell zur Laufzeit vom Server zum Client geschickt werden
  - der Proxy kann dann (unsichtbar für den Client) z.B. mit dem Server ein privates Protokoll realisieren oder sonstige Dinge lokal tun

- IDL automatisch aus Java-Programmen generieren
  - "reverse mapping" (weiterhin existiert natürlich IDL → Java mapping)
  - Java-Programmierer brauchen kein IDL zu nutzen und zu lernen
  - aus Java heraus sind so Objekte anderer Sprachen ansprechbar (bzw. z.B. Java-Server, der von Clients anderer Sprachen genutzt werden kann)

- Aufbrechen des "single language Paradigmas" von Java
  - Java-Objekte werden von CORBA aus zugreifbar

- Kleinere Einschränkungen jedoch notwendig
  - Java-RMI und CORBA-IDL sind nicht deckungsgleich

CORBA-"Konkurrenz"

- Microsoft: .NET und ähnliche Produkte

- Java-Technologie
  - z.B. Enterprise Java Beans (EJB), RMI

- Web-Services (und verwandte Systeme)
  - XML, WSDL, SOAP,...
  - Integrationsplattformen (z.B. WebSphere von IBM)

- ...