Course Description

The course introduces modern approaches to enterprise system integration.

Its objective is to survey principles and methods of software architecture in an enterprise environment.

In addition to technical aspects, the course covers organizational aspects of enterprise system integration, including architecture governance and Business-IT alignment.
Scope of the course: The long and winding road ...

- Capability-driven, data-driven, process-driven design
- Architectural Styles
- Middleware & Infrastructure
- Message oriented middleware
- WS
- REST
- JEE
- .NET
- Software Frameworks
- Spring Roo
- Analysis & Design
Scope of the course

- Microservices
- REST
- Architectural Styles
- Design patterns
- Agile practices
- Middleware & Infrastructure
- MOM
- REST
- Docker
- Service and API design
- Analysis & Design
- Domain-driven design
- Spring boot
- Software Frameworks
- Spring boot
- Domain-driven design
- Analysis & Design
- Service and API design
Structure of the Course

• 14 Lectures
  ◦ 3 guest lecturers (practitioners)

• 14 Practicals

• Project
  ◦ Presentations on 26 May
  ◦ Report due on 26 May

• See details on the Wiki pages:
  ◦ https://courses.cs.ut.ee/2017/esi

• Make sure you check the message board!
Grading

• 10 Assignments: 20 points
  ◦ 4-8 hours per homework ~ 70 hours

• Project: 20 points (~ 30 hours)
  ◦ 10 points for the product
    ◦ Soundness of design and architectural choices
    ◦ Scope of the solution (how much was implemented?)
    ◦ Degree of heterogeneity of the system
    ◦ Functional correctness/validation (does it work?)
  ◦ 5 points for written documentation
  ◦ 5 points for presentation

• Exam: 60 points
People

• Luciano García-Bañuelos <luciano.garcia at ut.ee>
  ◦ Lectures on
    ◦ Architectural Styles & Middleware
    ◦ Domain-Driven Design
    ◦ Service Design & Enterprise Integration Patterns
    ◦ Containerization

• Marlon Dumas <marlon.dumas at ut.ee>
  ◦ Service Analysis & Design

• Orlenys López-Pintado & Mykhailo Dorokov
  ◦ Lab assistants
Part I

PROBLEM STATEMENT AND INTRODUCTORY SCENARIO
The problem at glance

- Users want to execute business functions that span multiple applications
- Requires disparate applications to be connected to a common integration solution
- However:
  - No two applications are alike
  - Change is inevitable
Why change is important?

Information Technology

Change and innovation

Enables

Yields

Business Value

Yields

Index Group (1982)

INTRODUCTION

LUCIANO GARCÍA-BAÑUELOS

9
Enterprise system scenario

Information system of a small hotel

Three functional areas:

◦ Guest management: reservations, check-in (including room assignment), charges to room (e.g., room service), check-out, closing of bill

◦ Keeping track of room availability, status – how many rooms still available for a given night, forecasting how many guests will stay beyond their reservation, forecasting occupancy of hotel at a future date

◦ Scheduling room cleaning & maintenance: managing maid & maintenance staff, cleaning rooms after late check-out

• Can you see dependencies across functional areas?
Enterprise system scenario

**Change # 1:** A restaurant is opened for hotel guests and external customers:

- External customer pays for meal directly

- Hotel guest has option to pay for meal directly or charge to room
  - In either case, hotel likes to keep a record of the guest eating in the restaurant, if guest is willing to reveal his room number
  - Restaurant offers breakfast for guests

→ Restaurant billing system has touch-points with hotel guest system
Change # 2: A loyalty program is introduced:

• Set up new system to keep track of customers who are in loyalty program
• Loyalty program has multiple touch-points with hotel system and restaurant system, e.g.,
  ◦ At reservation time, inquiring if customer is a member of loyalty program, offering promotions, giving priority for room allocation to loyalty club members
  ◦ At check-in, inquire if customer is in loyalty program, if not offer sign-up, if yes offer better room
  ◦ Offer discounts at restaurant
  ◦ Offer discounts on other services at check-out and record length of stay and points earned
Enterprise system scenario

**Change # 3:** Hotel is bought out by a franchise:

- Reservations can be made either through the franchise system or by calling the hotel
  - Room availability have to deal with this dual sourcing
  - Forecasting & pricing outsourced to franchise’s system
- Loyalty program merged into franchise’s program
  - Hotel no longer operates the loyalty program, but must interoperate with loyalty program system of franchise
  - Benefits of existing loyal customer must be honored
  - The franchise loyalty program gives points for all expenses paid, whereas the original loyalty program looked only at nights stayed in hotel
- Room cleaning/maintenance still managed locally
Part II

EVOLUTION OF ENTERPRISE SOFTWARE
The Anatomy of an Enterprise System

Application layers

- Users interact with the system through a **presentation layer** (aka **user interface** or UI)

- The **application logic** (aka business logic) determines what the system actually does:
  - Enforces business rules
  - Coordinates business processes

- The **data access layer** facilitates the access to persistent data manipulated by the application logic.
  - Includes access to databases, search engines, document managers and/or a file system.
Tiers or Layers?

• N-tier architectures aim at splitting the application in components into different tiers/layers

• Tiers refer to **physical distribution** of components
  ◦ Components can be executed over a collection of computers

• Layers refer to **logical separation** of components
  ◦ Layered architecture: Communication should only happen with the contiguous layers
  ◦ Promotes reuse and logic independence (components should be seamlessly replaceable)

⇒ The number of tiers in the system somehow reflects the evolution of software architectures w.r.t. distribution
# Evolution of computation

<table>
<thead>
<tr>
<th>60’s</th>
<th>70’s</th>
<th>80’s</th>
<th>90’s</th>
<th>00’s</th>
<th>10’</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM mainframes</td>
<td>IBM, DEC Mini-comp.</td>
<td>PC, DOS Mac, MacOS</td>
<td>Windows Internet</td>
<td>Web 2.0 Mac OS X</td>
<td>iOS, Android HTML5</td>
</tr>
<tr>
<td></td>
<td>Unix, VAX</td>
<td>WANs &amp; LANs</td>
<td>Web &amp; HTTP</td>
<td>Windows XP</td>
<td>SaaS, PaaS, IaaS</td>
</tr>
<tr>
<td>Monolithic applications</td>
<td>Dumb terminals</td>
<td>Client-Server</td>
<td>CORBA &amp; Browsers</td>
<td>Application servers</td>
<td>Cloud computing</td>
</tr>
<tr>
<td>Batch processing</td>
<td>Time-sharing</td>
<td>Relational DBMSs</td>
<td>CGI</td>
<td>SOA &amp; ROA</td>
<td>Microservices</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NoSQL</td>
</tr>
</tbody>
</table>

INTRODUCTION LUCIANO GARCÍA-BAÑUELOS
1-tier architecture

- All layers are bundled in a monolithic entity
- Typical "mainframe" architecture
  - Users access the system through dumb terminals
  - All computation happens in a single computer

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>No context switching in the control flow</td>
<td>Limited scalability due to restrictions in the number of processors</td>
</tr>
<tr>
<td>Centralization eases resource management/sharing</td>
<td>Oftentimes the code is platform specific, limiting portability</td>
</tr>
<tr>
<td>Code highly optimisable</td>
<td>Intertwined code hindering maintenance</td>
</tr>
</tbody>
</table>
With the introduction of computer networks, computation started to be physically distributed.

Application layers are distributed depending on the computing power of clients:
- Thin clients execute only presentation layer
- Fat clients execute both presentation and application logic layers

The concept of API makes its appearance.

Notable example: Database management systems
- The separation of data access layer promotes logical independence, reducing the impact of replacing a database technology on the presentation application logic layers.
2-tier architecture (2/2)

• With the arrival of PCs, the presentation layer moved to the client
  ◦ Smartphones?

• With this approach, it is possible to have multiple presentation layers
  ◦ Text (console) application
  ◦ Graphical user interfaces (e.g., Java swing)
  ◦ Web applications (e.g., HTML5, Javascript, etc.)

• Web as the universal platform for computing?
  ◦ Google’s Chromium OS
3-tier architecture: The middleware arrives

- Middleware is a level of indirection between clients and other layers
  - Simplifies the design of client applications by reducing the number of interfaces
  - Encapsulates integration logic and global application logic
  - Locates resources, accesses them, and integrates results (mediates between application logic/data access layers)
N-tier architecture

• N-tier architectures result from connecting several 3-tier systems and/or adding a layer to allow clients to access the system through a Web server (“Web layer”)

• The Web layer is hosted in a Web application server: a middleware accessible through the Web.

• Web application servers are taking also parts of the functionality of traditional middleware – the boundary between the Web layer and the middleware is blurred.
Flexibility versus Performance

• The more boxes, the more modular the system: more opportunities for distribution and parallelism; more extensibility points.

• The more boxes, the more arrows: more connections need to be maintained, more coordination is needed. Complexity increases.

• The more boxes, the greater the number of context switches and intermediate steps to get to the data. Performance degrades.

• System designers try to balance the flexibility of modular design with performance demands.

There is no problem in system design that cannot be solved by adding a level of indirection. There is no performance problem that cannot be solved by removing a level of indirection.
Back to the ... future?

Product catalog
User manager
Shopping cart
Payment
Shipping

Application server
The GILT experience

we sell every day at noon
Once upon a time ...
... breaking apart the monolith

Monolithic Java App; huge bottleneck for innovation.

Hidden linkages; buried business logic

2011: java, scala, loosely-typed services

teams focused on business lines

lots of duplicated code :(

Large loosely-typed JSON/HTTP services

Adrian Trenaman, VP Engineering, Gilt
Microservices Dublin Meetup, Feb 2015
... and the story continues

2015: micro-services
Microservices: A matter of scale ...

A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers

A microservices architecture puts each element of functionality into a separate service...

... and scales by distributing these services across servers, replicating as needed.
Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

-- Melvyn Conway, 1967