**Overzicht**

- Architectuur intro
- Client/Server architecturen
- UML-modellen van aantal architectuur stijlen
- Definitie van architectuur
- Architectuur evaluatie:
  - Kwantitatieve methoden, metrieken
    - vb Siemens/VDO
  - Kwalitatieve aanpak, bv gebaseerd op scenario’s

**What is Software Architecture?**

*A definition:*

**Software Architecture** is the global organization of a software system, including
- the division of software into subsystems/components,
- policies according to which these subsystems interact,
- the definition of their interfaces.

*(free after) Object Oriented Software Engineering*

T. C. Lethbridge & R. Laganière
McGraw Hill, 2001

**The Importance of Architecture**

“A correct architecture has the largest single impact on cost and quality of the product.”

Maranzano, ATT, 1995
Business Objectives of Sw. Arch.

Reduce development cost
Through improved communication between developers and earlier assessment of design alternatives and risks

Reduce time-to-market
Through enabling reuse and gradual evolution

Reduce maintenance cost
Through incorporation of foreseeable changes

Improve product quality
Increase fitness for use through stakeholder involvement; allow them to evaluate different architectural solutions and their consequences (what-if questions)

Client-Server Architectures

- What is Client/Server?
- Why Client/Server?
- 2-tier C/S model
- 3-tier C/S model
- Comparison of the models (pro’s / con’s)

What is Client / Server?

Client: an application that makes requests (to the servers) and handles input/output with the system environment (need not know of existence of other clients)

Server: an application that services requests from clients (need not know clients)

Client/Server System: an application that is built from client and server applications

Typical application area: distributed multi-user (business) information systems

Why Client / Server?

- multiple users want to share and exchange data
- first: attempt: shared file-server
- problem: scalability to ±10 due to contention for files and volume of data-transfer

Typical application with 3 layers

data management: storing, retrieving and updating data
2-tier C/S Example: Thin Client

Thin Client C/S: largest part of processing at the server-side

Typical for legacy systems or simple network devices
Heavy load on network and server
(most processing power at client side not used)

2-tier C/S Example: Thick Client

Thick Client:
significant processing at the client-side

Server is essentially manager of database transactions
E.g. banking ATM systems
System management is more complex

2-tier C/S Model Evaluation

Advantages:
- allows sharing of data between multiple users
- reduce network load
- allows multiple concurrent users
- scalable upto about 100 clients (over a LAN)

Disadvantages:
- difficult to change functionalities of server and client
- scalability of applications is limited by server & network capacity
- changing application logic is difficult if it is distributed over C&S

3-tier Reference Model

An even broader organizational scope of sharing and exchanging data requires coordination across multiple applications and databases

The complexity of the middle tier ranges from reactive with little intelligence (e.g. resource management and interconnection services) to active with much intelligence (active enforcement of global constraints and coordination of activities across applications e.g. workflow)

Advantage of business logic-tier:
changes to business are localized
(compared to intertwining with application logic)

3-tier C/S Model

Technical Implications
The sharing of multiple data sources by multiple applications requires advanced mechanisms for:
- Consistency/Integrity: e.g. locking, scheduling (queueing; prioritization)
- Performance:
  e.g. replication, scheduling, load-balancing, process monitoring

Advantages:
- 3-tier C/S caters for heterogeneous platforms
- scalable upto about 1000 clients
- allows easier reallocation of application/business logic
- allows integration of multiple data-sources
- enables transactions across multiple data sources
- easier version-management of applications
- less vulnerable than centralized computing

Disadvantages:
- difficult to implement and maintain
- difficult to guarantee security
- …
The term “N-tier” is used in different ways:

- To distinguish C/S systems with multiple servers (and possibly multiple middleware systems) from systems with a single server (then called 3-tier).
- To denote a middle-layer that may be separated over multiple intermediate nodes/platforms
- To denote a system where the middle-tier is a middleware layer (Corba, DCOM, EJB) that connects arbitrary applications (rather than DB’s only).

Client / Server Summary

Applications are split into tasks (C/S reference model)

- Task are mapped on platform where they are most efficiently handled
  - presentation layer on client
  - data management and storage on a server
  - possible intermediate platforms for transaction multiplexing and global coordination

With the aim of obtaining
- scalability: changing number of clients
- interoperability: client may use data from multiple sources

Overzicht

- Architectuur intro
- Client/Server architecturen
- UML-modellen van aantal architectuur stijlen
- Definitie van architectuur
- Architectuur evaluatie:
  - Kwantitatieve methoden, metrieken
    - vb Siemens/VDO
  - Kwalitatieve aanpak, bv gebaseerd op scenario’s

Overzicht

- Architectuur intro
- Client/Server architecturen
- UML-modellen van aantal architectuur stijlen
- Definitie van architectuur
- Architectuur evaluatie:
  - Kwantitatieve methoden, metrieken
    - vb Siemens/VDO
  - Kwalitatieve aanpak, bv gebaseerd op scenario’s

Other Architectural Styles

- Blackboard
- Layers
- Pipes-and-filters
- Broker
- …

What is Software Architecture?

Classic Definitions 1

An architecture is the set of significant decisions about
- the organization of a software system,
- the selection of the structural elements and their interfaces by which the system is composed, together with their behaviour as specified in the collaborations among those elements,
- the composition of these structural and behavioural elements into progressively larger subsystems,
- the architectural style that guides this organization

Booch, Rumbaugh, and Jacobson
What is Software Architecture?

Classic Definitions 2
The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.

David Garlan and Dewayne Perry
April 1995 IEEE Transactions on Software Engineering

What is Software Architecture?

Definition 3
Architecture of software is a collection of design decisions that are expensive to change.

Alexander Ran, Nokia Research
September 2001 European Conference on Software Engineering

“The things that are fixed”

What is Software Architecture?

Definition 4
The fundamental organization of a system embodied by its components, their relationships to each other and to the environment and the principles guiding its design and evolution.

IEEE Standard P1471 Recommended Practice for Architectural Description of Software-Intensive Systems

Types of Architectures

Single product
→ Future-proof with respect to a particular product

Product family – e.g. TVs / Telephones
→ Instantiation of many related products
→ Stable with respect to a restricted set of Quality attributes
→ Future-proof with respect to a product type

Architectural Dimensions

Functional dimensions
- Structure
  - Functionality
  - Component interfaces
- Dependability
  - Performance
  - Timeliness
- Local/Global Behavior
  - Control modes
  - Activities/Transactions
- C&S protocols

- Composability
- Interoperability
- Reusability

Non-Functional dimensions
- General
  - Scalability
  - Flexibility
- Maintainability
- Reusability
- Openness
- User-Friendliness
- Cost price

- Cost prize
- Reusability
- Interoperability
- Composability

Positioning Architecture

The question: The answer: Implementation: Deployment:

- Requirements
- Architecture
- Design
- Executable

- Features
- Use cases
- Dependability
- Timing
- Reliability
- Security
- Quality
- Standards
- Etc.

- HL-Design
  - Components
  - Interfaces
  - Interactions
- Styles
- Constraints
- Guidelines
- Reuse
- Etc.

- Decomposition
  - Algorithms
  - Data structures
  - Distribution
  - Scheduling
  - Recovery
  - Language
  - Encryption
  - Etc.

- Memory allocation
- Dynamic
  - Instantiation
- Call stacks
- Garbage
collection
- Machine code
- Etc.
Architecture is making decisions

The life of a software architect is a long (and sometimes painful) succession of suboptimal decisions made partly in the dark.

- You will not have all information available
- You will make mistakes, but you should learn from them
- There is no objective measure for ‘goodness’

Architecture Evaluation

Basically two approaches to analyze architecture:
• Quantitative: How much …?
  Measuring techniques, including:
  metrics, simulations, prototypes
• Qualitative: What if …?
  Questioning techniques, including:
  scenario’s, questionnaires, check-lists

Metrics

How good is the software?
How much does it cost?
When will it be finished?

A metric is a function : SW \( \rightarrow \) numbers

Objectives:
• predict qualitative properties in a quantitative way
• improve controllability and thereby productivity and predictability
• reducing risk

Example of Measuring Approach to Architectural Analysis

Mapping Software on Hardware Architecture

Focus on performance: real-time responses

Example: car navigation system of Siemens/VDO)
Design Space for Embedded Systems

- Communication Templates
- Computation Templates
- Scheduling/Arbitration

Which architecture is better suited for our application?

Architectures

- Architecture # 1
  - LookUp
  - RISC
  - EDL
  - DMA
  - PCI
  - WD

- Architecture # 2

Design Space Exploration

- Respect conflicting criteria for instance:
  - performance measure
  - cost measure

- Critical issues
  - choose “right” abstractions
  - estimation techniques

Why Performance Analysis?

- Application
- Architecture
- Mapping
- Estimation

This process takes place on several levels of abstraction.

Example Packet Processing in Networks

- IP Backbone Network

Problem Specification – Application

- Flows
- Task – Reso
- Resources

Problem Specification

- Flows
delay

- Task – Reso
cost

- Resources
**Problems in Performance Analysis**

- Distributed processing of applications on different resources
- Interaction of different applications on different resources

**Combining Everything – Scheduling Network**

**Investigating possible solutions**

How does Exploration work?

Use, e.g., Evolutionary Algorithms

**When to use Evolutionary Algorithms**

If the problem is...

diffuse + complex

and...

if multiple optimization criteria are involved.

**Principles of Evolution**

- Selection
- Crossover
- Mutation

**Applying the Principles of Evolution**

Generation n+1

Evaluate Fitness

Selection

Mutation

Crossover
Design Space Exploration

Case Study: **Car Navigation System**
(Siemens/VDO)

- Car radio with integrated navigation system
- Several tasks may execute concurrently in a multi-processor architecture
- User interface needs to be responsive
- Traffic messages (TMC) need to be processed in a timely way, especially important for traffic warnings

User Interface

System Architecture
Task 1: Change Audio Volume

Task 2: Lookup Destination Address

Task 3: Receive TMC Messages

Basic Scheduling Choices

We use pre-emptive fixed priority scheduling for all components and applications:

- Highest Priority: Change Volume
- Medium Priority: Address Lookup
- Lowest Priority: Receive TMC

Architecture Alternatives

- Bus connection
- Point-2-point connections
- Radio / Navigation on one CPU
- Radio / MMI on one CPU
- All tasks on one CPU
Calculus to Analyze Architecture

Analysis – End-to-End Delay

Analysis – Sensitivity Analysis

Overzicht

- Architectuur intro
- Client/Server architecturen
- UML-modellen van aantal architectuur stijlen
- Definitie van architectuur
- Architectuur evaluatie:
  - Kwantitatieve methoden, metrieken
    - vb Siemens/VDO
  - Kwalitatieve aanpak, bv gebaseerd op scenario’s

Scenario-based techniques

( SAAM, ATAM, SBAR, .. )

In ATAM 3 types of scenarios:

- Use case (user interaction with system)
- Growth (typical anticipated changes)
- Exploratory (extreme changes, to stress)

There are also heuristics to evaluate quality attributes using scenario’s; e.g. investigate impact of change scenarios on architecture