Script **generated by TTT**

Title: Matthes: Soft-Arch (17.01.2012)

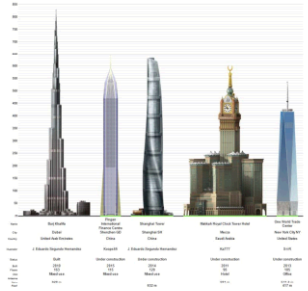
Date: Tue Jan 17 19:15:19 CET 2012

Duration: 23:45 min

Pages: 9

Software Architectures

6. Architecture for Systems of Systems



Prof. Florian Matthes, Sascha Roth
Software engineering for business information systems (sebis)

wwwmatthes.in.tum.de

Fakultät für Informatik
Technische Universität München

APPLICATION

ENTERPRISE IT ARCHITECTURE

Software Architectures

6. Architecture for Systems of Systems

Prof. Florian Matthes, Sascha Roth
Software engineering for business information systems (sebis)

wwwmatthes.in.tum.de

EVOLUTION

Software Architectures: Outline © sebis 1

6 – Architectures for Systems of Systems

sebis

- "Conventional" Middleware for Distributed Information Systems [Al05]
  - RPC and Related Middleware
  - Object Brokers
  - Message-Oriented Middleware
  - EAI Middleware: Message Brokers
- Web Services
- Service-Oriented Architecture
- REST [Fi00]

ROM

Recommended Reading: [Al04]

Alonso and his co-authors deliberately take a step back. Based on their academic and industrial experience with middleware and enterprise application integration systems, they describe the fundamental concepts behind the notion of Web services and present them as the natural evolution of conventional middleware, necessary to meet the challenges of the Web and of B2B application integration.

From this perspective, it becomes clear why Web services are needed and how this technology addresses such needs.

Alonso, G.; Casati, F.; Kuno, H.; Machiraju, V.: *Web Services – Concepts, Architectures and Applications*. Springer, 2004.

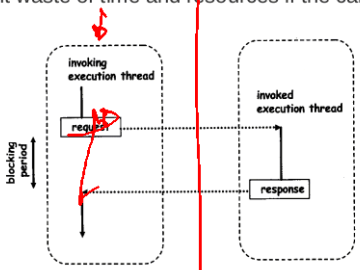
Rather than providing a reference guide or a manual on how to write a Web service, the authors discuss challenges and solutions that will remain relevant regardless of how emerging standards and technologies evolve.

Communication in an Information System

- The dominating characteristic of any software interaction is whether it is *synchronous* or *asynchronous*.
- Formally: blocking vs. non blocking
- Synchrony has nothing to do with concurrency and parallelism.

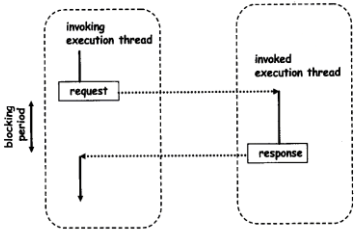
Synchronous or Blocking Calls

- In synchronous interaction, a thread of execution calling another thread must wait until the response comes back before it can proceed.
- This leads to simpler design which are easier to understand.
  - The state of the calling thread will not change before the response comes back.
  - There is a strong correlation between the code that makes the call and the code that deals with the response.
- Can be a significant waste of time and resources if the call takes time to complete



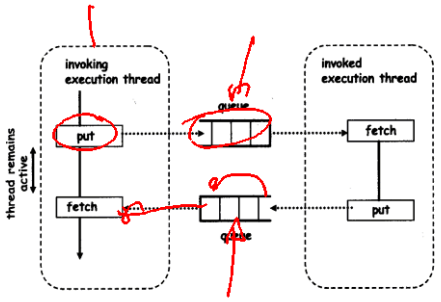
Synchronous or Blocking Calls

- In synchronous interaction, a thread of execution calling another thread must wait until the response comes back before it can proceed.
- This leads to simpler design which are easier to understand.
  - The state of the calling thread will not change before the response comes back.
  - There is a strong correlation between the code that makes the call and the code that deals with the response.
- Can be a significant waste of time and resources if the call takes time to complete



Asynchronous or Non Blocking Calls

- Simple example: e-mail
- A message is sent and, some time later, the program checks whether an answer has arrived.
- This allows the calling program to perform tasks in the meanwhile and eliminates the need for any coordination between both ends of the interaction.



Middleware

- Middleware offers programming abstractions that hide some of the complexity of building a distributed application.
  - The middleware takes care of some aspects.
  - The programmer has access to functionality that otherwise would have to be implemented from scratch.
- There is a complex software infrastructure that implements those abstractions.
  - Tends to have a large footprint
  - Extensions and enhancements of the original programming abstraction make the infrastructure more complex.