Web Services e Enterprise Service Bus

Architetture di Integrazione per Sistemi Distribuiti

Prof. Fedeli Massimo

IIS Fermi Sacconi Ceci - Ascoli Piceno

Obiettivi della presentazione

- **Panoramica completa su Web Services**
  - SOAP: protocollo di comunicazione XML-based
  - WSDL: linguaggio di descrizione dei servizi
  - UDDI: registro per la discovery dei servizi
- **Architetture di integrazione**
  - Service-Oriented Architecture (SOA)
  - Enterprise Application Integration (EAI)
  - Enterprise Service Bus (ESB)
  - Java Business Integration (JBI)
- **Pattern e best practices**
  - Pattern architetturali e di integrazione
  - Casi d'uso reali con esempi pratici

Differenza tra Servizi Web e Web Services

Servizi Web (Web Applications)

- Interfaccia utente (HTML/CSS/JS)
- Interazione umana diretta
- Browser come client
- Output: pagine web visualizzabili

Web Services

- Interfaccia programmatica (API)
- Integrazione macchina-macchina (M2M)
- Applicazioni come client
- Output: dati strutturati (XML/JSON)

Differenza Chiave

I Web Services sono progettati per l'integrazione automatizzata tra sistemi software, non per l'interazione umana diretta.

Evoluzione del Middleware

Motivazione dell'evoluzione:

- Crescente eterogeneità dei sistemi
- Necessità di attraversare firewall
- Riduzione del coupling tra componenti
- Standard aperti e interoperabili

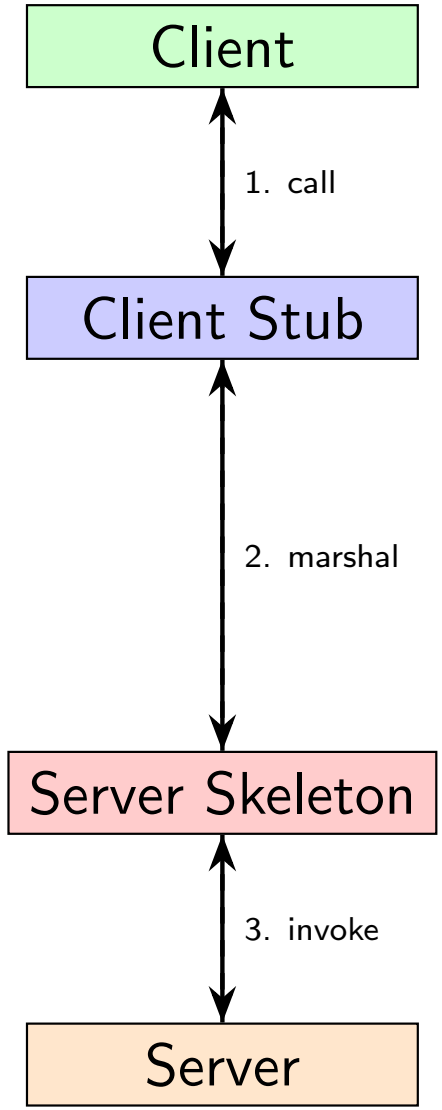
Modello Client-Server e RPC

Remote Procedure Call (RPC)

- Chiamata di procedura remota come se fosse locale
- Trasparenza della distribuzione
- Comunicazione sincrona
- Marshalling/unmarshalling parametri

Limiti principali:

- Accoppiamento stretto
- Problemi con firewall
- Eterogeneità limitata
- Gestione errori di rete complessa



CORBA: elementi essenziali

Common Object Request Broker Architecture

Componenti principali

- **ORB (Object Request Broker)**: middleware per comunicazione oggetti distribuiti
- **IDL (Interface Definition Language)**: linguaggio neutro per definire interfacce
- **Stubs e Skeletons**: proxy client-side e server-side
- **IIOP (Inter-ORB Protocol)**: protocollo di comunicazione tra ORB

Vantaggi:

- Interoperabilità multi-linguaggio (Java, C++, Python, etc.)
- Location transparency
- Object-oriented

Svantaggi:

- Complessità di configurazione e deployment
- Problemi con firewall (porte dinamiche)

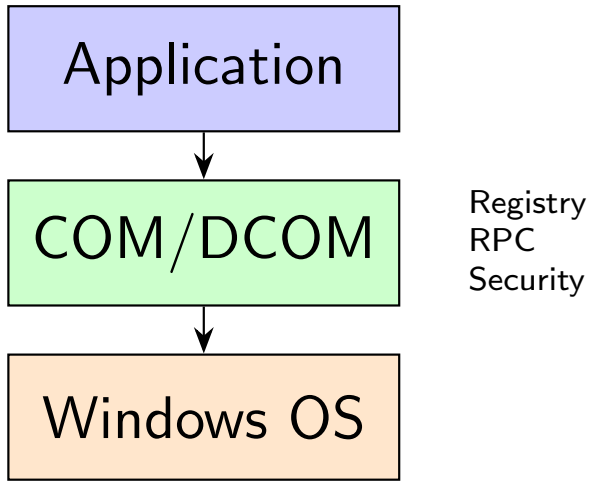
COM/DCOM (.NET)

Component Object Model / Distributed COM
Caratteristiche:

- Tecnologia Microsoft proprietaria
- Integrazione nativa con Windows
- Registrazione componenti nel Registry
- DCOM per distribuzione su rete
- Active Directory per discovery

Limitazioni:

- Piattaforma-dipendente (Windows)
- Complessità di configurazione DCOM
- Problemi di sicurezza
- Difficoltà attraversamento firewall



Evoluzione: COM → DCOM → COM+ → .NET Remoting → WCF

Sistemi Middleware: confronto rapido

Caratteristica	CORBA	DCOM	Web Services
Linguaggi	Multi-linguaggio	Windows-centric	Multi-linguaggio
Piattaforme	Multi-piattaforma	Windows	Multi-piattaforma
Coupling	Stretto	Stretto	Loose
Firewall	Problematico	Problematico	Friendly (HTTP)
Protocollo	IIOP	DCOM RPC	SOAP/HTTP
Discovery	Naming Service	Active Directory	UDDI/WSIL
Standard	OMG	Microsoft	W3C/OASIS
Complessità	Alta	Media	Media-Bassa
Formato dati	Binario (CDR)	Binario	XML (testuale)
Performance	Eccellente	Buona	Media

Conclusione

I Web Services hanno vinto per: **standard aperti**, **firewall-friendly**, **loose coupling** e **interoperabilità**.

XML come metalinguaggio

eXtensible Markup Language
Caratteristiche:

- Metalinguaggio per dati strutturati
- Self-describing (tag personalizzati)
- Human-readable e machine-parseable
- Validazione tramite DTD o XSD
- Separazione contenuto/presentazione

Ruolo nei Web Services:

- Serializzazione dati
- Definizione interfacce (WSDL)
- Messaggi SOAP

```
<?xml versio
<libro>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<titolo>Web␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<autore>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<nome>Mario<
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<cognome>Ros
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</autore>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<anno>2024</
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<prezzo valu
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣45.50
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</prezzo>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</libro>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
```

Validazione XSD:

- Tipi di dati
- Struttura documenti
- Vincoli di integrità

SOAP: panoramica

Simple Object Access Protocol

Definizione

Protocollo XML-based per lo scambio di messaggi strutturati in ambienti decentralizzati e distribuiti. Può essere trasportato su HTTP, SMTP, TCP, JMS.

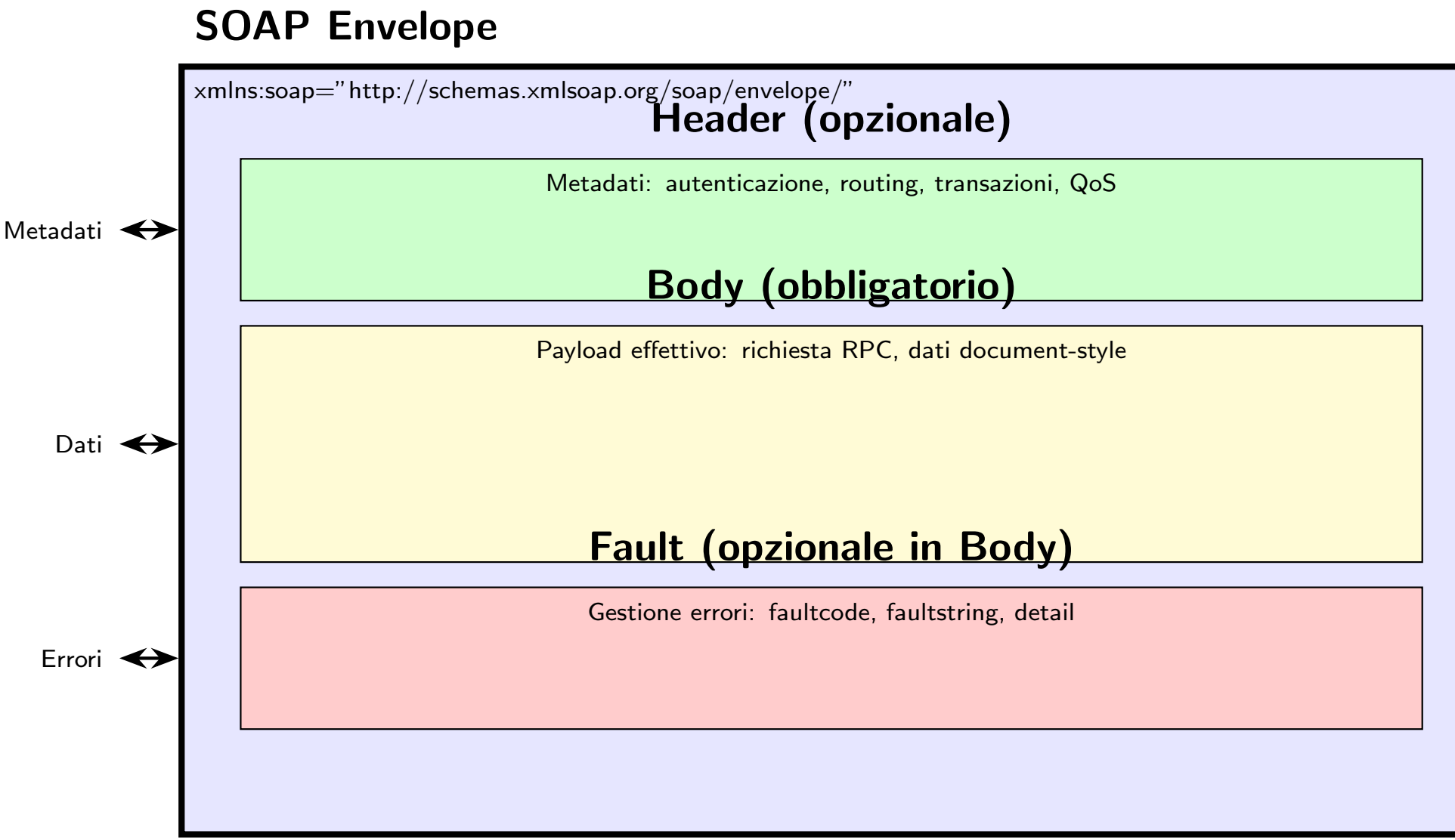
Caratteristiche principali:

- **Stateless**: ogni richiesta è indipendente
- **Estensibile**: header personalizzabili per metadati
- **Neutrale**: indipendente da linguaggio e piattaforma
- **Standard**: specifiche W3C ben definite

Struttura del messaggio:

- **Envelope**: container root del messaggio
- **Header**: informazioni di controllo (opzionale)
- **Body**: payload effettivo del messaggio
- **Fault**: gestione errori e eccezioni

Schema: SOAP Envelope



SOAP: esempio di request

Richiesta GetLastTradePrice per un'azione

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:m="http://www.stock.org/stock">

  <soap:Header>
    <m:Authentication>
      <m:Username>user123</m:Username>
      <m:Password>pass456</m:Password>
    </m:Authentication>
  </soap:Header>

  <soap:Body>
    <m:GetLastTradePrice>
      <m:StockSymbol>AAPL</m:StockSymbol>
    </m:GetLastTradePrice>
  </soap:Body>

</soap:Envelope>
```

Nota: La richiesta viene inviata via HTTP POST all'endpoint del servizio.

SOAP: namespace e binding

Namespace SOAP standard:

- `http://schemas.xmlsoap.org/soap/envelope/`
- `http://schemas.xmlsoap.org/soap/encoding/`

Encoding Style:

- **SOAP Encoding**: regole SOAP per serializzazione
- **Literal**: rispetta esattamente lo schema XSD

HTTP Binding:

Header	**Valore**
Method	POST (tipicamente)
Content-Type	text/xml; charset=utf-8
SOAPAction	"URI-azione" o ""
Content-Length	lunghezza body

SOAPAction header:

- Indica l'intent della richiesta
- Utile per routing e firewall
- Può essere stringa vuota ""

Stili di Interazione SOAP

Document-Style

- Scambio di documenti XML
- Asincrono (fire-and-forget)
- One-way o notification
- Loose coupling
- Validazione tramite XSD

Uso tipico:

- Messaggistica asincrona
- Event notification
- Batch processing

RPC-Style

- Chiamata procedura remota
- Sincrono (request-response)
- Parametri e valori di ritorno
- Tight coupling
- Encoding SOAP tipico

Uso tipico:

- Query database
- Calcoli real-time
- Transazioni immediate

Best Practice

Document/Literal Wrapped è lo stile raccomandato: combina validazione XSD con semplicità RPC.

SOAP Fault: gestione errori

Struttura del Fault element

```
                        <soap:Fault>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<faultcode>soap:Client</faultcode>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<faultstring>Invalid␣Stock␣Symbol</faultstring>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<faultactor>http://www.stock.org/stockquote</faultactor>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<detail>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<e:StockError xmlns:e="http://www.stock.org/errors">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<e:ErrorCode>1001</e:ErrorCode>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<e:Message>Symbol␣XXXX␣not␣found␣in␣database</e:Message>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</e:StockError>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</detail>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</soap:Fault>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
```

Fault codes standard:

- **VersionMismatch**: versione SOAP non supportata
- **MustUnderstand**: header obbligatorio non compreso
- **Client**: errore nella richiesta del client
- **Server**: errore nell'elaborazione server-side

Strategie di gestione: retry, fallback, logging, alert

WSDL: concetti chiave

Web Services Description Language

Scopo

Linguaggio XML per descrivere in modo formale e machine-readable le interfacce dei Web Services: operazioni, parametri, tipi di dati, binding e endpoint.

Struttura a due livelli:
Parte Astratta

- **Types**: schemi XSD per tipi di dati
- **Messages**: definizione messaggi
- **Operations**: operazioni disponibili
- **PortType/Interface**: raggruppamento operazioni

Parte Concreta

- **Binding**: protocollo e formato (SOAP/HTTP)
- **Service**: endpoint URL reale
- **Port**: combinazione binding+indirizzo

Specifica dell'implementazione

WSDL: esempio di types e messages

Definizione Types (XSD Schema)

```
<types>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<schema targetNamespace="http://example.com/stockquote"
                            xmlns="http://www.w3.org/2001/XMLSchema">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<element name="GetLastTradePriceRequest">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<complexType>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<all>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<element name="symbol" type="string"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</all>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</complexType>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</element>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<element name="GetLastTradePriceResponse">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<complexType>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<all>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<element name="price" type="float"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</all>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</complexType>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</element>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</schema>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</types>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
```

```
<message name="GetLastTradePriceInput">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<part name="body" element="tns:GetLastTradePriceRequest"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</message>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<message name="GetLastTradePriceOutput">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<part name="body" element="tns:GetLastTradePriceResponse"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</message>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
```

WSDL: binding e endpoint

Binding: collega PortType a protocollo concreto

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
        <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
```

Service ed Endpoint

```
<service name="StockQuoteService">
    <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
        <soap:address location="http://example.com/stockquote"/>
    </port>
</service>
```

Uso pratico del WSDL

Workflow di utilizzo:

1. **Pubblicazione**: il provider pubblica il WSDL
2. **Discovery**: il consumer trova il WSDL (UDDI, URL, registry)
3. **Generazione stub**: tool automatici creano proxy client
   - Java: wsimport, Axis, CXF
   - .NET: wsdl.exe, svcutil.exe
   - Python: suds, zeep
4. **Invocazione**: il client usa lo stub come oggetto locale
5. **Validazione**: verifica automatica parametri contro schema

Vantaggi:

- Contract-first development
- Type-safety compile-time
- Documentazione machine-readable
- Interoperabilità garantita

UDDI: panoramica

Universal Description, Discovery and Integration

Motivazione

Registry distribuito per pubblicare e scoprire Web Services. Concetto analogo alle "Pagine Gialle" per i servizi web.

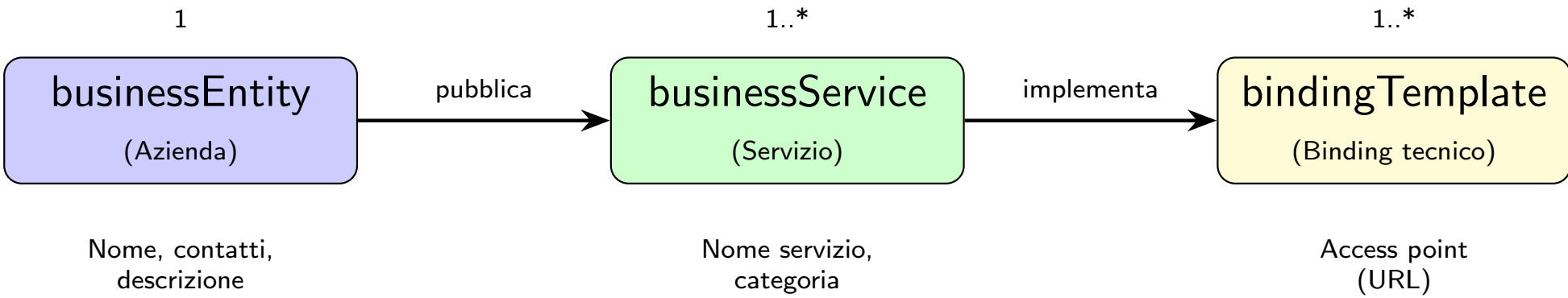
Tre componenti principali:

- **White Pages**: informazioni di contatto business (nome, indirizzo, etc.)
- **Yellow Pages**: classificazione per categoria industriale/geografica
- **Green Pages**: informazioni tecniche sui servizi (WSDL, binding)

Entità del modello dati:

- **businessEntity**: organizzazione che pubblica servizi
- **businessService**: descrizione logica del servizio
- **bindingTemplate**: dettagli tecnici accesso (URL, protocollo)
- **tModel (technical model)**: metadati e specifiche tecniche

Schema: modello UDDI



Operazioni UDDI:

- **Publish API**: registrazione e aggiornamento servizi
- **Inquiry API**: ricerca e recupero informazioni

UDDI: White/Yellow/Green Pages

White Pages

Nome azienda
Indirizzo
Contatti
Descrizione generale

Yellow Pages

Categoria industriale
Settore merceologico
Area geografica
Classificazione

Green Pag

WSDL locatio
Binding tecnic
Protocolli suppor
Parametri acces

Esempio ricerca:

1. Cerco in Yellow Pages: "servizi finanziari" + "Roma"
2. Ottengo lista aziende (White Pages)
3. Per ogni azienda, accedo ai dettagli tecnici (Green Pages)
4. Download WSDL e generazione stub client

WSIL: ispezione leggera dei servizi

Web Services Inspection Language

Motivazione

Alternativa più semplice e decentralizzata a UDDI. Documento XML statico che elenca i servizi disponibili su un server.

Caratteristiche:

- File `inspection.wsil` nella root del web server
- Contiene link a WSDL e altri documenti WSIL
- Non richiede infrastruttura registry centralizzata
- Scoperta gerarchica e distribuita

Confronto con UDDI:

	UDDI	**WSIL**
Centralizzato	Sì	No
Ricerca	Potente	Limitata
Complessità	Alta	Bassa
Uso tipico	Pubblico	Intranet

Situazione attuale: UDDI pubblici dismessi; WSIL usato in ambito

Performance dei Web Services

Fattori che impattano le performance:

Overhead XML:

- Parsing e serializzazione
- Dimensione messaggi (verbosità)
- Validazione schema

Overhead HTTP:

- Latenza connessione TCP
- Header HTTP
- Nessuna connessione persistente (HTTP/1.0)

Overhead SOAP:

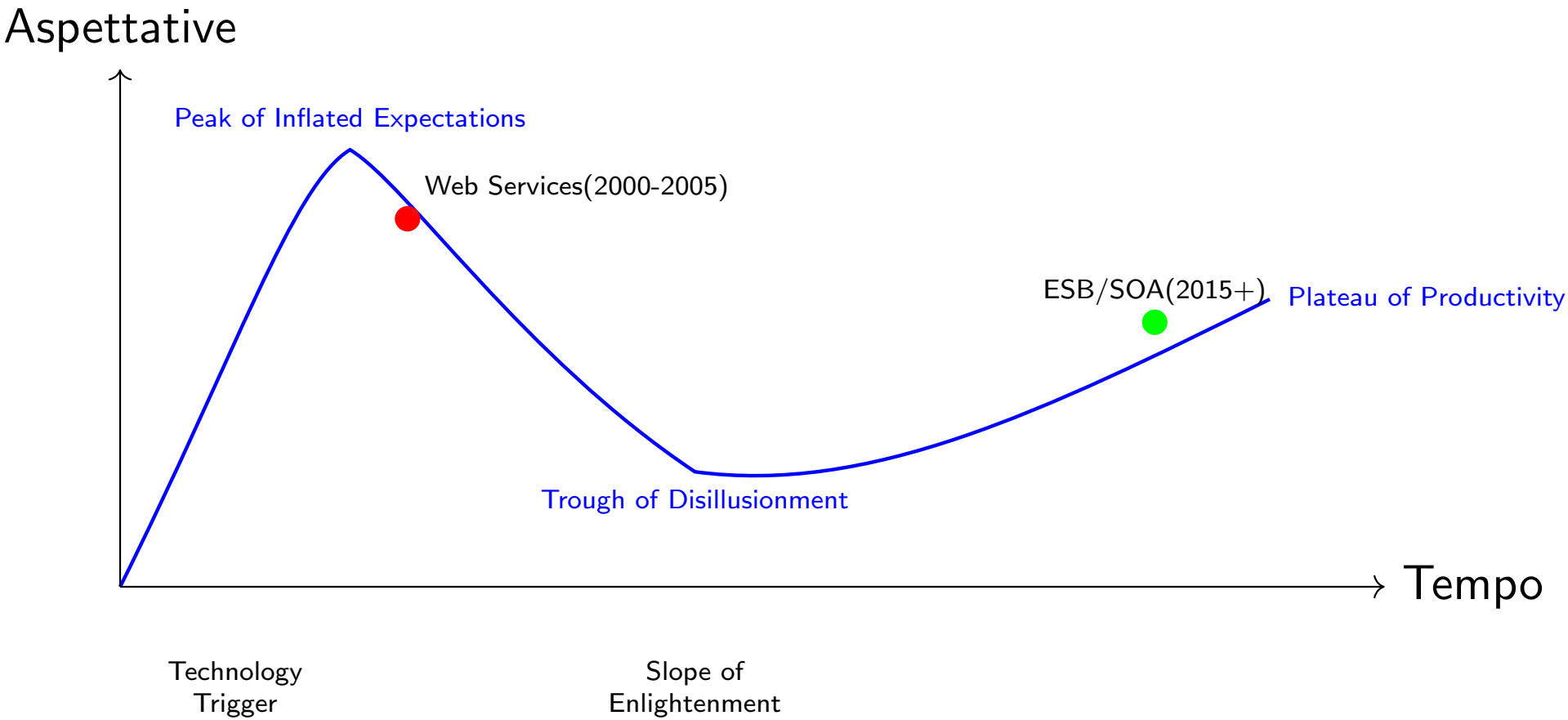
- Envelope wrapping
- Namespace processing
- Encoding/decoding

Strategie di ottimizzazione:

1. **Caching**: response caching
2. **Compressione**: gzip HTTP
3. **Batching**: aggregare richieste
4. **Binary attachments**: MTOM/XOP
5. **Keep-alive**: HTTP/1.1 persistent
6. **Asynchronous**: non-blocking I/O
7. **Load balancing**: distribuire carico

Ciclo di adozione tecnologica

Gartner Hype Cycle



Evoluzione:

- **2000-2005**: hype massimo su Web Services

BPEL4WS: orchestrazione

Business Process Execution Language for Web Services

Definizione

Linguaggio XML per orchestrare Web Services in processi di business complessi. Permette di definire flussi di lavoro che coordinano l'invocazione di multipli servizi.

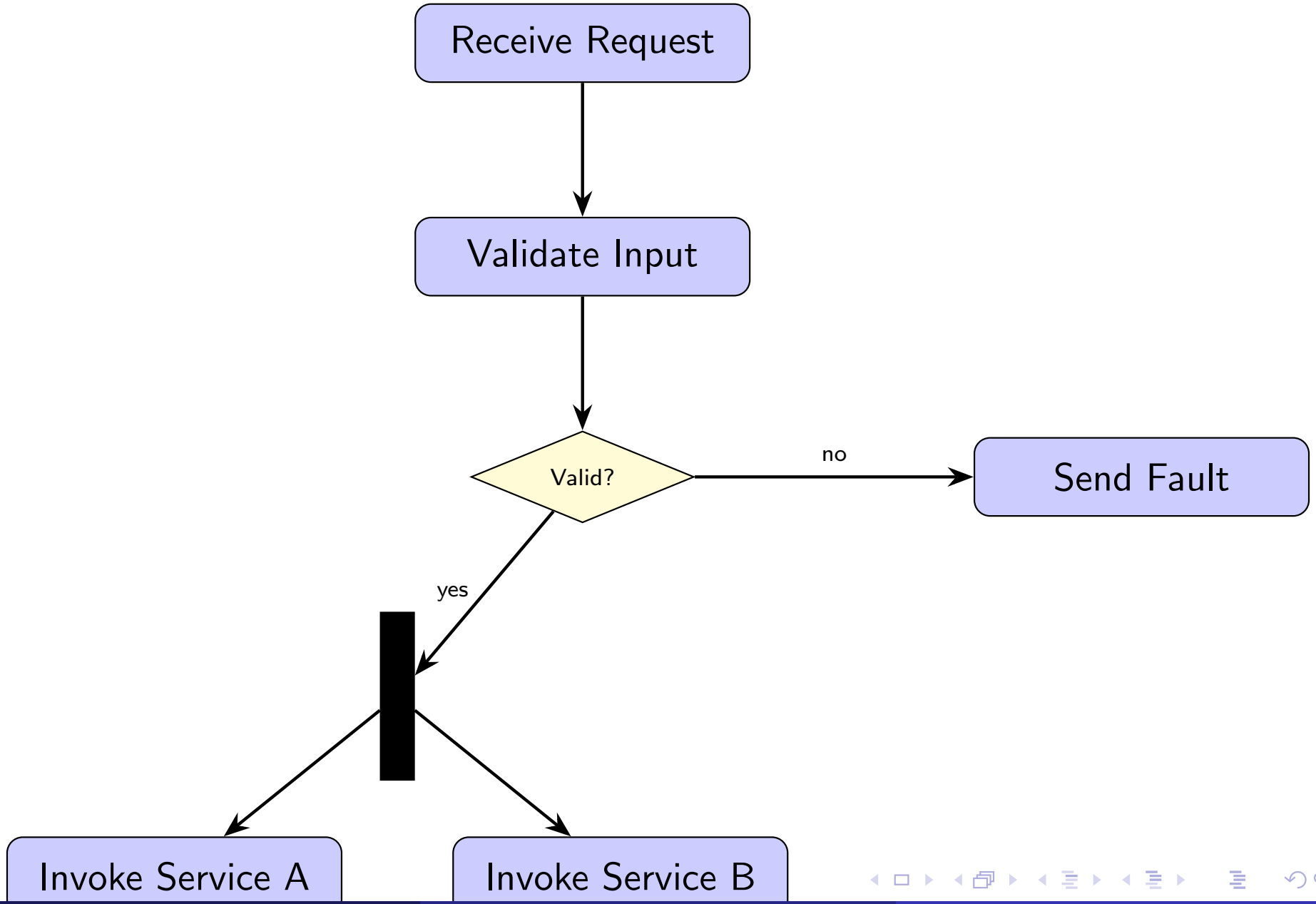
Costrutti principali:

- **Sequence**: esecuzione sequenziale di attività
- **Flow**: esecuzione parallela (fork/join)
- **Switch/If**: decisioni condizionali
- **While/RepeatUntil**: cicli iterativi
- **Invoke**: chiamata a Web Service
- **Receive/Reply**: ricezione richieste e invio risposte
- **Assign**: manipolazione variabili
- **Throw/Catch**: gestione eccezioni

Caratteristiche:

- Basato su WSDL per interfacce

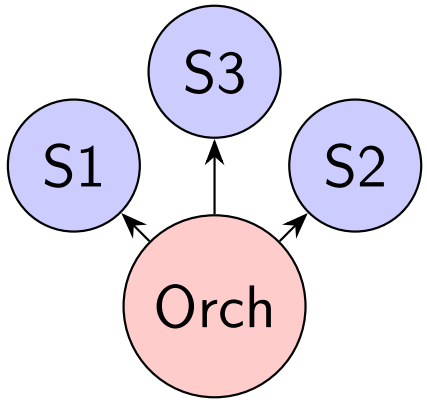
Diagramma: processo BPEL semplificato



Orchestrazione vs Coreografia

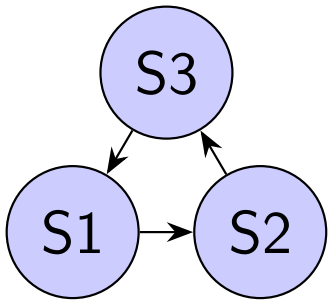
Orchestrazione

- **Controllo centralizzato**
- Un orchestratore coordina
- Visione globale del processo
- BPEL tipicamente
- Single point of control
- Più facile da monitorare



Coreografia

- **Coordinamento distribuito**
- Ogni servizio sa cosa fare
- Peer-to-peer interaction
- WS-CDL, event-driven
- Maggiore resilienza
- Più complesso da debuggare

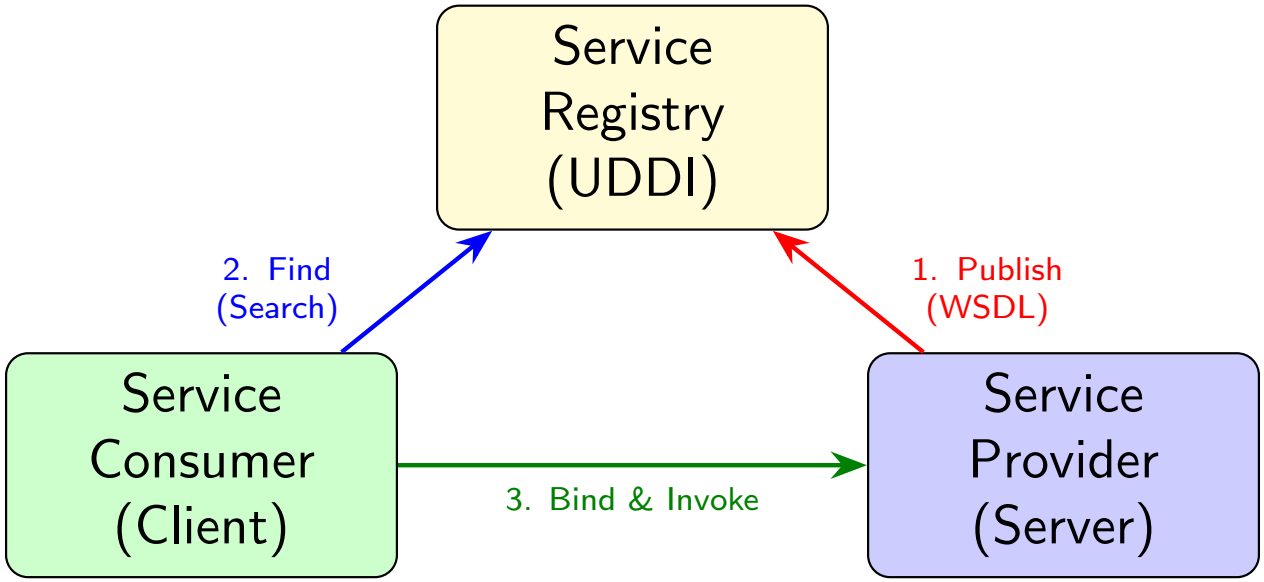


Quando usare

Orchestrazione: processi complessi, controllo fine grained

Integrazione: pattern find-bind-invoke

Il paradigma SOA fondamentale



Fasi dettagliate:

1. **Publish**: il provider registra il servizio nel registry con WSDL
2. **Find**: il consumer cerca servizi per categoria/nome
3. **Bind**: download WSDL e generazione stub/proxy
4. **Invoke**: chiamata effettiva al servizio via SOAP

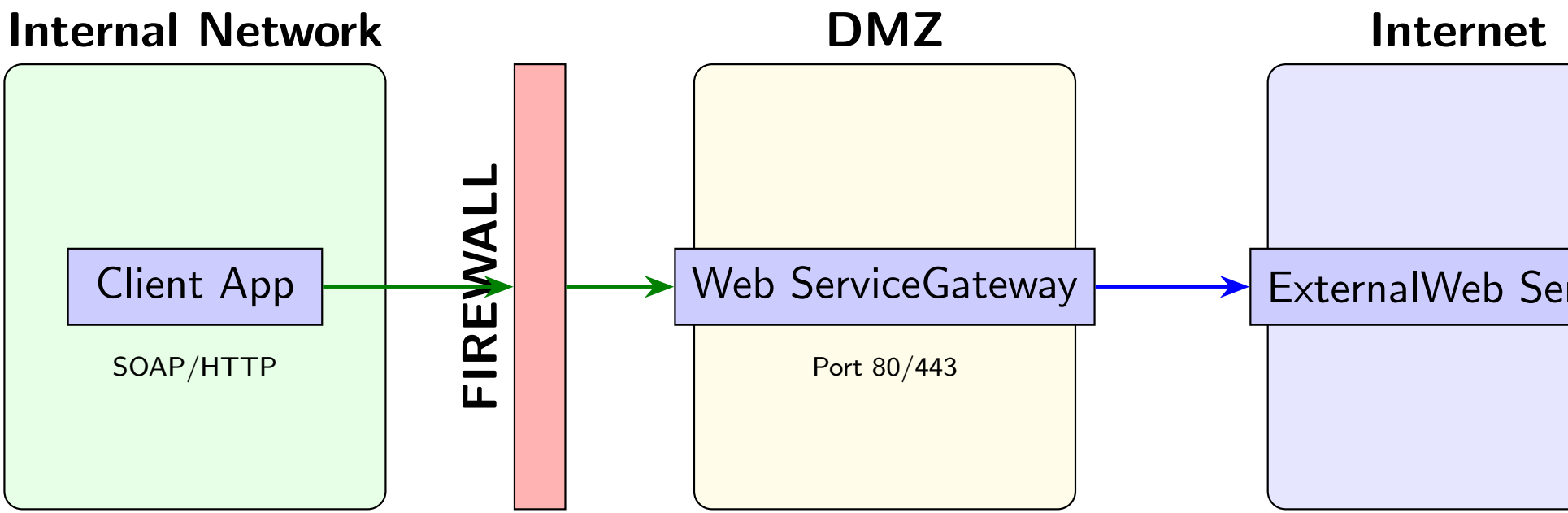
Vantaggi: loose coupling, discovery dinamico, flessibilità

Gateway e firewall

Web Services e sicurezza di rete

Vantaggio chiave

I Web Services usano HTTP/HTTPS (porte 80/443) che attraversano facilmente firewall e proxy aziendali, a differenza di CORBA o DCOM che usano porte dinamiche.



Sicurezza nei Web Services

Livelli di sicurezza

1. **Transport Level Security**
   - HTTPS/TLS: cifratura e autenticazione server
   - HTTP Basic/Digest Auth
   - Protegge solo punto-a-punto
2. **Message Level Security**
   - WS-Security: sicurezza end-to-end
   - XML Signature: integrità e non-ripudio
   - XML Encryption: confidenzialità
   - Security tokens: SAML, X.509, Kerberos

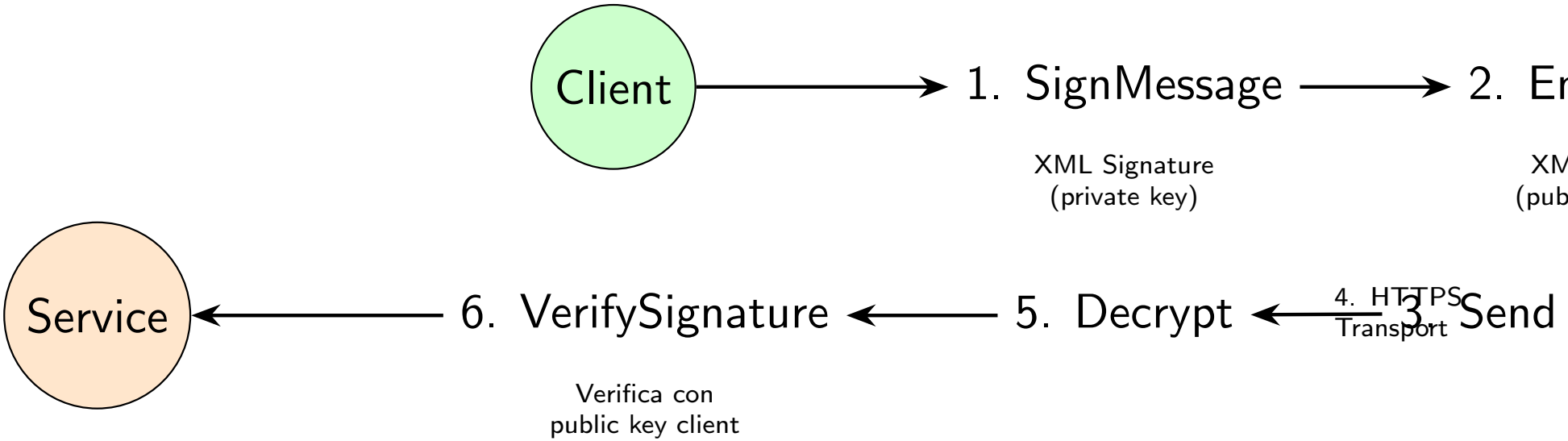
Standard WS-Security:

- **UsernameToken**: credenziali nell'header SOAP
- **BinarySecurityToken**: certificati X.509
- **SAML Token**: asserzioni di autenticazione/autorizzazione
- **Timestamp**: prevenzione replay attacks

Best practices:

- Sempre HTTPS in produzione
- WS-Security per scenari multi-hop

Schema: flusso sicuro con WS-Security



Elementi nel messaggio:

- **wsse:Security** header con token, signature, encryption references
- **ds:Signature**: firma digitale di parti del messaggio
- **xenc:EncryptedData**: parti cifrate del payload

Estensioni e specifiche correlate

WS-* Stack: famiglia di specifiche enterprise

Specifica	Funzione
WS-Security	Sicurezza messaggi (firma, cifratura, token)
WS-Policy	Dichiarazione requisiti e capability
WS-Addressing	Routing, addressing endpoint-neutral
WS-ReliableMessaging	Garanzia delivery, ordering, deduplication
WS-AtomicTransaction	Transazioni distribuite (2PC)
WS-Coordination	Coordinamento processi distribuiti
WS-BPEL	Orchestrazione processi di business
WS-Discovery	Discovery dinamico servizi su rete
WS-Eventing	Publish/subscribe event notification
WS-Trust	Trust model, token issuance

Complessità

Lo stack WS-* è molto potente ma anche complesso. In molti scenari REST/JSON è preferito per semplicità, mentre WS-* resta per enterprise mission-critical.

Composizione e gestione degli errori

Fault Handling in processi distribuiti

Problema

In orchestrazioni che coinvolgono multipli servizi, un errore in un punto qualsiasi richiede strategie di ripristino per mantenere coerenza.

Strategie:

1. **Retry**: tentativi ripetuti con backoff esponenziale
2. **Timeout**: limite temporale per ogni invocazione
3. **Fallback**: servizio alternativo se primario fallisce
4. **Circuit Breaker**: evitare cascading failures
5. **Compensating Transactions**: annullare operazioni già completate

Compensazioni:

- Ogni attività ha un'azione compensativa inversa
- In caso di errore, esecuzione compensazioni in ordine inverso
- Esempio: prenotazione volo + hotel
  - Successo: volo OK, hotel OK
  - Fallimento hotel: compensa cancellando volo già prenotato

EAI: approccio tradizionale

Enterprise Application Integration

Definizione

Insieme di tecnologie e processi per integrare applicazioni enterprise eterogenee (ERP, CRM, SCM, legacy systems) per condividere dati e processi.

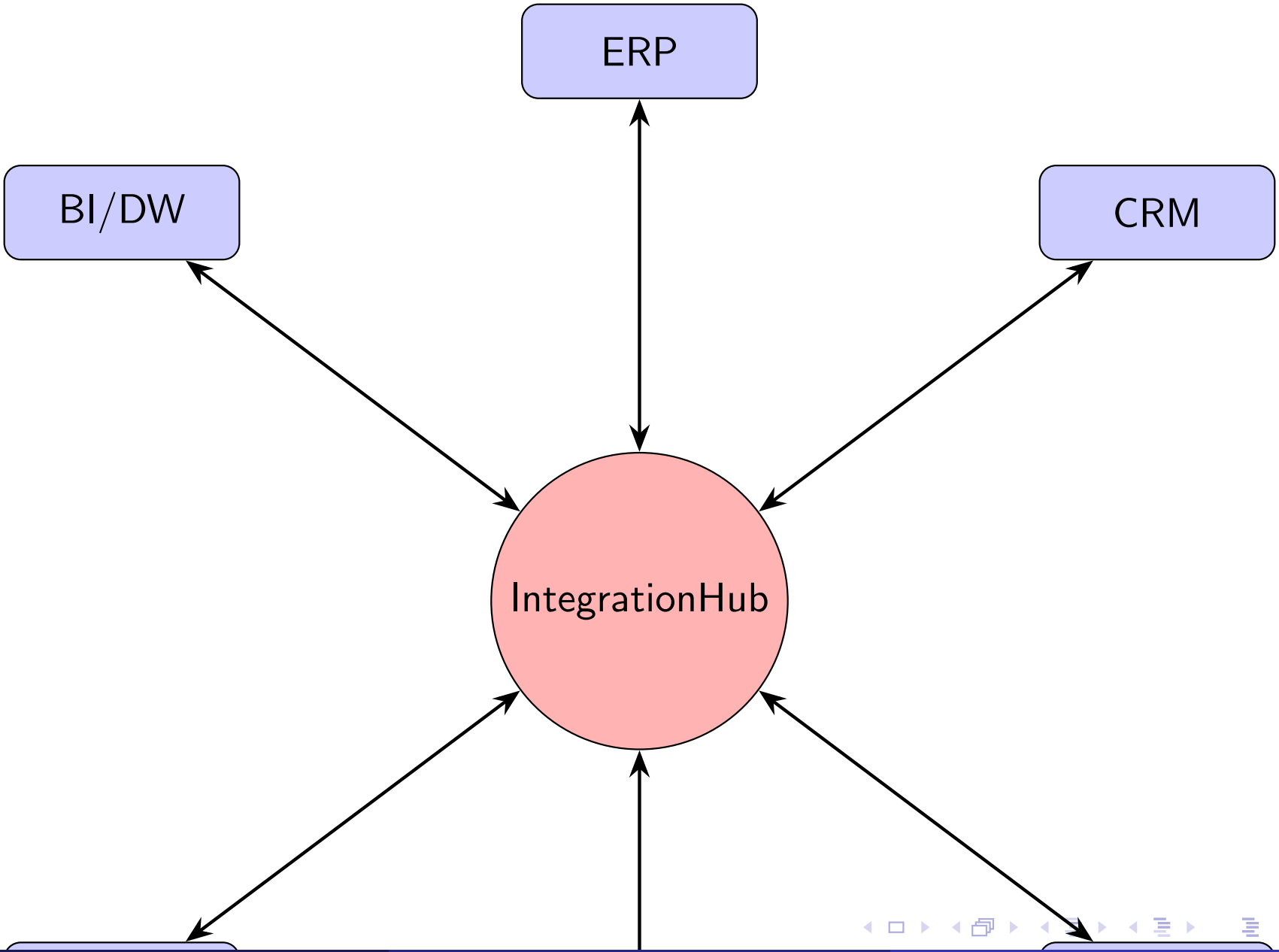
Approcci storici:

- **Point-to-Point**: connessioni dirette tra applicazioni
  - Pro: semplice inizialmente
  - Contro: $O(n^2)$ connessioni, manutenzione impossibile
- **Hub-and-Spoke**: broker centralizzato
  - Pro: centralizzazione, $O(n)$ connessioni
  - Contro: single point of failure, bottleneck
- **Enterprise Service Bus (ESB)**: bus distribuito
  - Pro: scalabilità, resilienza, standard
  - Contro: complessità configurazione

Tecniche EAI:

- **ETL**: Extract, Transform, Load per batch integration

Schema: Hub-and-Spoke



Bus di integrazione

Message Bus Architecture

Concetto

Infrastruttura di comunicazione distribuita (bus logico) a cui tutte le applicazioni si connettono. Il bus gestisce routing, trasformazione e delivery dei messaggi.

Vantaggi rispetto a Hub-and-Spoke:

- **Scalabilità**: aggiunta nodi bus senza colli di bottiglia
- **Resilienza**: nessun single point of failure
- **Distribuzione geografica**: nodi in datacenter diversi
- **Specializzazione**: nodi dedicati per task specifici

Sfide:

- Complessità configurazione distribuita
- Governance e monitoraggio multi-nodo
- Consistenza routing rules
- Debugging più complesso

Tecnologie:

Enterprise Service Bus: definizione

ESB - Enterprise Service Bus

Definizione formale

Architettura middleware che implementa un bus di comunicazione per integrare servizi eterogenei tramite routing intelligente, trasformazione dati, orchestrazione e gestione centralizzata delle policy.

Funzionalità core:

- **Routing**: intelligente, content-based, dinamico
- **Trasformazione**: XSLT, mapping, enrichment
- **Protocol mediation**: SOAP, REST, JMS, FTP, etc.
- **Service orchestration**: BPEL, flow control
- **Security**: autenticazione, autorizzazione, encryption
- **Transaction management**: distributed transactions
- **Monitoring Management**: metriche, logging, audit

Principi architetturali:

- Loose coupling tra servizi
- Location transparency

Caratteristiche principali di ESB

Uniformità di accesso

- Interfaccia uniforme verso il bus
- Nasconde eterogeneità sottostante
- Adattatori per legacy systems

Disaccoppiamento

- Servizi non si conoscono direttamente
- Comunicazione via bus
- Cambiamenti isolati

Monitoring e QoS

- SLA enforcement
- Metriche real-time
- Alert e notifiche

Gestione centralizzata

- Policy management
- Security governance
- Version control

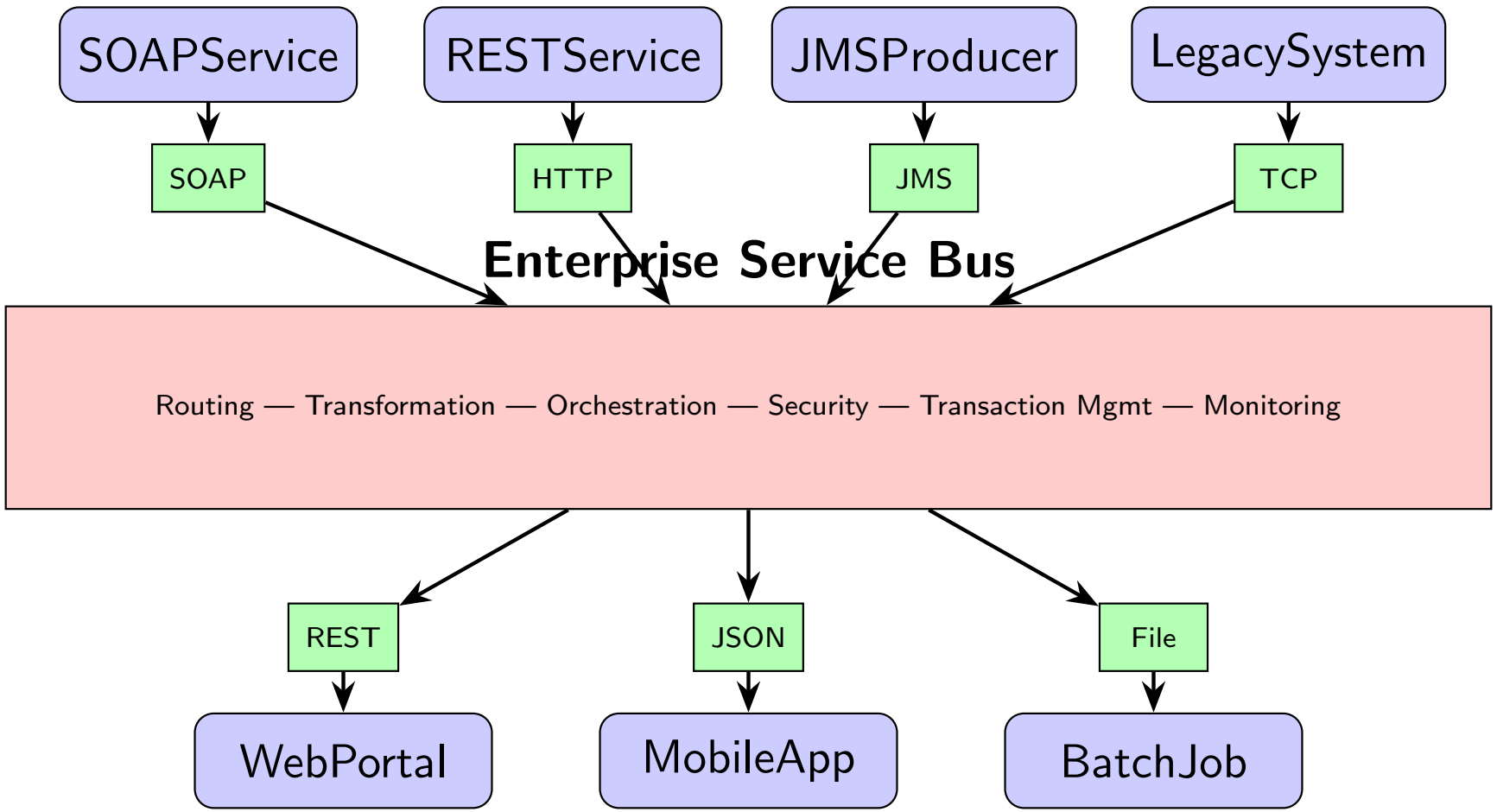
Scalabilità

- Clustering e load balancing
- Horizontal scaling
- Failover automatico

Logging e Audit

- Message logging completo
- Tracciamento end-to-end
- Compliance e audit trail

Schema: architettura ESB

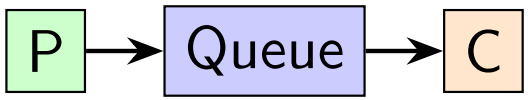


Message Oriented Middleware

MOM - Foundation dell'ESB

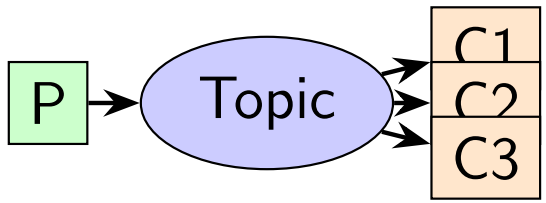
Queue (Point-to-Point)

- Un producer, un consumer
- FIFO processing
- Load balancing tra consumer
- Garanzia di delivery
- Persistenza messaggi



Topic (Publish/Subscribe)

- Un publisher, N subscriber
- Broadcasting
- Event-driven
- Durable/non-durable
- Message filtering



Quality of Service (QoS):

- **At-most-once**: possibile perdita messaggi
- **At-least-once**: possibili duplicati
- **Exactly-once**: delivery garantito una sola volta (costoso)

Normalized Message Router (NMR)

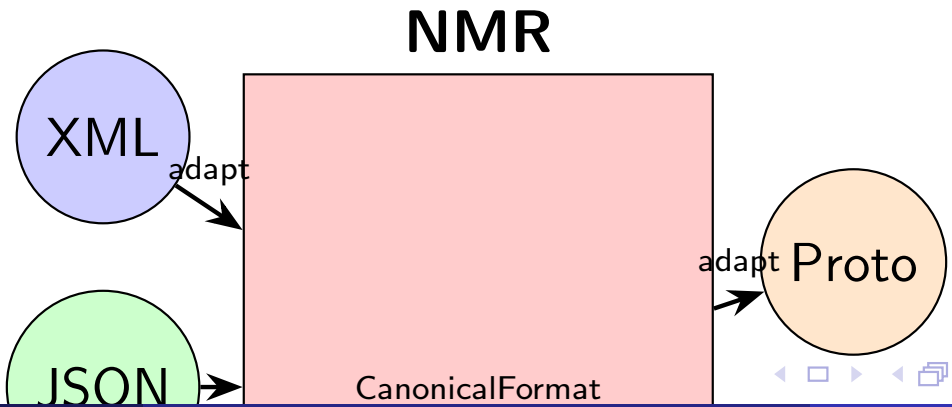
JBI Concept: Normalized Message Router

Principio

I messaggi sono convertiti in un formato canonico neutro (normalized) all'ingresso nel bus. Le trasformazioni tra formati eterogenei avvengono solo ai bordi, non internamente.

Vantaggi:

- **Riduzione complessità**: O(n) adattatori invece di $O(n^2)$
- **Riuso logica**: trasformazioni centralizzate
- **Loose coupling**: servizi non conoscono formati altrui
- **Manutenzione**: cambio formato esterno = 1 adattatore



JBI: Java Business Integration

JSR 208 - Standard Java per ESB

Obiettivo

Definire un'architettura standard e pluggable per integrare componenti di integrazione in Java. Permette interoperabilità tra vendor diversi.

Componenti architetturali:

- **Normalized Message Router (NMR)**: bus interno
- **Service Engine (SE)**: logica di business (BPEL, XSLT, validation)
- **Binding Component (BC)**: protocolli esterni (SOAP, HTTP, JMS, FTP)
- **JMX Management**: amministrazione e monitoraggio

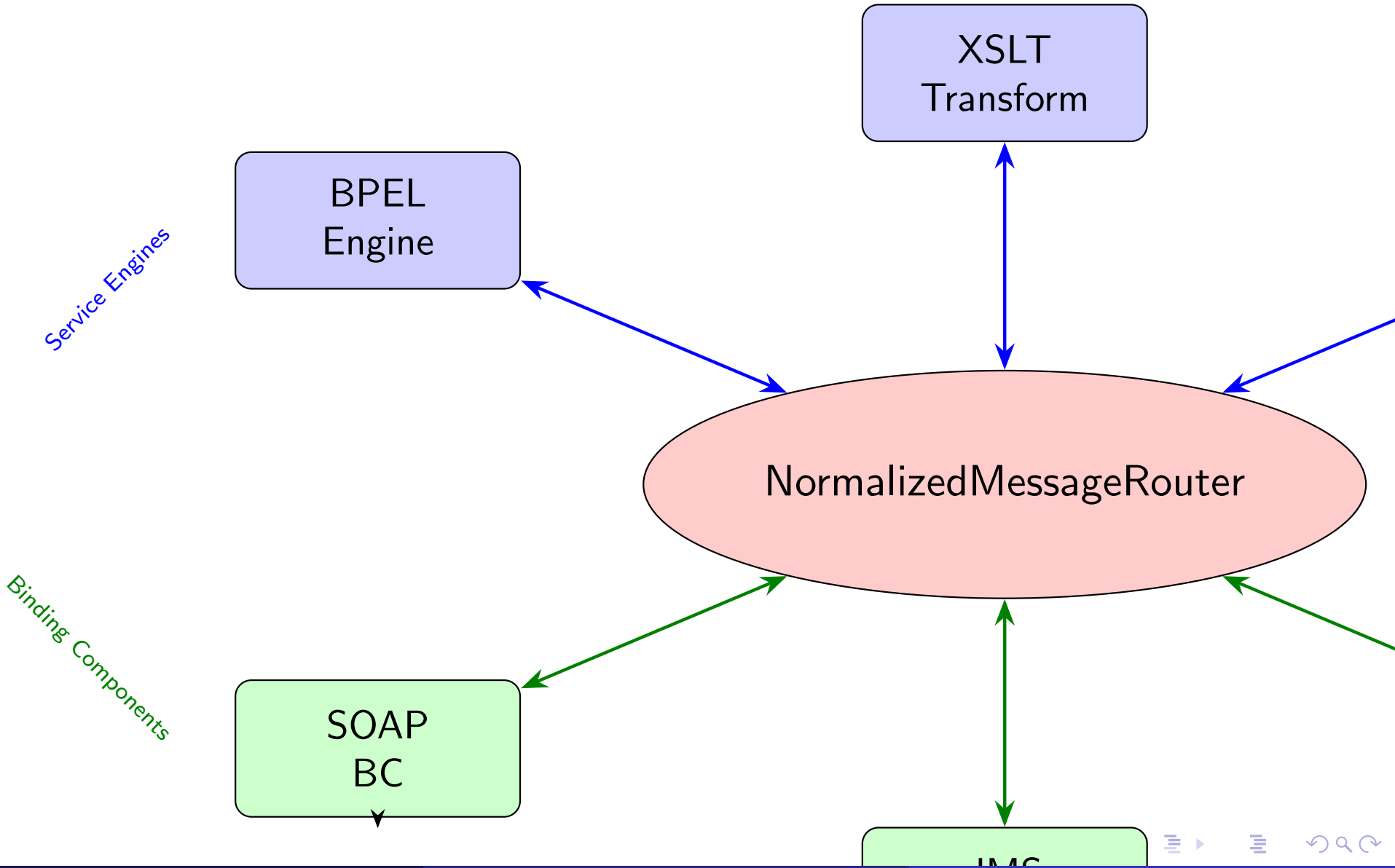
Normalized Message:

- MessageExchange: contenitore dello scambio
- NormalizedMessage: payload + properties + attachments
- Pattern: In-Only, In-Out, etc.

Implementazioni:

- Apache ServiceMix

Schema: JBI components



Pattern di scambio JBI

Message Exchange Patterns (MEP)

Pattern	Descrizione
In-Only	Fire-and-forget. Consumer invia messaggio senza aspettare risposta. Asincrono.
Robust In-Only	Come In-Only ma con acknowledgment o fault.
In-Out	Request-Response. Consumer invia e attende risposta. Sincrono o asincrono.
In Optional-Out	Consumer invia, risposta opzionale. Provider decide se rispondere.

Quando usarli:

- **In-Only**: notifiche, logging, event broadcasting
- **Robust In-Only**: delivery garantito con ack
- **In-Out**: query, transazioni, richieste con risposta obbligatoria
- **In Optional-Out**: pattern flessibili, callback condizionali

Gestione:

- Il pattern è specificato dal consumer

Routing e trasformazione

Due pilastri dell'ESB

Routing

- **Content-Based**: analisi payload
- **Header-Based**: metadata routing
- **Rule-Based**: business rules
- **Dynamic**: lookup runtime registry

Esempio Content-Based:

- Order amount ¡ 1000 $\rightarrow$ Service A
- Order amount $\geq$ 1000 $\rightarrow$ Service B + approval
- Country = "IT" $\rightarrow$ Tax

Trasformazione

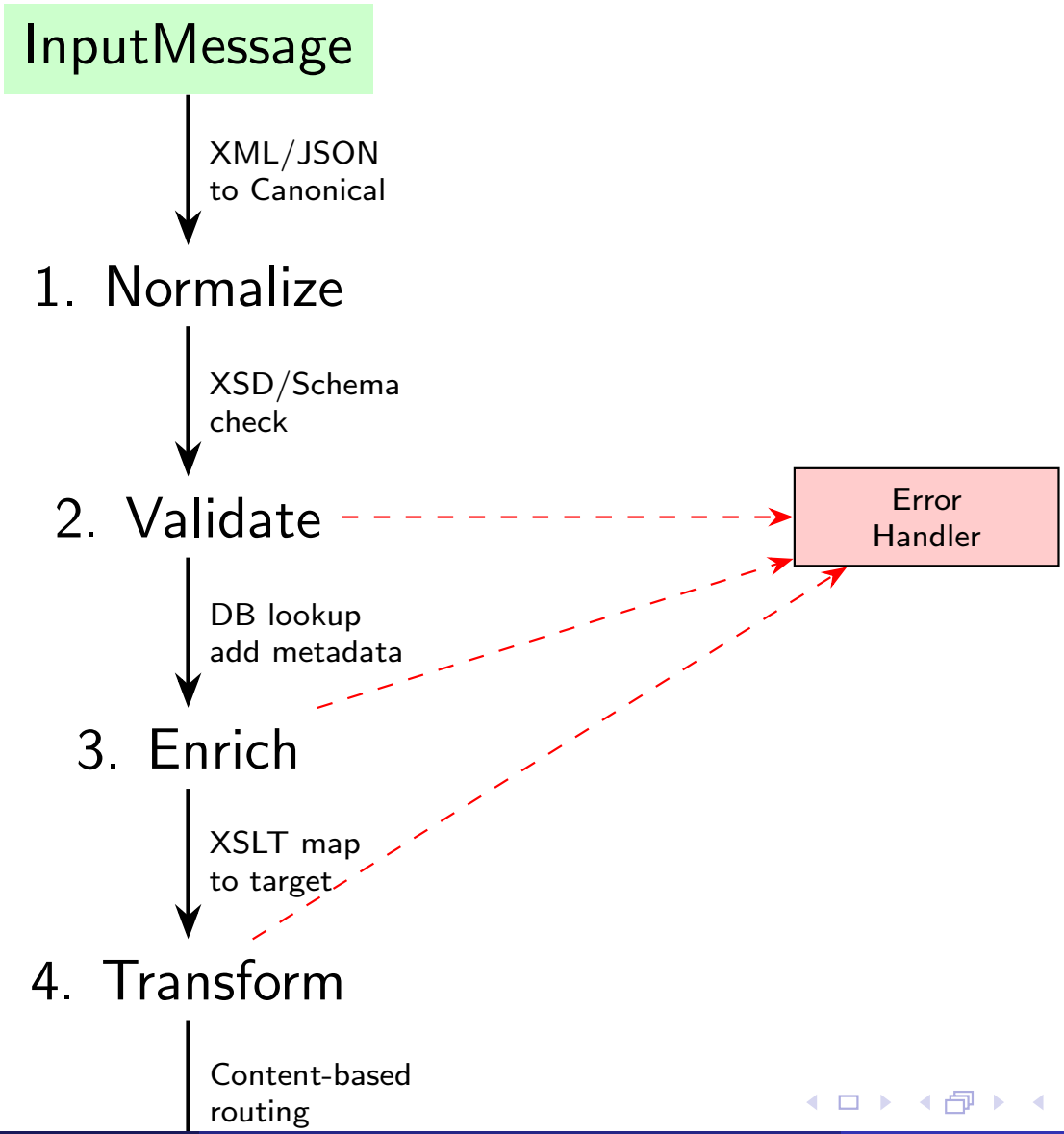
- **XSLT**: XML-to-XML
- **Data Mapper**: visual mapping
- **Custom Code**: Java, Groovy, etc.
- **Template**: Velocity, Freemarker

Operazioni comuni:

- Format conversion (XML$\leftrightarrow$JSON)
- Field mapping e renaming
- Data enrichment (lookup DB)
- Aggregation/splitting

Pipeline di trasformazione: esempio

Flusso end-to-end con trasformazioni multiple



Monitoraggio e gestione operativa

Observability nell'ESB

Componenti chiave:

1. **Logging**
   - Message payload logging (attenzione dati sensibili!)
   - Audit trail completo
   - Correlation ID per tracking end-to-end
2. **Metrics**
   - Throughput (msg/sec)
   - Latency (percentili: p50, p95, p99)
   - Error rate
   - Queue depth
3. **Tracing Distribuito**
   - Span per ogni hop
   - Visualizzazione dependencies
   - Bottleneck identification
4. **Alerting**
   - Threshold-based alerts
   - Anomaly detection
   - Incident management integration

Scalabilità e resilienza

Tecniche per sistemi mission-critical
Scalabilità

- **Clustering**: nodi ESB multipli
- **Load balancing**: distribuire carico
- **Sharding**: partizionamento dati
- **Caching**: ridurre latenza
- **Async processing**: decoupling temporale

Horizontal Scaling:

- Aggiungere nodi ESB
- Shared-nothing architecture
- Message queue distribuite

Resilienza

- **Failover**: switch automatico
- **Circuit breaker**: prevent cascading
- **Retry**: exponential backoff
- **Timeout**: limits attesa
- **Bulkhead**: isolamento failure

High Availability:

- Active-Active deployment
- Geographic redundancy
- Health checks continui

SLA e QoS

Service Level Agreements e Quality of Service

SLA - Service Level Agreement

Contratto tra provider e consumer che specifica metriche di performance, disponibilità e supporto. Spesso con penali per mancato rispetto.

Metriche SLA tipiche:

- **Availability**: 99.9% uptime (43 min downtime/mese)
- **Response Time**: p95 ¡ 200ms, p99 ¡ 500ms
- **Throughput**: min 1000 req/sec
- **Error Rate**: ¡ 0.1%
- **MTTR**: Mean Time To Recovery ¡ 15 min

QoS nell'ESB:

- **Priority queues**: messaggi critici prima
- **Resource reservation**: capacity dedicata
- **Rate limiting**: garantire fair usage
- **Circuit breaking**: protezione cascading failures

Enforcement:

Esempio pratico: integrazione CRM-ERP

Scenario: sincronizzazione ordini

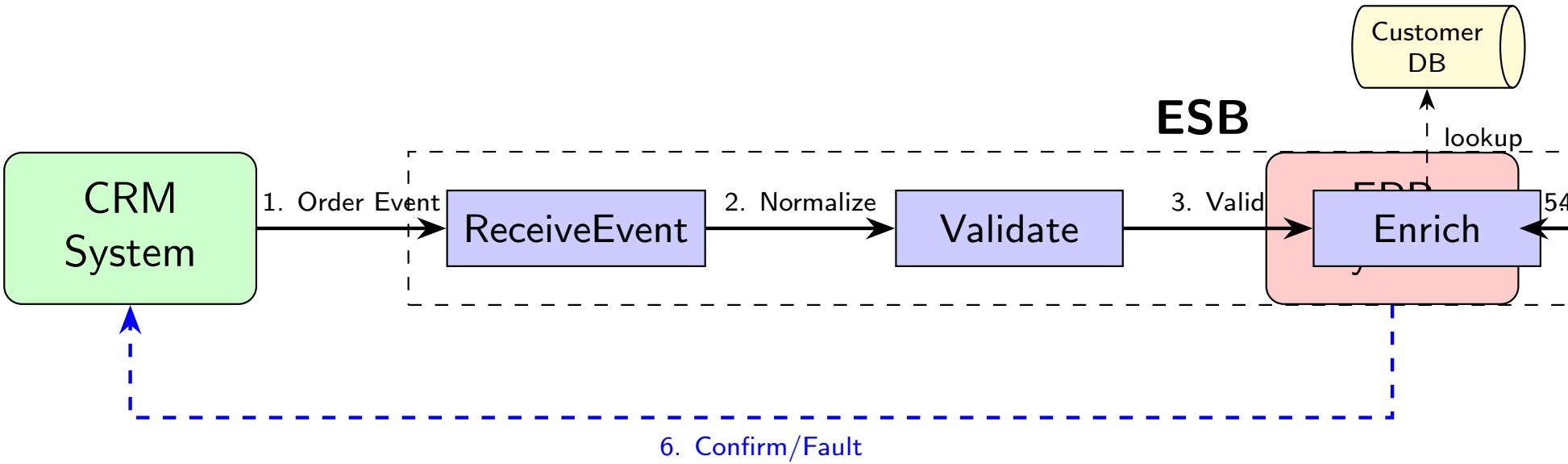
Requisito: Quando un ordine viene creato nel CRM, deve essere automaticamente propagato all'ERP per la gestione magazzino e fatturazione.

Sequenza operazioni:

1. **Trigger**: evento "NewOrder" dal CRM (via webhook o JMS)
2. **ESB Reception**: Binding Component riceve evento
3. **Normalizzazione**: conversione formato CRM $\rightarrow$ formato canonico
4. **Validazione**: verifica completezza dati ordine
5. **Enrichment**: lookup cliente in DB per dati fiscali
6. **Trasformazione**: mapping formato canonico $\rightarrow$ formato ERP
7. **Invocazione ERP**: chiamata SOAP/REST a ERP service
8. **Gestione risposta**:
   - Successo: conferma a CRM
   - Fallimento: retry logic o notifica manuale
9. **Audit**: logging completo transazione per compliance

Pattern applicati: Content-Based Routing, Data Enrichment, Protocol

Diagramma: integrazione CRM-ERP



Rollback e compensazioni

Compensating Transactions Pattern

Problema

In sistemi distribuiti non è possibile usare transazioni ACID classiche.
Servizi esterni potrebbero non supportare 2-phase commit.

Soluzione: Saga Pattern

- Sequenza di transazioni locali
- Ogni transazione ha una compensazione
- In caso di errore, esegui compensazioni in ordine inverso

Esempio: prenotazione viaggio

1. Prenota volo $\rightarrow$ Compensazione: cancella volo
2. Prenota hotel $\rightarrow$ Compensazione: cancella hotel
3. Prenota auto $\rightarrow$ Compensazione: cancella auto
4. Errore al passo 3 $\rightarrow$ esegui: cancella hotel, cancella volo

Considerazioni:

- Le compensazioni potrebbero fallire (gestire idempotenza)
- Eventual consistency invece di strong consistency

Testing di Web Services

Strategie di test multi-livello

1. **Unit Testing**
   - Test singoli componenti in isolamento
   - Mock di dipendenze esterne
   - Framework: JUnit, TestNG, pytest
2. **Contract Testing**
   - Validazione WSDL/schema
   - Verifica input/output contract
   - Tools: Pact, Spring Cloud Contract
3. **Integration Testing**
   - Test flussi end-to-end
   - Ambienti staging con servizi reali o stubs
   - Tools: SoapUI, Postman, REST Assured
4. **Performance Testing**
   - Load testing: throughput sotto carico
   - Stress testing: comportamento al limite
   - Tools: JMeter, Gatling, k6
5. **Security Testing**
   - Penetration testing

DevOps e CI/CD per servizi

Pipeline automatizzata per Web Services

Fasi della pipeline:

1. **Source Control**
   - Git repository per WSDL, XSD, code
   - Branching strategy (GitFlow, trunk-based)
2. **Build**
   - Compilazione codice
   - Generazione artefatti (WAR, JAR)
   - Validazione WSDL/XSD
3. **Test Automatici**
   - Unit test, integration test
   - Contract verification
   - Code coverage report
4. **Deploy**
   - Deploy su ambiente di staging
   - Smoke tests
   - Blue-green o canary deployment
5. **Monitoring**
   - Health checks post-deploy

Migration strategy

Approccio incrementale legacy → ESB/SOA

Fase 1: Assessment

- Inventario sistemi esistenti
- Identificazione integrazioni point-to-point
- Analisi volumi e criticità

Fase 2: Pilot

- Selezione caso d'uso non critico
- Implementazione ESB per integrazione pilota
- Validazione approccio e ROI

Fase 3: Expansion

- Migrazione progressiva integrazioni
- Priorità su high-value / high-pain
- Approccio strangler fig pattern

Fase 4: Consolidation

- Dismissione integrazioni legacy
- Standardizzazione su ESB
- Governance e best practices

Esempi di prodotti ESB

Vendor e soluzioni open-source

Prodotto	Tipo	Caratteristiche
Mule ESB	Commercial	Potente, CloudHub, vasto ecosistema
WSO2 ESB	Open Source	API-centric, microservices-ready
IBM Integration Bus	Commercial	Enterprise-grade, mainframe integration
Apache ServiceMix	Open Source	JBI-compliant, Karaf-based
Apache Camel	Open Source	Routing engine, 300+ connectors
Red Hat Fuse	Commercial	Basato su Camel, Kubernetes-native
Oracle SOA Suite	Commercial	Full stack SOA, BPEL engine
Talend ESB	Open Source	Data integration focus

Trend attuali:

- Shift verso API management e microservices
- Cloud-native ESB (service mesh, Istio, Linkerd)
- iPaaS (Integration Platform as a Service)
- Event-driven architecture (Kafka, event streaming)

Confronto: Hub-and-Spoke vs Bus

Sintesi dei trade-off

Caratteristica	Hub-and-Spoke	Bus (ESB)
Scalabilità	Limitata	Elevata
SPOF Risk	Alto	Basso
Gestione	Centralizzata (più semplice)	Distribuita (più complessa)
Performance	Bottleneck centrale	Distribuita
Costo iniziale	Basso	Medio-Alto
Complessità	Bassa	Media-Alta
Manutenzione	Più semplice	Più articolata
Resilienza	Bassa	Alta
Geo-distribution	Difficile	Nativa

Raccomandazione:

- **Hub**: piccole-medie aziende, integrazioni limitate
- **ESB**: enterprise, high-volume, mission-critical
- **Ibrido**: hub + bus per bilanciare complessità e scalabilità

Sicurezza nell'ESB

Gestione centralizzata delle policy di sicurezza

Vantaggi centralizzazione:

- Policy security uniformi applicate a tutti i servizi
- Single point of enforcement
- Audit centralizzato
- Gestione certificati e credenziali centralizzata

Meccanismi:

1. **Authentication**
   - OAuth 2.0 / OpenID Connect
   - SAML assertions
   - Certificati X.509
   - API keys
2. **Authorization**
   - RBAC (Role-Based Access Control)
   - ABAC (Attribute-Based Access Control)
   - Policy enforcement points
3. **Encryption**
   - TLS/SSL per transport

Best practices architetturali

Principi guida per progettazione robusta

1. **Design for Failure**
   - Assume ogni componente può fallire
   - Circuit breakers, timeouts, retries
   - Graceful degradation
2. **Idempotenza**
   - Operazioni ripetibili senza effetti collaterali
   - Essenziale per retry logic
   - Usa unique message IDs
3. **Versioning**
   - Versioning esplicito API e servizi
   - Backward compatibility quando possibile
   - Deprecation policy chiara
4. **Contratti stabili**
   - WSDL/OpenAPI come contratto
   - Consumer-driven contracts
   - Breaking changes con major version bump
5. **Loose Coupling**
   - Servizi indipendenti

Versioning dei servizi

Strategie per evoluzione senza breaking changes

1. **URL Versioning**
   - `/api/v1/orders`
   - `/api/v2/orders`
   - Pro: esplicito, semplice
   - Contro: proliferazione endpoint
2. **Header Versioning**
   - `Accept: application/vnd.api+json; version=2`
   - Pro: URL stabili
   - Contro: meno visibile
3. **Query Parameter**
   - `/api/orders?version=2`
4. **Namespace WSDL**
   - `xmlns:v1="http://...v1"`
   - `xmlns:v2="http://...v2"`
   - Pro: standard SOAP
   - Contro: verboso

Compatibility Guidelines:

- **Additive changes**: OK senza breaking
  - Nuovi parametri opzionali
  - Nuovi endpoint
  - Nuovi campi response
- **Breaking changes**: richiedono nuova versione
  - Rimozione campi

Governance in SOA

Gestione del ciclo di vita dei servizi

SOA Governance

Insieme di policy, processi e strumenti per assicurare che i servizi siano progettati, implementati, usati e mantenuti in modo coerente con gli obiettivi aziendali.

Componenti:

1. **Service Registry**
   - Catalogo centralizzato servizi
   - Metadati: owner, SLA, dependencies
   - Stato: dev, test, prod, deprecated
2. **Policy Management**
   - Naming conventions
   - Security standards
   - Performance requirements
3. **Lifecycle Management**
   - Design → Development → Test → Deploy → Operate → Retire
   - Approval gates tra fasi
   - Change management

Metriche e KPI

Key Performance Indicators per ESB/SOA

Metrica	Target tipico	Uso
Latency (p50)	¡ 100ms	Performance media
Latency (p95)	¡ 200ms	Outliers gestibili
Latency (p99)	¡ 500ms	Worst-case acceptable
Throughput	¿ 1000 req/s	Capacità sistema
Error Rate	¡ 0.1%	Affidabilità
Availability	¿ 99.9%	Uptime
MTBF	¿ 720h (30 giorni)	Mean Time Between Failures
MTTR	¡ 15 min	Mean Time To Recovery
Queue Depth	¡ 1000 msg	Backlog monitoring
CPU Usage	¡ 70%	Resource utilization

Dashboard e Alerting:

- Real-time dashboard per ops team
- Alert quando threshold violati
- Trend analysis per capacity planning
- SLA compliance reporting

Caso di studio sintetico

Azienda manifatturiera: integrazione supply chain

Scenario: Azienda con ERP, sistemi fornitori esterni, portale B2B, magazzino automatizzato. Necessità di orchestrare ordini, spedizioni, fatturazione.

Architettura proposta:

- **ESB centrale**: Mule ESB on-premise
- **Connettori**:
  - SAP ERP (IDoc, RFC)
  - Fornitori (REST API, EDI, email)
  - Warehouse (SOAP WS)
  - Portal B2B (REST/JSON)
- **Orchestrazione**: BPEL per processo ordine completo
- **Monitoraggio**: ELK stack + Grafana
- **Sicurezza**: mTLS, OAuth tokens, API gateway

Benefici ottenuti:

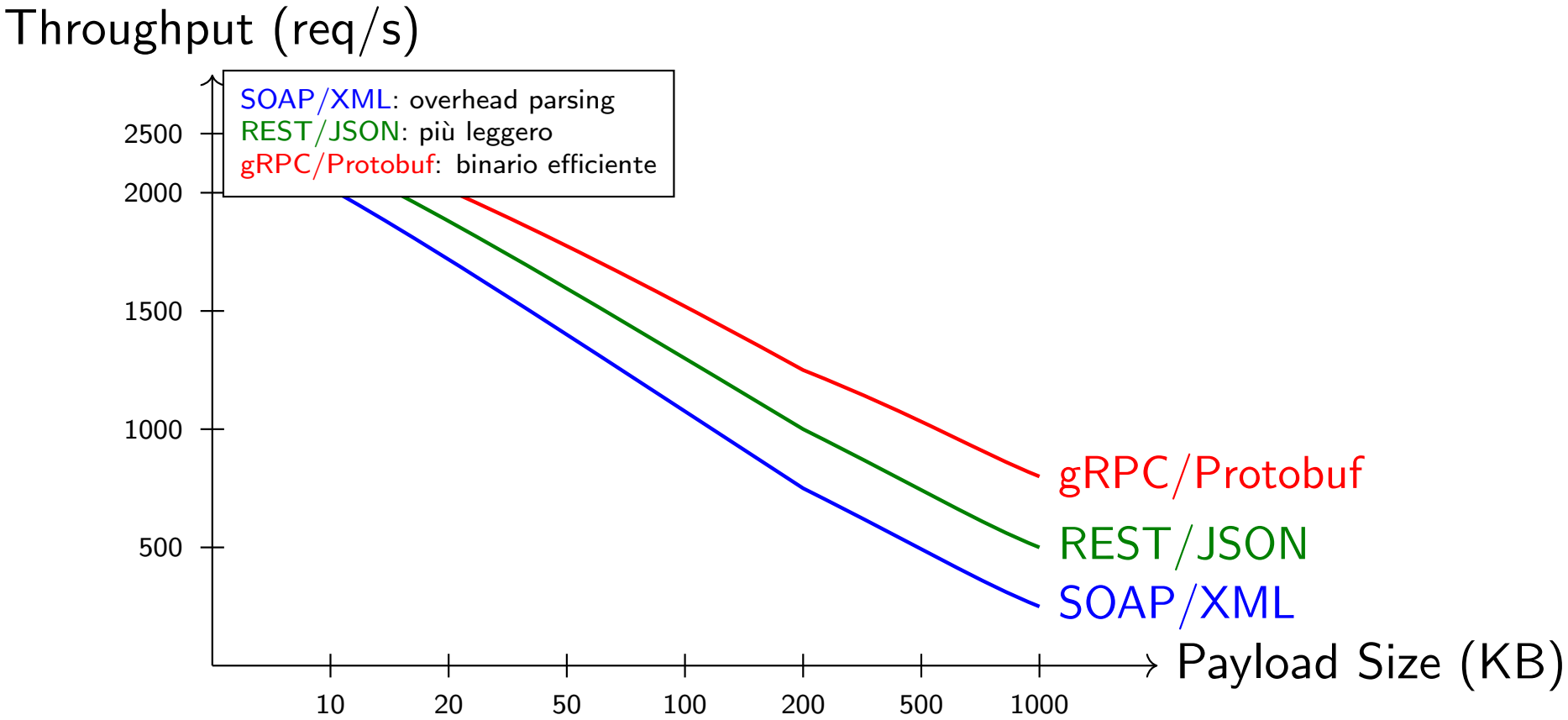
- Riduzione 70% tempi integrazione nuovi fornitori
- Eliminazione errori manuali (data entry)
- Visibilità real-time su supply chain

Esempio: flow end-to-end

Tracciamento completo transazione

1. **Evento iniziale**: Cliente crea ordine su portale web
   - Timestamp: 2024-12-08T10:15:30Z
   - Correlation ID: ORD-2024-123456
2. **Ingresso ESB**: HTTP POST ricevuto
   - Validazione JWT token
   - Logging request
3. **Orchestrazione BPEL**:
   - Verifica disponibilità magazzino (SOAP call)
   - Calcolo prezzo con sconti (Service Engine)
   - Creazione ordine in ERP (SAP RFC)
   - Notifica fornitori (JMS message)
4. **Sistema Terzo**: Fornitore conferma disponibilità
   - Callback asincrono via REST
5. **Risposta**: Conferma ordine a cliente
   - HTTP 200 + Order ID
   - Email notifica
6. **Audit**: Log completo in DB audit
   - Compliance GDPR

Grafico: confronto throughput vs payload



Osservazioni:

- Payload grandi penalizzano SOAP per overhead XML
- gRPC/Protobuf eccelle con dati strutturati complessi
- Per payload < 10KB le differenze sono minime

Raccomandazioni progettuali

Linee guida pratiche

1. **Start Simple**
   - Non over-engineer inizialmente
   - Aggiungere complessità quando necessario
   - Proof of concept prima di committare
2. **Document Everything**
   - WSDL/OpenAPI sempre aggiornati
   - Architettura diagrams
   - Runbooks operativi
3. **Automate**
   - CI/CD pipeline completa
   - Infrastructure as Code
   - Automated testing
4. **Monitor Proactively**
   - Non aspettare problemi per monitorare
   - Baseline metrics in produzione
   - Capacity planning basato su trend
5. **Security by Design**
   - Non aggiungere sicurezza dopo

Prospettive future

Tendenze emergenti

1. **Shift verso architetture moderne:**
   - **Microservizi**: granularità fine, deploy indipendenti
   - **Service Mesh**: Istio, Linkerd per comunicazione service-to-service
   - **Serverless**: FaaS per integrazione event-driven
2. **API-first design:**
   - REST/JSON dominante per semplicità
   - GraphQL per query flessibili
   - gRPC per performance critical
   - AsyncAPI per event-driven
3. **Cloud-native integration:**
   - iPaaS (Integration Platform as a Service)
   - Managed ESB in cloud (Azure Integration Services, AWS AppFlow)
   - Kubernetes-native integration (Knative, Camel-K)
4. **Event streaming:**
   - Apache Kafka per real-time data pipelines
   - Event sourcing e CQRS patterns

Risorse e letture consigliate

Standard e Specifiche:

- W3C SOAP Specifications: `https://www.w3.org/TR/soap/`
- W3C WSDL 2.0: `https://www.w3.org/TR/wsdl20/`
- OASIS WS-* Specifications: `https://www.oasis-open.org/`
- JSR 208 JBI: `https://jcp.org/en/jsr/detail?id=208`

Libri di riferimento:

- "Enterprise Integration Patterns" - Hohpe & Woolf
- "SOA Principles of Service Design" - Thomas Erl
- "Web Services Essentials" - Cerami
- "Building Microservices" - Sam Newman

Progetti Open Source:

- Apache Camel: `https://camel.apache.org/`
- Apache ServiceMix: `https://servicemix.apache.org/`
- WSO2: `https://wso2.com/`
- Mule (Community): `https://www.mulesoft.com/`

Tutorial e Corsi:

- Udemy, Coursera, Pluralsight: corsi su SOA, ESB, Web Services

Conclusioni

Punti chiave della presentazione

1. **Web Services** hanno rivoluzionato l'integrazione enterprise con standard aperti e firewall-friendly
2. **SOAP/WSDL/UDDI** formano la base tecnica ma con complessità significativa
3. **ESB** rappresenta l'evoluzione da integrazioni point-to-point a architetture scalabili e governate
4. **Orchestrazione e coreografia** permettono composizione di processi complessi
5. **Governance, monitoraggio, sicurezza** sono pilastri per deployment enterprise
6. **Futuro** è verso microservizi, service mesh, API management e event streaming

Suggerimenti per approfondire

- Sperimentare con progetti open-source (Camel, ServiceMix)