Distributed Applications

Software Engineering 2017
Alessio Gambi - Saarland University

Based on the work of Cesare Pautasso, Christoph Dorn, and other sources
ReCap
Software Architecture

A software system’s architecture is the set of principal design decisions made about the system.

N. Taylor et al.

<table>
<thead>
<tr>
<th>Abstraction</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualization and Representation</td>
<td>Quality Attributes</td>
</tr>
</tbody>
</table>
Every system a software architecture has

- Prescriptive Architecture
- Descriptive Architecture

What designers want:

- Intent
- System Artifacts
- Realization
- Recovery
Design

• Architectural Styles
• Architectural Patterns
• Building Blocks
  - *Software Components*
  - *Component API/Interfaces*
  - *Software Connectors*
3-Tier Architecture

Presentation

Business logic

Data source

Web Browser

Front End

App Server

Back End
Distributed Applications
Client/Server

- Many clients, active, close to users
- One server, passive, close to data
- Single point of failure, scalability
- Security, scalability
Distribution and Lifecycle

In distributed applications the lifecycle of remote objects is disjoint from the local ones. We must explicitly design the lifecycle of those remote entities.
Distribution and Lifecycle

In distributed applications the lifecycle of remote objects is disjoint from the local ones. We must explicitly design the lifecycle of those remote entities.

<table>
<thead>
<tr>
<th>Static and Lazy instances</th>
<th>Leasing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per-request instances</td>
<td>Pooling</td>
</tr>
<tr>
<td>Client-dependent instances</td>
<td>Passivation</td>
</tr>
</tbody>
</table>
Remote object instances exist independently of any clients. They last as long as their container (server).
Lazy Instances

Instantiate object upon first request
Save computational resources
Per-request Instances

Each request processed by a fresh instance
Provide max logical isolation (but high cost)
Requests from the same client processed by the same instance (but there might be a one-to-many mapping)

Remote objects extend client logic and share its state
Leasing

Avoid removal of per-client objects when not used by periodically renew the lease.
Maintain a (possibly dynamic) set of generic objects to serve clients requests

Clean up state before returning to the pool
Passivation

Save resources by freezing “per-client” objects

Objects are reactivated upon first request
Remote invocations can be either synchronous or asynchronous. For asynchronous invocations we must handle the evolution of the distributed state across the nodes.

<table>
<thead>
<tr>
<th>One-way Patterns</th>
<th>Two-way Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fire and Forget</td>
<td>Poll Object</td>
</tr>
<tr>
<td>Sync with Server</td>
<td>Callback</td>
</tr>
</tbody>
</table>
Fire and Forget

Best effort (or nobody cares) semantics
Sync with Server

Requestor ensures that the request correctly arrived to server (but not processed)

Delivery confirmation semantics
Poll Object (or Future)

Local stub on client’s machine checks if results are ready
Callback

Execute code whenever the remote request returns
Publish/Subscribe

- Subscription to queues or topics
- Loose coupling
Pub/Sub vs Event-Driven
Pub/Sub vs Event-Driven

- no specific roles
- local/distributed
Pub/Sub vs Event-Driven

- opposite roles
- mostly distributed

- no specific roles
- local/distributed
Message Bus

- Publish
- Subscribe
- Notify
MOM

Message-Oriented Middleware

Sender

Queue

Receive Message

Receive Reply

Send Message

Queue

Receive Message

Sender

Receiver

Receiver

Receiver

Send Reply

JMS

stormmq

RabbitMQ
MOM
Message-Oriented Middleware

• Processing always on consumer
• Queues provide persistence and decoupling (async)
Reply or don’t reply?

MOMs enable both request-only and request-reply interactions despite sender/receiver do not know each other addresses
Reply or don’t reply?

MOMs enable both request-only and request-reply interactions despite sender/receiver do not know each other addresses

Uniquely identify a request message (ID)

MessageType=REQUEST|REPLY & MessageID = ID

Correlation between the requests and replies
Handling Messages

• Routing
  *Content-based, Dynamic*

• Filtering
  *Message filter*

• Transforming messages
  *Splitter, Aggregator*

• Transforming messages content
  *Normalizer, Content Enricher, Content Filter*

• Transforming message envelope
  *Envelope wrapper*
Content-based Routing

Destination decided using the payload
Dynamic Routing

Destination not fixed but chosen using rules
Message Filter

Remove un-needed messages
Splitter

Decompose a composite message in parts
Aggregator

Use the parts to create a composite message
Content Filter

Filter from a composite message unneeded payload
Content Enricher

Use additional data to augment messages
Normalizer

Route messages to translators which transform them to a common format.
Enveloper Wrapper

Bridged delivery via wrapping messages into other messages
Messaging Bridge

link multiple messaging systems to make messages exchanged on one also available on the others
Pipe & Filter

- Clean separation: filter process, pipe transport
- Heterogeneity and distribution
- Only batch processing, serializable data
- Composability, Reuse
Exclusive Partner Training Opportunity – San Francisco, Jacksonville, Miami, Columbia (MD)
OnRamp Training for Partners – The Next Generation of Web Development on the Microsoft Platform
Coming to Atlantic City – Microsoft Health & Life Sciences Developer Conference
Alabama Tech Events
Jackson, MS Visual Studio Event
Stream

- Send
- Receive
Streaming

- Infinite sequence of messages
  - *simple/primitive, complex*

- Discrete - Messages
  - *stock markets, twitter*

- Continuous - Data
  - *video, audio*
Streaming and Data Analytics

Unicast or multicast communication channels
No discrete unit of interaction (request/response)
Low overhead, but mostly transport/communication
Sync/Async Streams

• Synchronous
  *Time matters (e.g., minimum transfer rate)*

• Asynchronous
  *Sequence matters (e.g., no specific transfer rate)*
Sync/Async Streams

- Synchronous
  *Time matters (e.g., minimum transfer rate)*

- Asynchronous
  *Sequence matters (e.g., no specific transfer rate)*

- Isochronous
  *Time is essence (e.g., both min and max transfer rate)*
Processing Model

Complex Event Processor

Application-specific data processing

Incoming streams

Streaming data m

m3  m2  m1

...  ...  ...

Streaming data s

s3  s2  s1

Output complex messages

Complex stream/multiple streams data processing

Complex Event Processor

EsperTech

S4 distributed stream computing platform
Processing Model

Complex Event Processor

• Event representation
  *POJO, Maps, Object-Arrays, XML, etc.*

• Continuous processing
  *events processes as they arrive and sent to output*

• Listeners and notifications
  *both incoming and outgoing events*

• Domain specific languages (DSL)
  *describe conditions, transformations, etc.*
Specify interests on certain types of events
event-patterns, correlations of events, and more

High-level language SQL-like
standard and new clauses

Streams replace tables; events replace rows
it’s just an analogy

Statements target single and multiple data streams

https://docs.oracle.com/cd/E13157_01/wlevs/docs30/epl_guide/overview.html
EPL
Event Processing Language

- Standard clauses
  SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY

- Re-casted clauses
  INSERT INTO

- New clauses
  RETAIN, MATCHING, OUTPUT
EPL
Event Processing Language

- Retain
  virtual window (constraint amount of data)

- Matching
  sequence of events (logical and temporal operators)

- Output
  control/stabilize the output rate
Event Processing

Incoming events processed through sliding windows
- incremental, one event
- batched, chunks of events

Size of window limits the maximum number of events or the maximum amount of time to keep them
- time
- length

Conditions expressed on the window and events
Sliding Window with Length
Filter & Slide

Incoming Events

- $W_1(500)$
- $W_2(100)$ (Filtered out)
- $W_3(200)$
- $W_4(50)$ (Filtered out)
- $W_5(150)$ (Filtered out)
- $W_6(300)$

Time

Filter: Amount >= 200

Length Window - 5 Events

New Events

- $W_1$

Old Events

- $W_3$

- $W_6$
Slide & Filter

Incoming Events | Length Window – 5 Events | Filter: Amount >= 200 | New Events | Old Events
--- | --- | --- | --- | ---
W₁(500) | W₁ | × | W₁ | 
W₂(100) | W₂ W₁ | × |  | 
W₃(200) | W₃ W₂ W₁ | × | W₃ | 
W₄(50) | W₄ W₃ W₂ W₁ | × |  | 
W₅(150) | W₅ W₄ W₃ W₂ W₁ | × |  | 
W₆(300) | W₆ W₅ W₄ W₃ W₂ W₁ |  | W₆ | W₁
Sliding Window with Time
Batched Window with Time
Service Oriented

- Components outside control
- Standard connectors, precise interfaces
- Interface compatibility problem
- Loose coupling, reuse
<table>
<thead>
<tr>
<th>Components</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tight coupling</td>
<td>Loose coupling</td>
</tr>
<tr>
<td>Client requires library</td>
<td>Message exchanges</td>
</tr>
<tr>
<td>Client / Server</td>
<td>Peer-to-peer</td>
</tr>
<tr>
<td>Extendable</td>
<td>Composable</td>
</tr>
<tr>
<td>Fast</td>
<td>Some overhead</td>
</tr>
<tr>
<td>Small/Medium</td>
<td>Medium/Large</td>
</tr>
<tr>
<td>Buy and install</td>
<td>Pay-per-use</td>
</tr>
<tr>
<td>Local</td>
<td>Remote</td>
</tr>
</tbody>
</table>
Composition/Orchestration

build systems out of the composition of existing ones
Business Processes

- Many alternative notations and languages
  \textit{WSCi, BPML, BPEL4WS, BPSS, XPDL}

- Standard protocols and technologies
  \textit{WSDL, XML, HTTP, JSON, SMTP, FTP, …}

- Two “BIG” players
  SOAP + WS-*, HTTP+RESTFul
Messaging Bridge

link multiple messaging systems to make messages exchanged on one also available on the others
Software Architecture

- Work Flow Engine
- Enterprise BUS
Layered Target Architecture

SOA / BPM architecture

Channels

Business Processes

BPM
Business Rules
Human Workflow

IT Service Orchestration

BPEL PM

IT (integration) Services

SERVICE BUS
DATA INTEGRATION

Systems/Services

Finance
MAINFRAME
SERVICES
DB
PARTNERS
EVENTS

https://www.slideshare.net/kumargaurav66/oracle-soaand-bpm
Say “what what” ? In the cloud!
## Heavy vs Light

#### Old vs New

<table>
<thead>
<tr>
<th></th>
<th>SOAP</th>
<th>REST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A XML-based message protocol</td>
<td>An architectural style protocol</td>
</tr>
<tr>
<td>2</td>
<td>Uses WSDL for communication between consumer and provider</td>
<td>Uses XML or JSON to send and receive data</td>
</tr>
<tr>
<td>3</td>
<td>Invokes services by calling RPC method</td>
<td>Simply calls services via URL path</td>
</tr>
<tr>
<td>4</td>
<td>Does not return human readable result</td>
<td>Result is readable which is just plain XML or JSON</td>
</tr>
<tr>
<td>5</td>
<td>Transfer is over HTTP. Also uses other protocols such as SMTP, FTP, etc.</td>
<td>Transfer is over HTTP only</td>
</tr>
<tr>
<td>6</td>
<td>JavaScript can call SOAP, but it is difficult to implement</td>
<td>Easy to call from JavaScript</td>
</tr>
<tr>
<td>7</td>
<td>Performance is not great compared to REST</td>
<td>Performance is much better compared to SOAP - less CPU intensive, leaner code etc.</td>
</tr>
</tbody>
</table>
Process View

[Diagram showing a comparison between SOAP and REST protocols for web service interactions, including HTTP client, web server, web service implementation, and database operations.]

http://webtechsharing.com/soap-vs-rest/
• Understand the size and complexity of your system and distribute functions and data (lifecycle)

• Embrace diversity: not only RPC, not only sync

• Aim at satisfy your the requirements with the right method (different patterns/styles for different parts)
Additional Readings

- Markus Völter et al.: Remoting Patterns – Foundation of Enterprise, Internet and Realtime Distributed Object Middleware, Wiley Series in Software Design Patterns, 2004
- Martin Fowler’s blog: https://martinfowler.com/
- Fay Chang et al. Bigtable: a distributed storage system for structured data. (OSDI ’06)
- Eric Redmond and Jim R. Wilson: Seven Databases in Seven Weeks – A Guide to Modern Databases and the NoSQL Movement
- Eventual consistency: http://queue.acm.org/detail.cfm?id=1466448