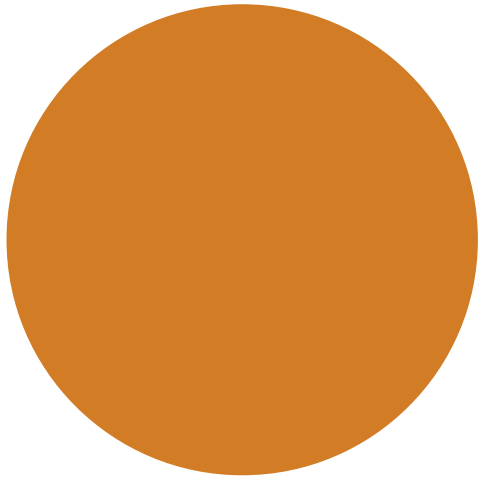
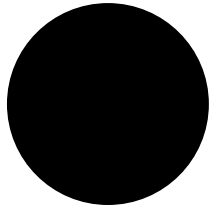
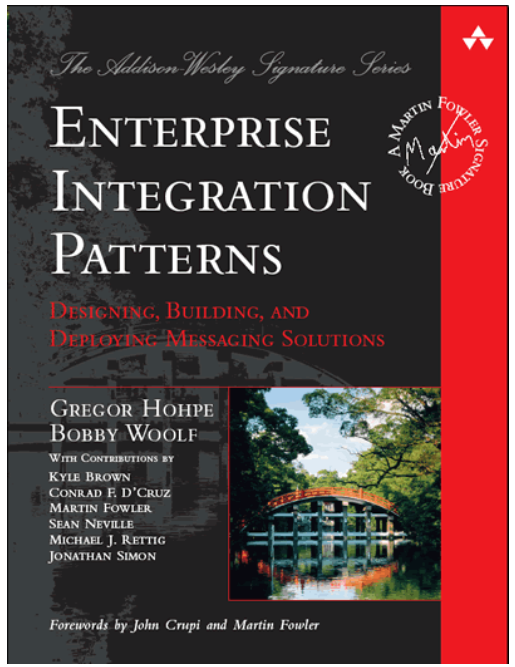



Enterprise Application Integration

Lesson 4
ESB and Web Services

Hanzehogeschool Groningen
University of Applied Sciences

Mind map for this EAI course



Theory on software design principles & patterns

Knowledge of Enterprise Applications

Theory on EAI messaging patterns

Software Components

Integration

Performance as a whole

+

Theory on EAI messaging technologies / standards

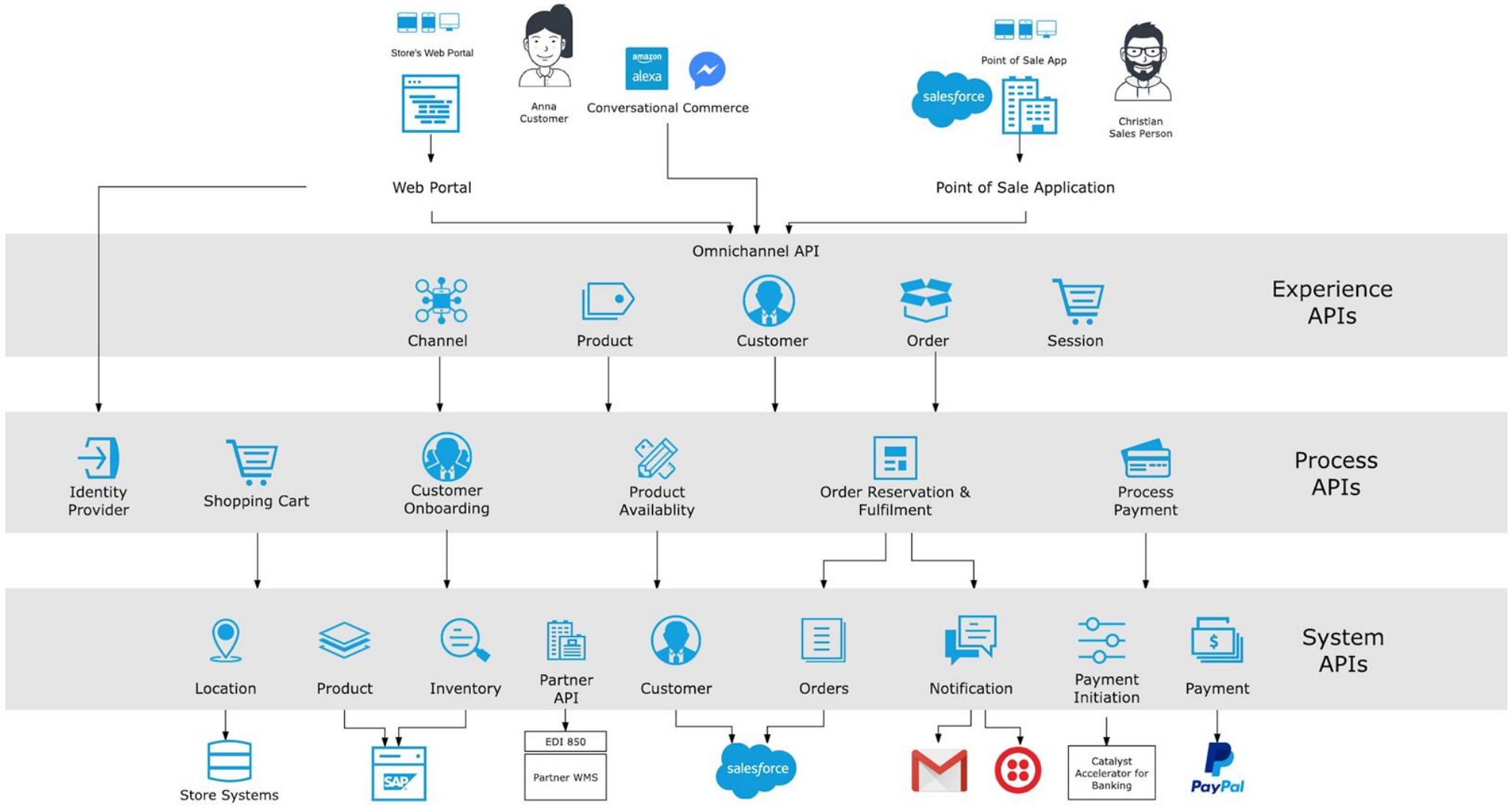
Theory on Integration Styles

Theory on IT architectures (SOA)

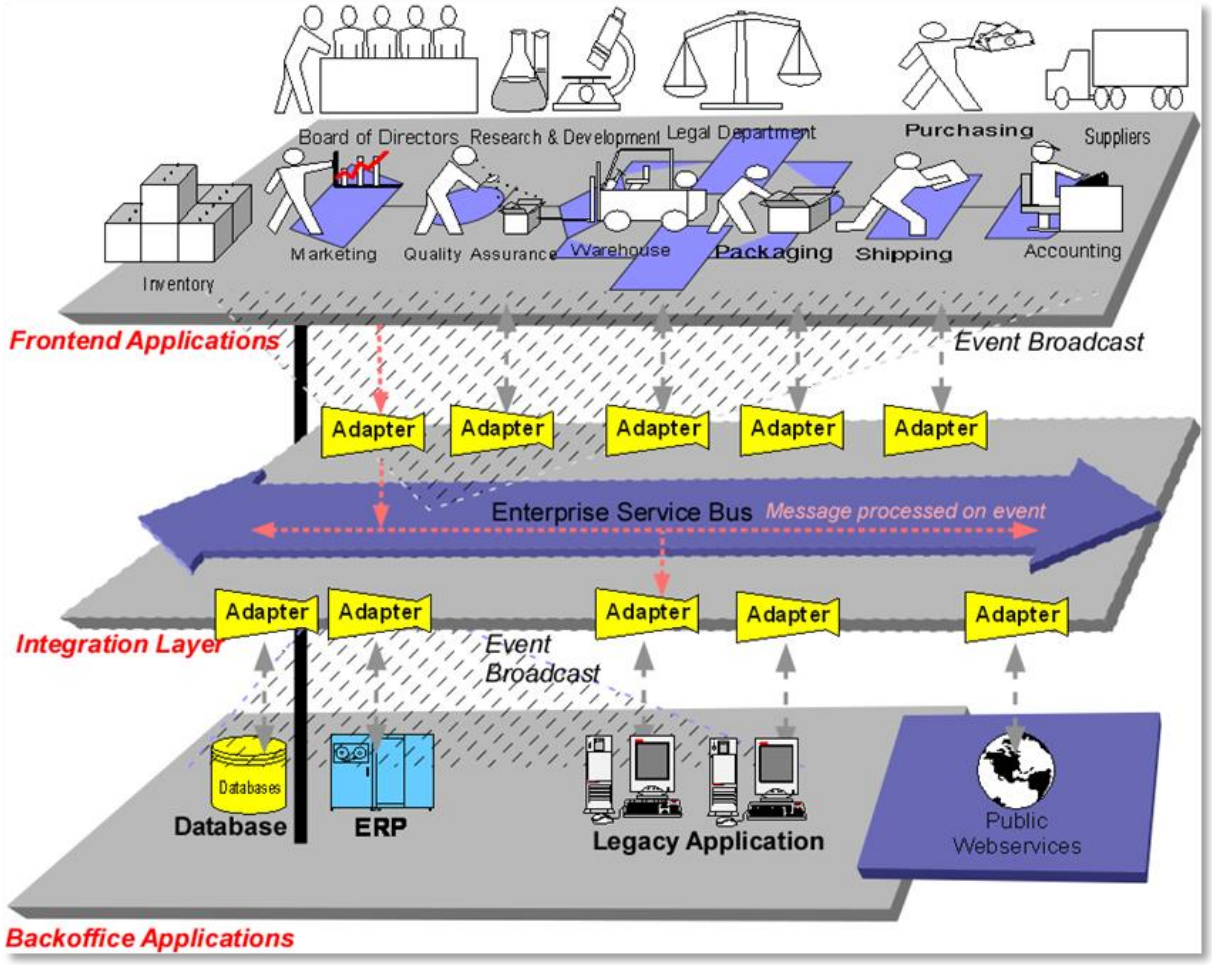
Key Business goal for EAI => Agility which enables short Time to Market

Knowledge on Middleware products from suppliers

Example of integrated business applications /distributed systems

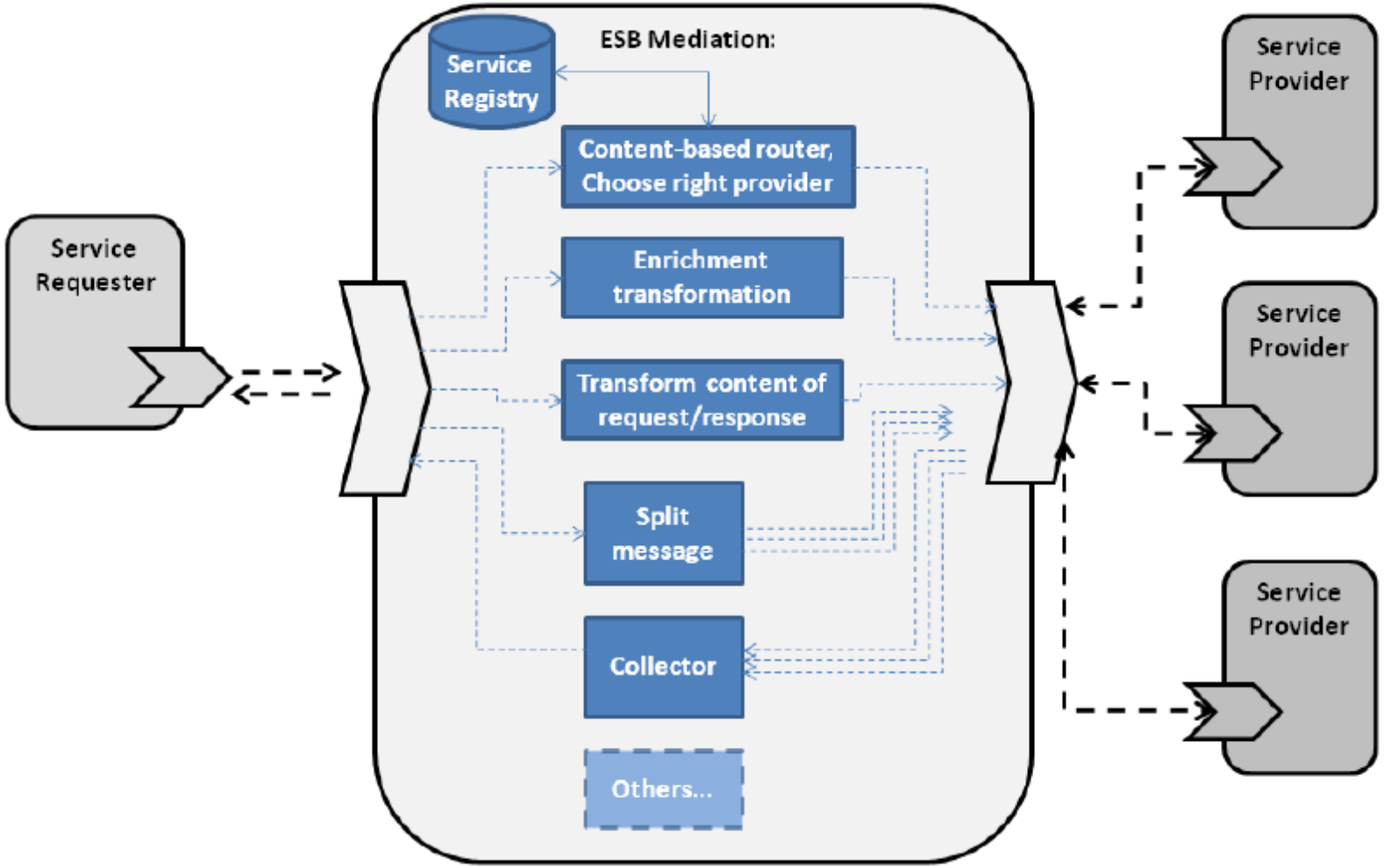


Integration using ESB



Source: https://commons.wikimedia.org/wiki/File:ESB.png

Mediation and Orchestration

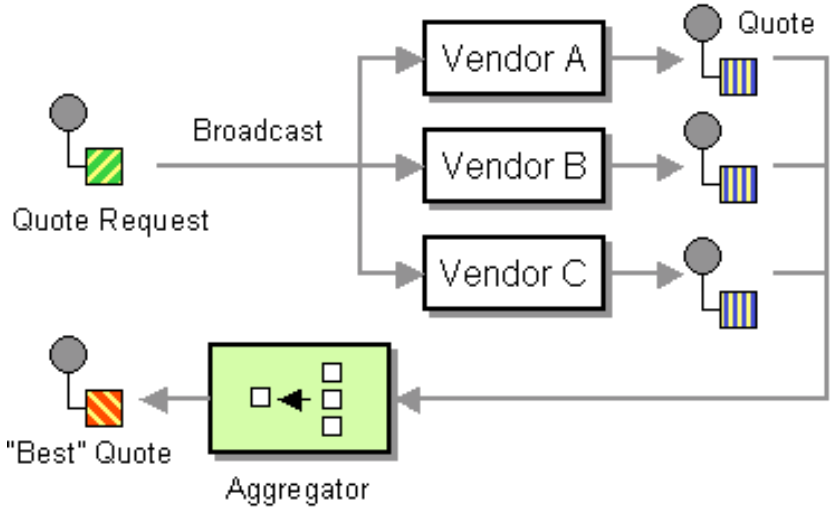


Mediation

Source: https://www.researchgate.net/figure/Mediation-between-Services-by-means-of-ESB-Figure-based-on-18_fig6_254468943

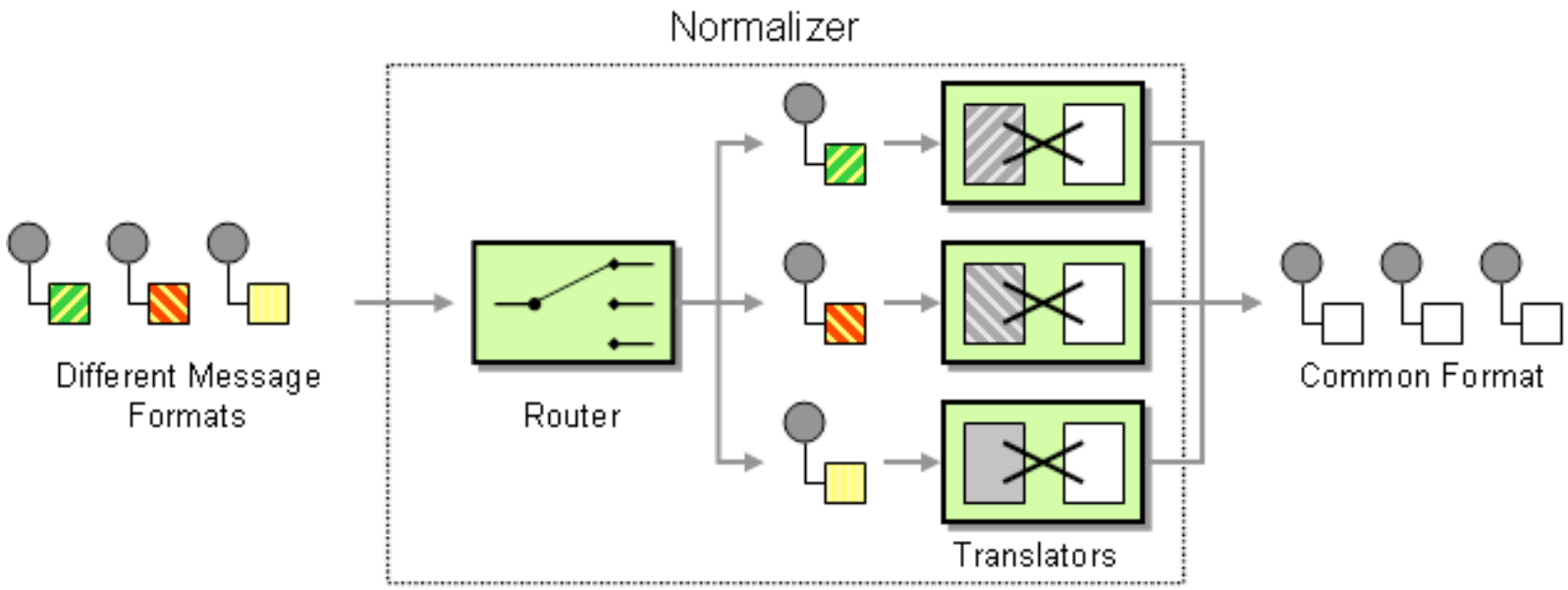
Message Patterns – Message Routing

- ✓ Content-based router
- ✓ Message filter
- ✓ Dynamic router
- ✓ Recipient list
- ✓ Splitter
- ✓ Aggregator
- ✓ Resequencer
- ✓ Compose message processor
- ✓ Routing slip
- ✓ Process manager
- ✓ Message broker
- ✓ Scatter-gather

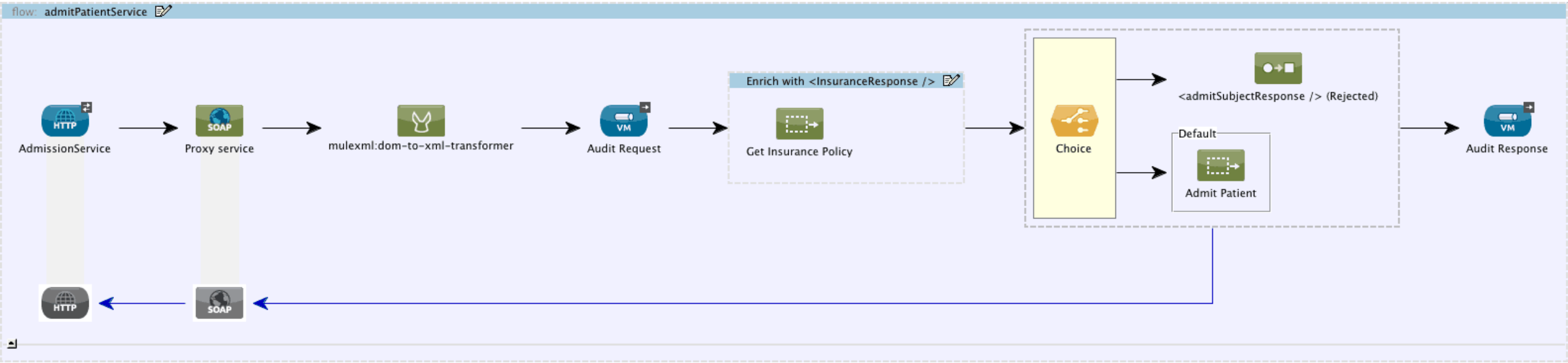


Message Patterns – Message Transformation

- ✓ Envelope wrapper
- ✓ Content enricher
- ✓ Content filter
- ✓ Claim check
- ✓ Normalizer
- ✓ Canonical data model



Mediation and Orchestration

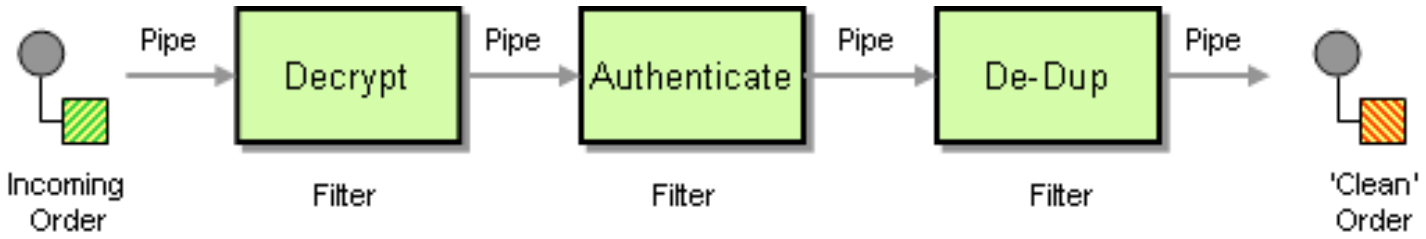


Orchestration

Source: https://blogs.mulesoft.com/wp-content/uploads/2012/11/admitPatientService.png

Message Patterns – Message Systems

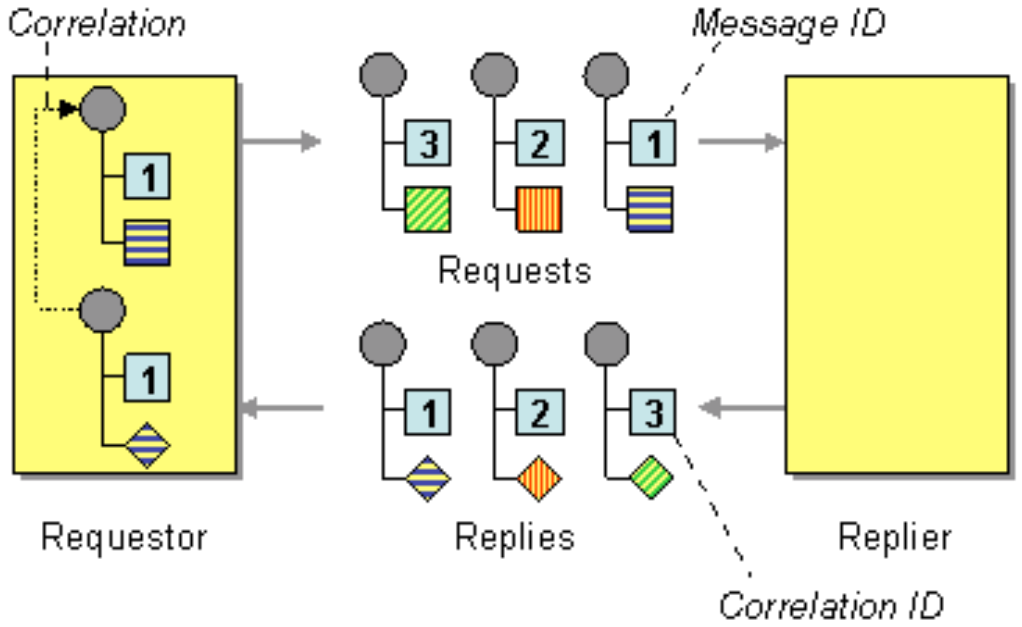
- ✓ Message Router
- ✓ Message Translator
- ✓ Message Endpoint
- ✓ Pipes and Filters



Message Patterns – Message Constructio n

- ✓ Command message
- ✓ Document message
- ✓ Event message
- ✓ Request-reply

- ✓ Return address
- ✓ Correlation identifier
- ✓ Message Sequence
- ✓ Message expiration



Core ESB Features

- Location Transparency
- Transformation
- Protocol Conversion
- Routing
- Enhancement
- Monitoring / Administration
- Security

ESB Advantages

- Lightweight
- Easy to Expand
- Scalable and Distributable
- Soa-Friendly
- Incremental Adoption

When to use an ESB

- How many applications do I need to integrate?
- Will I need to add additional applications in the future?
- How many communication protocols will I need to use?
- Do my integration needs include routing, forking, or aggregation?
- How important is scalability to my organization?
- Does my integration situation require asynchronous messaging, publish/consume messaging models, or other complex multi-application messaging scenarios?

EAI Pitfalls

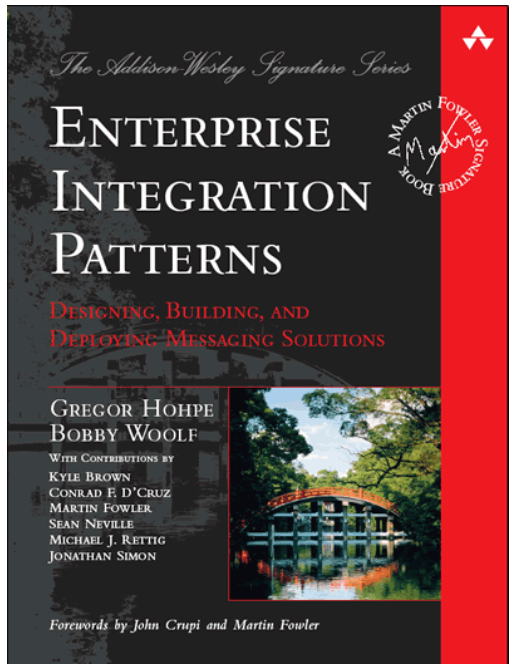
- Constant change
- Shortage of EAI experts
- Competing standards
- EAI is a tool paradigm
- Building interfaces is an art
- Loss of detail
- Accountability

ESB Vendors - More information

Fusion/SOA Suite

Websphere

Mind map for this EAI course



Theory on software design principles & patterns

Knowledge of Enterprise Applications

Theory on EAI messaging patterns

Software Components

Integration

Performance as a whole

+

Theory on EAI messaging technologies / standards

Theory on Integration Styles

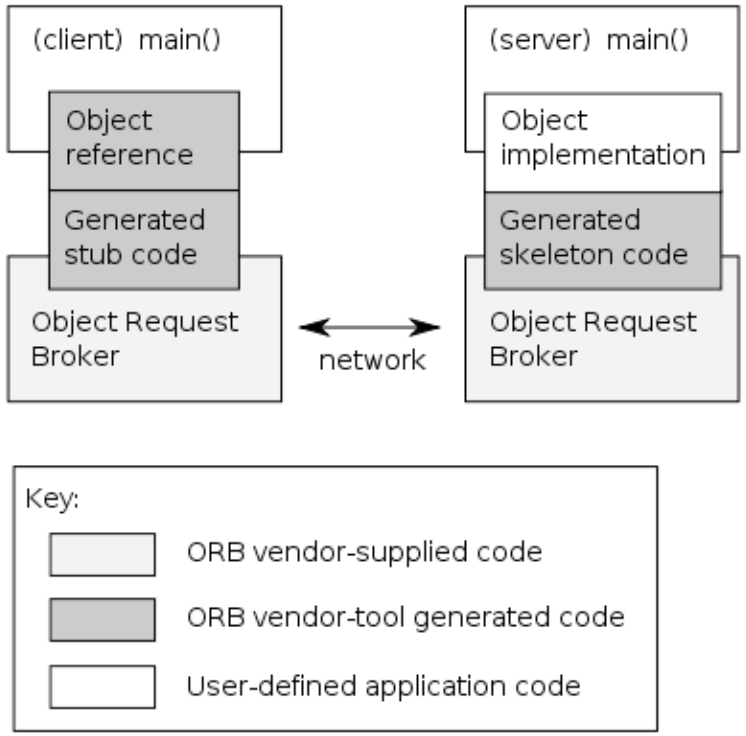
Theory on IT architectures (SOA)

Key Business goal for EAI => Agility which enables short Time to Market

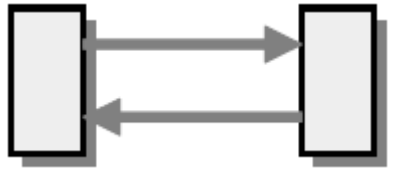
Knowledge on Middleware products from suppliers

Making Services Available – A brief History

- Using resources at a remote location
  - Obtaining data
  - Using resource power
  - Retrieving information
- Remote procedure calls
  - RMI
  - Corba
- Remote procedure calls components
  - Client -Server
  - Stub – Skeleton
  - Broker



Integration style:
Remote Procedure Invocation



Advantages

- Applies the principle of encapsulation
- Shares data and invokes processing of data.
- Each application:
  => maintains the integrity of its own data,
  => can alter the format of its internal data without affecting every other application.
- Applications can provide multiple interfaces to the same data.

Disadvantages

- Each application has to negotiate its interface with its neighbors.
- In some technical environments there are big differences in performance and reliability between remote and local procedure calls => risking slow and unreliable systems.
- Applications are still tightly coupled; doing certain things in a particular order can make it difficult to change systems independently.

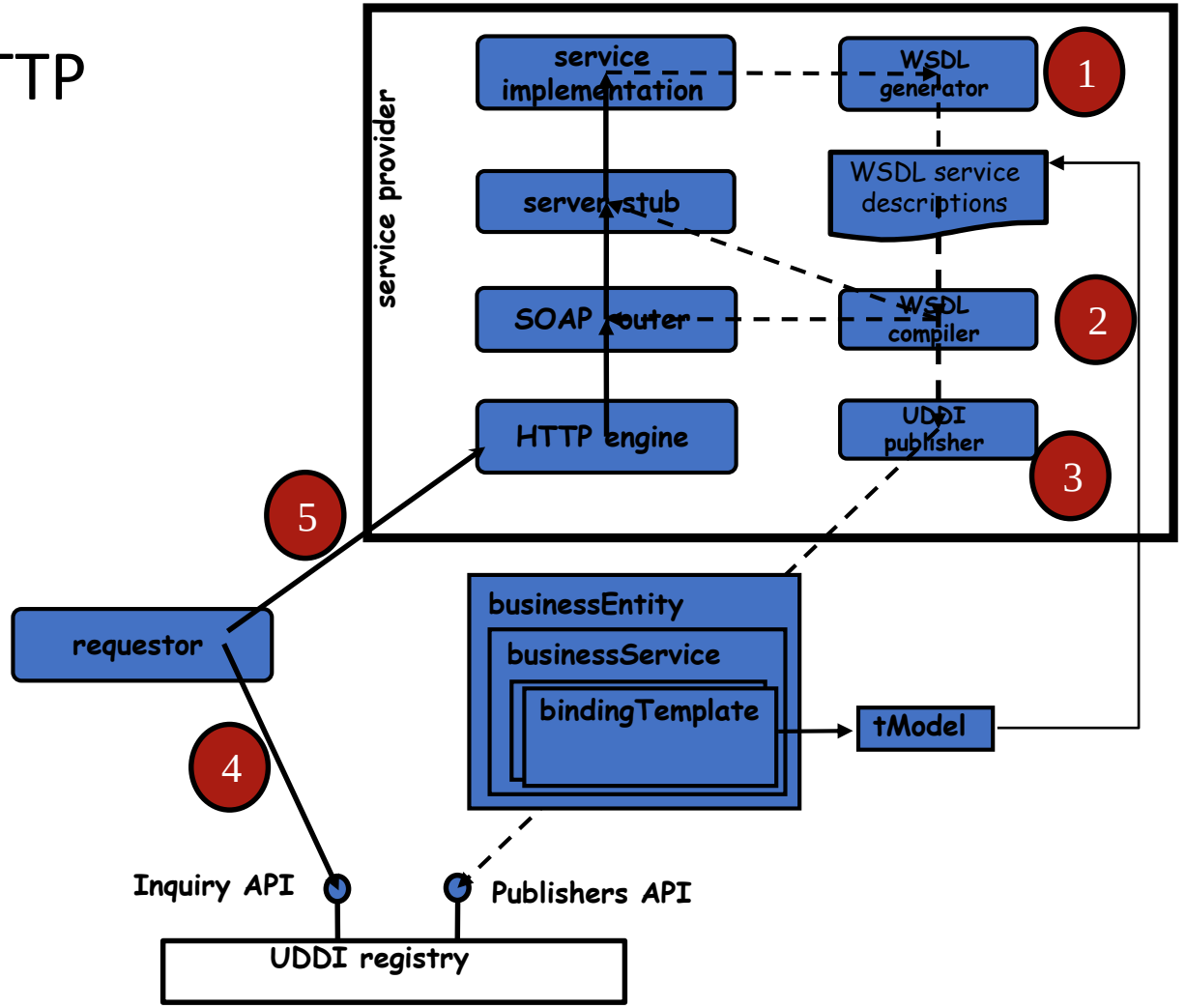
Remote Procedure Invocation is also called Remote Procedure Call (RPC). In fact RPC is the commonly used term nowadays.

Using Web Services over HTTP is a RPC technology.

(quotes from p. 50 etc.)

Making Services Available – A brief History

- Remote procedure calls using HTTP
  - XML-RPC
  - SOAP
- SOAP components
  - Contract (WSDL)
    - Service location
    - Available services
    - Data description
  - SOAP Envelope
    - Header
    - body



Making Services Available – A brief History

- Modern Webservices
  - REST-APIs
  - JSON
  - RAML
  - Swagger – OpenApi
  - Asynchronous

RAML:

- **Basic Information**. How to describe core aspects of the API, such as its name, title, location (or URI), and defaults and how to include supporting documentation for the API.
- **Data Types**. Modeling API data through a streamlined type system that encompasses JSON and XML Schema.
- **Resources**. How to specify API resources and nested resources, as well as URI parameters in any URI templates.
- **Methods**. How to specify the methods on API resources and their request headers, query parameters, and request bodies.
- **Responses**. The specification of API responses, including status codes, media types, response headers, and response bodies.
- **Resource Types and Traits**. The optional use of RAML resource types and traits to characterize resources.
- **Security**. Specifying an API security scheme in RAML.
- **Annotations**. Extending a RAML specification by defining strongly-typed annotations and applying them throughout the specification.
- **Includes, Libraries, Overlays, and Extensions**. How an API definition can consist of externalized definition documents, packaging collections of such definitions into libraries, separating and overlaying layers of metadata on a RAML document, and extending an API specification with additional functionality.