Enterprise Service Bus (ESB)

Contents:

- Review Integration Approaches
- Influencing Technologies
- Infrastructure Reqs
- ESB Container
- Integration Services
- ESB MOM

Mariano Cilia / cilia@informatik.tu-darmstadt.de

[From Chappell'04]

1

Recent Past

- Recent past years significant technology trends, such as
  - Service-oriented Architecture (SOA)
  - Enterprise Application Integration (EAI)
  - Business-to-Business (B2B)
  - Web Services (WS)
- They attempted to address the challenges of integrated business processes

2

Enterprise Service Bus

- The ESB concept is a new approach to integration
  - loosely coupled, highly distributed integration network
- Standards-based integration platform that combines:
  - messaging
  - web services
  - data transformation
  - intelligent routing
- Reliably connect and coordinate the interaction of diverse applications across (extended) enterprises with TX integrity

3

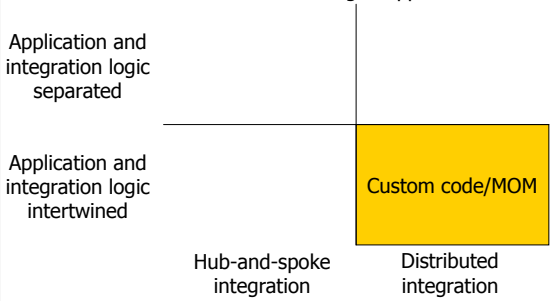
Motivation

- Some large organizations have more than 1.000 distinct app types with more than 10.000 instances running
  - It is unrealistic to assume that all apps can use the same technology to participate in an integration environment

4

Integration Approaches

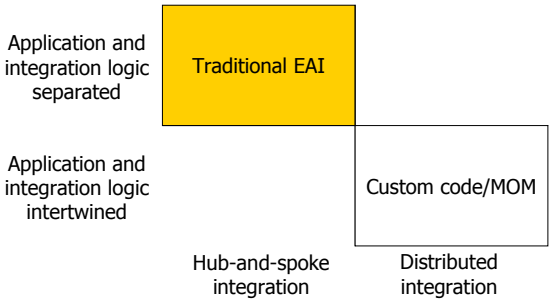
+ connect apps in a loosely coupled async fashion
- low-level coding in applications



5

Integration Approaches

+ Centralized functions, e.g. management
+ separation of app and integration logic
- Does not scale well



6

Integration Approaches

+ can interoperate through standard protocols
- Tightly coupled, does not scale well

	Hub-and-spoke integration	Distributed integration
Application and integration logic separated	Traditional EAI	
Application and integration logic intertwined	Application Server	Custom code/MOM

7

Integration Approaches

+ configured instead of coded
+ highly distributed
+ clear separation of buz and integr. logic

	Hub-and-spoke integration	Distributed integration
Application and integration logic separated	Traditional EAI	ESB
Application and integration logic intertwined	Application Server	Custom code/MOM

8

ESB - Origins

- ESB is not merely an academic exercise
- Born out of necessity!
  - Based on real requirements – difficult integration problems
- It's not just EAI + WS nor MOM + WS
  - A number of trends influenced on the evolution of the ESB
    - Both business- and technology-driven

9

Influencing Technologies

- Enterprise Application Integration (EAI)
  - Good practices in data transformation and manipulation
- e-Marketplaces and vertical trading hubs
  - Recognition of a need for open standards for protocols and service discovery mechanisms
  - First to introduce the model of loosely-coupled, distributed federation of individual companies operating autonomously but collaborating together
- Java Message Service (JMS)
  - Standard API and behavioral semantics of MOM
- Application Servers
  - Nurtured the evolution of some important standards like servlets, JDBC, JCA, …

10

Influencing Technologies (2)

- Y2K and post Internet-bubble economics
  - Caused major corporate reevaluation of big IT spending
  - Shift toward trying to make things work with the existing applications (and with a much smaller budget)
- Web services and SOA
  - Industry effort to provide platform-independent SOA
  - Relying on interoperable interface descriptions, protocols and data communications
- Evolution and maturation of standards
  - WS are important standards for XML, security and reliable messaging
- The "accidental" architecture
  - Nobody wants to create it but everybody has it

11

Evolution

- The invention of the ESB was not an accident
  - The result of vendors working with forward-thinking customers
    - Trying to build a standard-based integration network using a foundation of SOA, messaging and XML
- Positive influences from its predecessor approaches and avoids the downsides

12

Company Needs

- Various IT depts across business units needed to protect their investments in existing messaging and integration broker installations
- Be able to reach other corporate applications as shared services in a non-intrusive way
- Common API and event-driven service interfaces are a core part of the design of an ESB
  - Diversity in connectivity options is critical to the adoption of an integration strategy
  - Bring the infrastructure to the apps
    - Facilitating an incremental approach to adoption
  - core part of the definitions of an ESB, multiple…
    - client types
    - connectivity protocols
    - application adapters
    - MOM bridges

13

Towards the Extended Enterprise

- EAI + evolution of Internet + App Servers
  - Clear separation between communications and app integration infrastructure
    - Driven by the capabilities and limitations of the technology
      - J2EE App Server: clear distinction between what's inside the firewall (EJB container) and what's outside (web container)
- Hub-and-spoke EAI brokers could get as far as the corporate boundaries
  - Not built for scaling beyond that
- Various Bridging Technologies designed for the "edge" of the network
  - Web services focused on this

14

e-Commerce Trading Hubs

- Facilitates electronic commerce between buyers and suppliers in a supply chain
- A trading exchange acts as an intermediary, or semi-private business portal
  - "special" apps instead of browser-based
- Interaction between apps residing in a trading partner and the trading hub
- Trading hub provides value-added functions, such as dissemination of
  - Requests for Quote (RFQ) between buyer and multiple suppliers
  - Availability to Promise (ATP) data from suppliers to buyers

15

e-Commerce Trading Hubs – Why?

- Move away from expensive EDI Value Added Networks (VAN)
- Use public Internet as means of communications wherever possible
- Lower the barrier of entry
  - According to the budget of small and medium enterprises
- Potential community could consist of thousands of trading partners

16

Trading Hubs - Requirements

- Secure communications
  - Strict authentication and access control between entities connecting to the hub
- Asynchronous communications
  - "fire-and-forget"
- Reliable messaging
- Handle of large volumes of message traffic
- Need that segregated groups of backend apps expose and share their interfaces and data with other apps residing in other trading hubs
  - Maintaining their autonomy!
- BUT traditional MOM vendors did not have an infrastructure capable of spanning across Internet in a scalable and secure fashion

17

Context (Comment #1)

- Different levels of trust when dealing with business partners
  - Reliable messaging requires a piece of MOM installed on either side of the communication link
    - BUT business partners are not always in a position to install software on the partner-side
    - If the relationship is not that close (or not that stable) you need to supply clear instructions on how to send business documents over secure HTTP

18

Standards! (Comment #2)

- What do we mean when talking about standards?
  - In this context it refers to an specification or implementation that has gained enough traction in the industry with
    - Staying power
    - Open enough for multiple vendors

19

Standard-based Integration

- Why adopting standards?
  - (in-house) IT professionals with expertise
    - e.g. XML-related technologies (SOAP, WSDL, XSLT, XPath, XQuery,…)
    - Allows developers to apply this knowledge across a variety of (integration) projects
  - Increases the ability to integrate with business partners using standard interfaces and protocols
    - E.g. Java standards (JMS, JCA)
  - Reduces proprietary vendor lock-in

20

Towards the Integration Process

- Consider now …
  - Business Process definition
    - Represents a series of steps in a flow between apps
    - Steps can be mapped into service endpoints representing physical apps
    - Now it is possible to administratively insert additional steps into that path of control flow
      - Without modifying the apps directly
  - Content-based routing
  - Data transformation

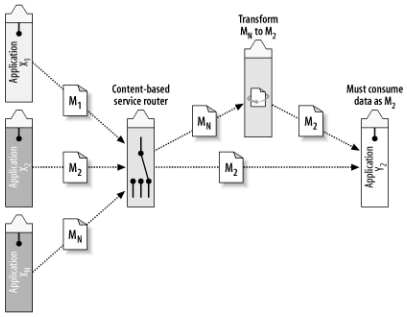
21

Towards the Integration Process (2)

- Complex integration tasks can be carried out through a sequence of inspection, routing and transformation combined with XML
- Greater flexibility in the coordination of development timelines of the apps that are driving the requirements

22

Towards the Integration Process (3)



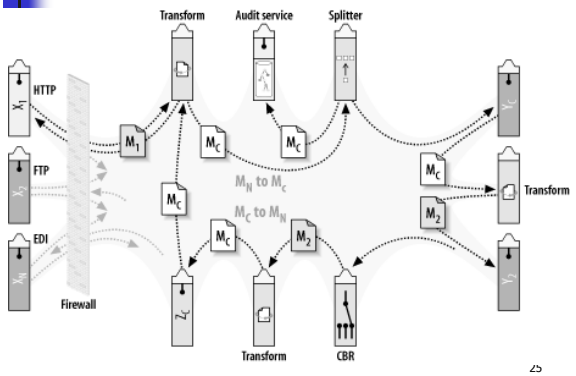
23

Towards a Generic Data Exchange

- When using specific point-to-point transformations between apps, the number of transformation instances increases exponentially with the number of apps
- Data translation to and from a Canonical format
  - A canonical, app-independent version of the data is key
  - Best-practice: decide on a set of canonical XML formats
    - Companies standardize the definition of common business entities
- The level of "format" is at the business-object level
  - e.g. the definition of a purchase order, or shipping address
  - Vertical dialects
    - Commerce relies mostly on xCBL
    - financial services rely on SWIFT
    - Healthcare on HIPPA or HL7

24

Towards a Generic Data Exchange (2)



25

Canonical - Benefits

- Each app focus on only one type of transformation to and from a common format
- New apps can adopt the canonical format directly
  - Not anticipated apps can tap into the flow of messages
- Services can be written to use the canonical format
- Standard stylesheet templates and libraries can be created and reused throughout the organization

26

ESB: Service Containers and Abstract Endpoints

Enterprise Service Bus (ESB)

- Provides a loosely coupled, highly distributed approach to integration
- An ESB consists (basically) of …
  - Message Oriented Middleware
    - Robust, reliable transport
    - Efficient movement of data across (abstract) channels
    - End-to-end reliability
  - Web Services
    - Service-oriented Architecture (SOA)
    - Abstract business services
  - Intelligent Routing
    - Based on content and context
    - Based on business process rules
  - XML data transformations
    - Based on XSLT and independent deployed services

28

ESB Infrastructure Requirements

- Highly distributed, scalable service containers
- Event-driven service invocation
- Centralized management of distributed integration configurations
- Diverse client connectivity and support for multiple protocols
- Unified security and access control model
- Distributed configuration and caching of deployment resources
  - Such as transformation scripts, routing rules
- Scriptable and declarative environment
- Changeable behavior of integration components based on configuration and rules
  - Instead of compiling behavior into code

29

An ESB …

- Can integrate apps built with .NET, C#, …
- Can utilize J2EE components (JMS, JCA,…) and the J2EE WS APIs
- XML standards
  - XSLT for data transformation
  - XPath for data access and intelligent routing
  - XQuery for querying "in-flight" data
- All these provide an open-ended, pluggable, service-oriented architecture
  - Supporting both industry-standard integration components as well as proprietary elements (through standardized interfaces)
- Three key elements, the use of:
  - Abstract endpoints (for representing remote services)
  - Internet-capable MOM
  - Distributed lightweight service container

30

Abstract Endpoints

- For the integration architect all applications and services are treated as abstract endpoint
- Logical abstractions of services that are plugged into the bus
- Endpoints represent diverse things
  - a single, monolithic app (like payroll system)
  - a suite of applications
  - an ERP
  - an integration broker
  - a business unit
  - an interface to a business partner

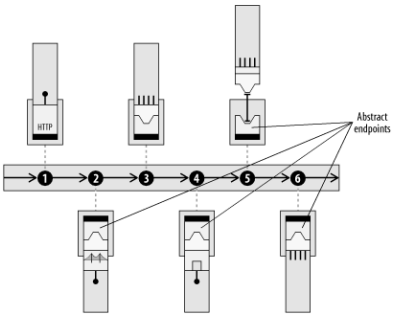
31

Abstract Endpoints

- All service endpoints are equal participants in an event-driven SOA
- Building out an integration network involves:
  - tying together service endpoints,
  - applying choreography,
  - transformation and
  - routing rules … into process flows between apps
- This allows the use of high-level tools to assemble service endpoints into process flows

32

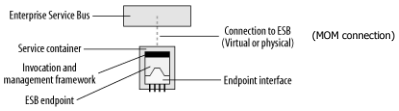
Endpoints – Abstract View



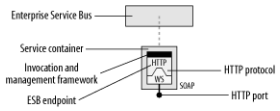
33

(Notations)

- Generic ESB endpoint



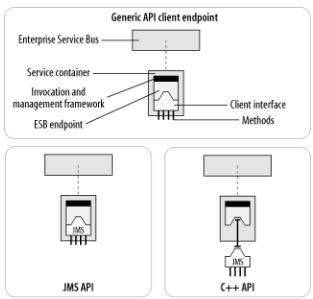
- SOAP or web service endpoint



34

(Notations)

- ESB client API endpoint



Exposes methods that an app or service would use to plug into the bus

Pure java, relying on JMS

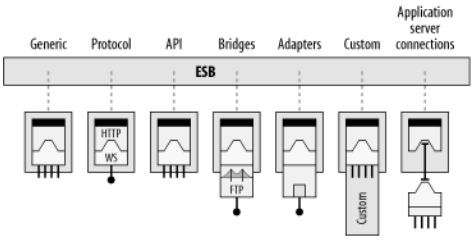
separation between the physical implementation of the C++ client and the endpoint.

Java and C++ via JNI or network layer

35

(Notations)

- Various client types and protocols



36

Connectivity at the Core

- Highly scalable enterprise messaging backbone is key!
  - Capable of asynchronously transporting data as messages in a secure an reliable fashion
- The messaging core could be:
  - A proprietary MOM
  - a MOM based on JMS
  - a MOM based on WS-ReliableMessaging
  - a generic messaging engine

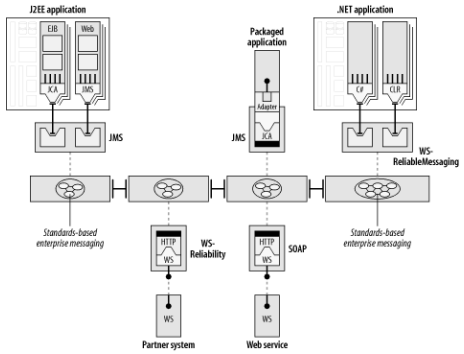
37

Connectivity at the Core (2)

- The endpoint provides an abstraction away from the underlying protocol
  - Allowing the underlying protocol to vary depending on the deployment situation and the QoS requirements
- It is responsibility of the ESB to map the high-level process flows into individual service invocations across the designated transport

38

Connectivity at the Core (3)



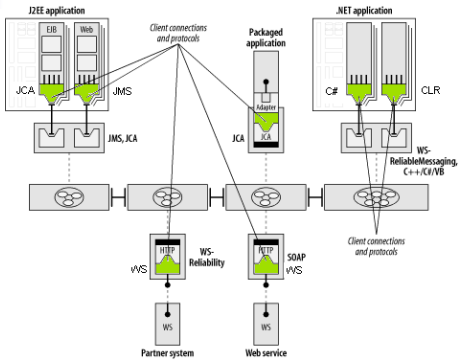
39

A diversity of connection choices

- Context
  - IT departments cannot adapt significantly legacy apps
  - At least it may be capable of dumping and loading flat files and transmitting them over FTP
  - The app in question may have never been intended to interface with anything
  - The app my have exposed a low-level, network socket-level interface as part of a point-to-point integration project
  - The app may reside at a business partner's site, being able to interoperate by using SOAP/HTTP
  - The app may rely on EDI
- Applications that want to connect to the ESB do not require drastic changes
  - It is the bus (not the apps) that provides the flexibility in connection technologies

40

A diversity of connection choices



41

Integration Services

- These services allow apps easy interoperability, for instance
  - Transformation service
    - Applies a transformation from one format/context/data structure to another
      - an XSLT stylesheet to convert in-flight XML messages from one XML dialect to another
  - Routing service
    - Evaluates expressions against in-flight messages to determine where to send it next
      - Applies an XPath expression to look into the XML document to decide destination
  - Logging
    - Copy the message for auditing and tracking purposes

42

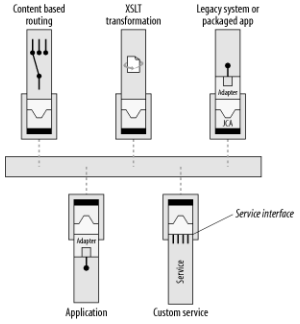
Integration Services (2)

- They can be independently deployed anywhere within the network
  - Precise deployment of integration capabilities at specific (demanding) locations.
  - Can scale independently
- As a result services can easily be
  - upgraded,
  - moved,
  - replaced or
  - replicated …

  without affecting cooperating apps

43

Integration Services (3)



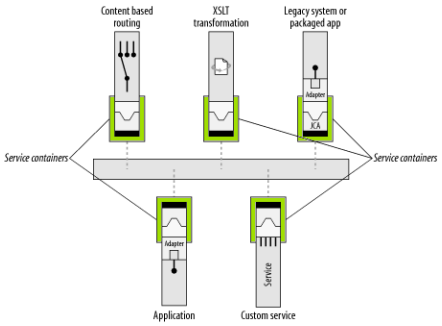
44

ESB Service Container

- Physical manifestation of the abstract endpoint
- Provides the implementation of the service interface
- Is a remote process that can host software components
  - With the goal of hosting integration services
- Simple and lightweight

45

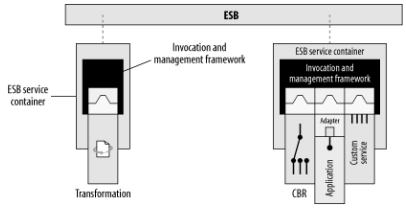
ESB Service Container (2)



46

ESB Service Container (3)

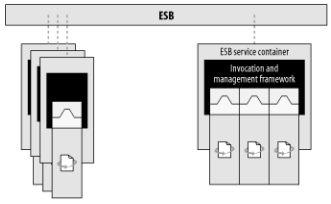
- Can host a single service or can combine multiple services



47

ESB Service Container (4)

- Services may be scaled within a container, and several containers may be scaled across multiple machines

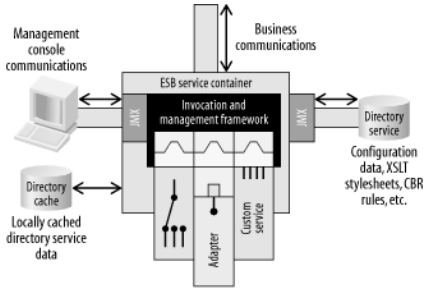


48

Anatomy of a Service Container

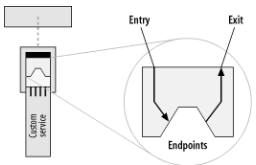
- Supports the retrieval of configuration data from a directory service
  - including a local cache of configuration data
    - If directory service becomes temporarily unavailable it can continue to operate
- Management input data includes
  - Start
  - Stop,
  - Shutdown and
  - Refresh cache
- Management output data consists of event tracking
  - e.g. successful execution of a service or a failure occurred
- Management console (or some other mgmt tool)
  - To manage input and output data
  - Relying on standard protocols as SNMP

Anatomy of a Service Container



The ESB Service Interface

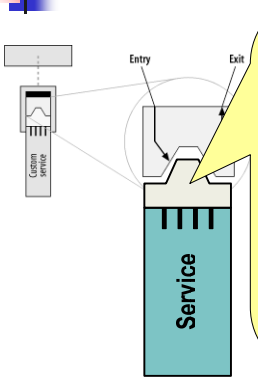
- A container manages an entry endpoint and an exit endpoint
  - Used by the container to dispatch a message to and from the service



The ESB Service Interface (2)

- (XML) messages are received by the service from a configurable entry point
- Upon completion of its task, the service implementation simply places its output message in the exit endpoint
  - The exit message may be based on the input or on a completely new one
  - In sophisticated cases, one input message can transform into many outputs

The Service Interface (3)




```
public void service(ESBServiceContext ctx)
throws ESBServiceException {

   // Get any runtime process parameters.

   Parameters params = ctx.getParameters( );

   // Get the message.
   ESBEnvelope env = null;
   env = ctx.getNextIncoming( ); // entry
   if (env != null) {
      ESBMessage msg = env.getMessage( );
      ...
     //  Perform the service
      //  Operate on the message
}
// Put the message to the Exit Endpoint
ctx.addOutgoing(env); // exit
}
```

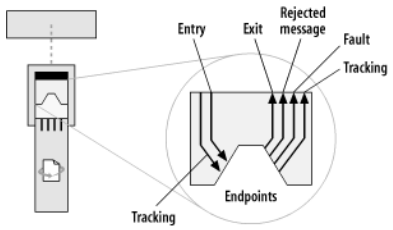

Auditing, Logging and Error Handling

- Important business function within an integration strategy
- Allow to track the exchanged data at a business level
- An ESB container provides an auditing and logging facility
- Multiple sources for tracking data
  - System-level
    - Info about the service itself
    - The flow of messages can be tracked and monitored
  - Application-level auditing, logging and fault handling
    - Accomplished through additional endpoints

Auditing, Logging and Error Handling

- Additional endpoints:
  - Tracking endpoint
    - A service can place a message in the tracking endpoint in addition to the normal exit endpoint
  - Rejected-message endpoint
    - Can be used for system-level errors, e.g. malformed XML documents or a service throwing exceptions
  - Fault endpoint
    - Any app-level errors or faults
- The service implementation has these endpoints available
  - It simply places data/messages in the corresponding endpoint
- Endpoints can be configured separating the implementation of the service from the details about e.g. a fault handling

Auditing, Logging and Error Handling



Auditing, Logging and Error Handling

- Tracking can be handled at both
  - the individual service level
    - Faults and auditing can be tied to the context of the greater business process
  - the business process level
    - May make use of different implementations of individual services over time
- Auditing and logging can be also done at the ESB level
  - e.g. placing a persistence service in the path of messages in a process flow
    - Less intrusive to each individual service implementation

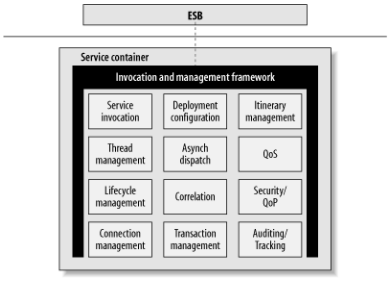
ESB Service Container Facilities

- The ESB provides a variety of facilities that a service implementation may use
  - Location
  - Routing
  - Service invocation
  - Management
  - Lifecycle management
    - Startup, shutdown, resource cleanup
  - Transaction service
  - Thread pooling
  - Connection management
    - Bindings to the underlying MOM
    - Fault tolerance and retry logic
  - Quality of Protection (QoP) and security
  - Quality of Service (QoS)

ESB Service Container Facilities

- Some of these are (best) delegated to the underlying MOM
- The container facilities represent an agreement between the container and the underlying middleware
  - Each facility has its own configuration that can be dealt with relying on an administrative tool (part of the ESB)
    - e.g. QoS for exactly-once delivery would be set as an administrative property of a service (not coded into the service)

ESB Service Container Facilities



Administration of ESB Facilities

- Full-blown GUI
- Command-line interface
  - For those accustomed to automating processes by building scripts
- API
  - There may be a legacy administrative tool in place

61

ESB Service Invocation

- The traditional find/bind/invoke model requires writing the routing and flow logic into the apps that need to be connected
- In ESB basic design a service is not invoked directly by another service, but is part of a larger event-driven process flow
  - Find/bind/invoke occurs as part of the ESB mechanism, but it is separated out from the business logic
    - The implementation of a service:
      - Simply deals with an entry/exit endpoint metaphor
      - Focus only on the implementation logic of a service

62

ESB Service Invocation

- A service receives a (XML) message (request) from an entry endpoint
- Upon completion of its task, the service implementation places its output message (reply) in the exit endpoint
- Find/bind/invoke are not defined by written code but through configuration and deployment tools

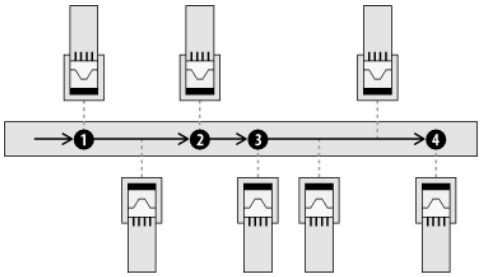
63

Message Routing

- Service definition is separated from the mechanism for locating and invoking services
- The integration architect administratively defines a composite business process flow by plugging services together into a message itinerary
- Message itinerary are to enabling a highly distributed SOA across an ESB
- Details of the itinerary…
  - are stored as metadata on the message (header)
  - carried with the message as it travels across the bus
- The ESB service container knows how to evaluate the itinerary of messages combined with configuration knowledge

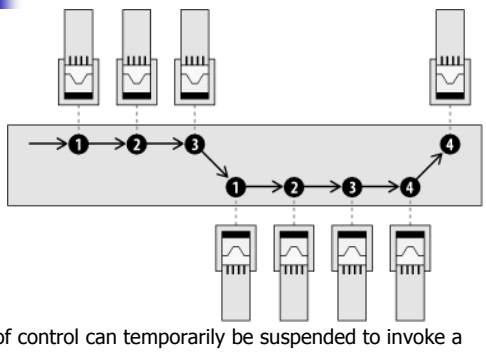
64

Itinerary-based Routing



65

Itinerary-based Routing



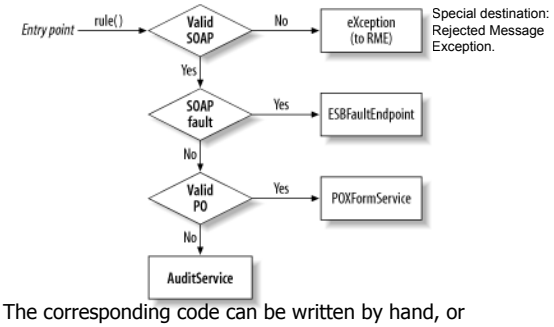
Flow of control can temporarily be suspended to invoke a subprocess, and then resumed when the subprocess returns

66

Content-based Routing

- The routing service relies on the invocation of a rule() method to perform routing logic
  - a validation step may be included
- Routing rules can be expressed and processed with different technologies
  - XML, javascript, java, BPEL, …
- A destination may be:
  - another service,
  - another process itinerary,
  - a MOM endpoint,
  - or an external web service
- Metadata and rules processed at the remote container (not by the centralized rule engine)
  - As opposed to the traditional hub-and-spoke EAI broker

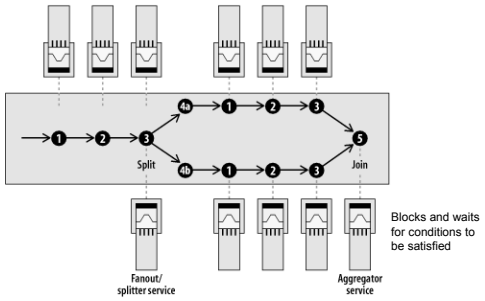
Content-based Routing



The corresponding code can be written by hand, or generated from a GUI interface.

Sophisticated Routing Strategies

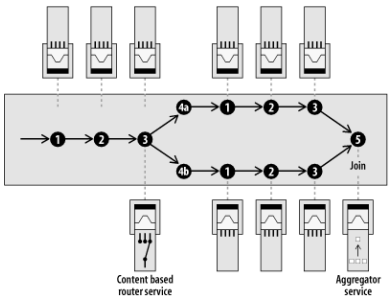
- Multi-itinerary splitter pattern



Multiple paths of execution to happen in parallel

Sophisticated Routing Strategies

- Multi-itinerary CBR pattern

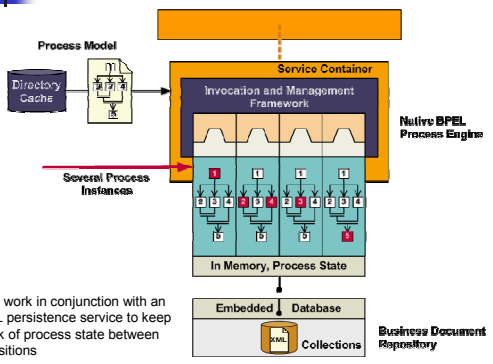


Choose between multiple possible paths

Sophisticated Process Flow

- Branching conditions could be also described using BPEL4WS
- BPEL can handle more complex situations than itineraries
- Requires an orchestration service
  - Can manage stateful information about a business process that spans a length of time
  - Business processes may include human interaction (e.g. approvals, …)
- Itineraries can carry state and recover from failures

Process Flow – BPEL



Can work in conjunction with an XML persistence service to keep track of process state between transitions

Routing – Handling Failures

- Routing rules are not treated in a centralized way
  - No single point of failure
  - Partial unavailability of the ESB network does not affect everyone
  - Configuration info can be also cached locally
- What to do with a RME message?
  - Handled by an error-handling service
  - Incorporated into an auditing and logging
  - To custom logic applied to the message based on more business rules

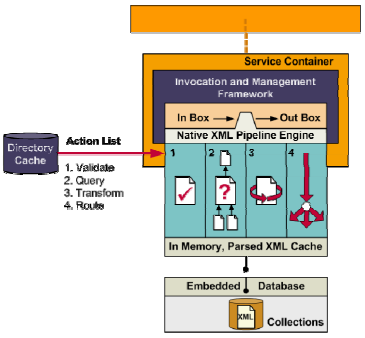
73

XML Storage Service

- Storage
  - Could be implemented using a RDBMS with XML2Rel mappings
  - Native XML may be best suited
- Storage service can be used to collect data from multiple data sources
  - Can enable BAM functionality

74

XML Server



- Multi-steps operating on the same instance of an XML doc
- Parsed only once
- Forming an XML processing pipeline

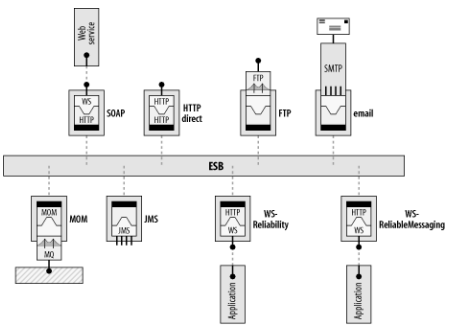
75

The ESB MOM Core

- Capable of supporting a variety of industry-standard access mechanisms and protocols
- Multi-protocol messaging bus
  - Supports async delivery of messages with configurable QoS options
  - Primary means of getting into and out of the ESB
- Scalable clustering
- Physical topology is independent of the service-oriented view of an ESB process flow
- Routing details are not exposed to the application level or the ESB process level

77

The ESB MOM Core



78

The ESB MOM Core – Benefits

- Consistent QoS levels
- Consistent management
- Metrics and monitoring
- Consistent troubleshooting techniques
- Consistent scalability methodology
- Consistent performance tuning techniques
- Consistent security and ACL model
- Publish/Subscribe broadcast of information
- Message persistence and reliability of message delivery

79

MOM Interoperability

- JMS is the only adopted standard for MOM
  - But it does not define interoperable on-the-wire format
    - The wire protocol is proprietary of a MOM vendor for providing highly efficient reliable delivery of messages
- WS-Reliability and WS-ReliableMessaging are industry efforts that define an open interoperable wire protocol for reliable messaging
  - Based on the SOAP protocol

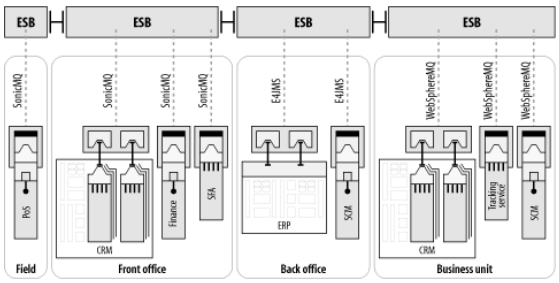
80

MOM Interoperability (2)

- Having an open protocol means that the MOM piece of software on either side of the conversation can be provided by different vendors
  - Less room for innovation on efficient delivery and reliability
- 80/20 Rule (domain dependent :)
  - 80% you need just one MOM vendor
  - 20% you'll be mixing MOM vendors via bridging technology

81

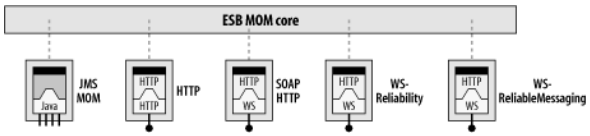
MOM Interoperability



82

Direct Protocol Handlers

- The message broker itself provides the bridging, or mapping, between the external protocol and the internal MOM channel



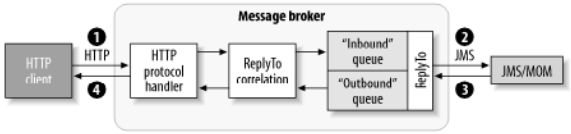
83

Direct Protocol Handlers – HTTP

- HTTP Invocation from an "external" client
  - ESB listen for inbound HTTP or SOAP requests
  - The protocol handler performs a mapping of the HTTP content into a JMS message
  - Places the JMS message into a queue or pub/sub destination
  - When the response arrives at the outbound queue it is sent back to the HTTP protocol handler

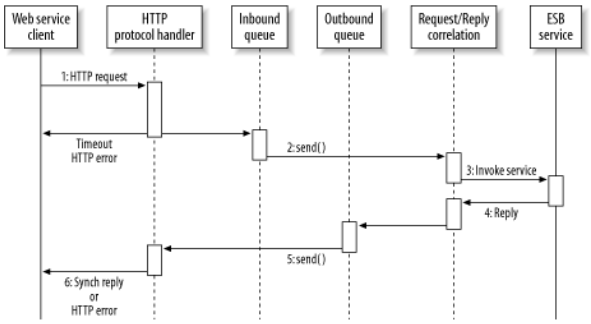
84

Direct Protocol handlers – HTTP



85

Direct Protocol handlers – HTTP



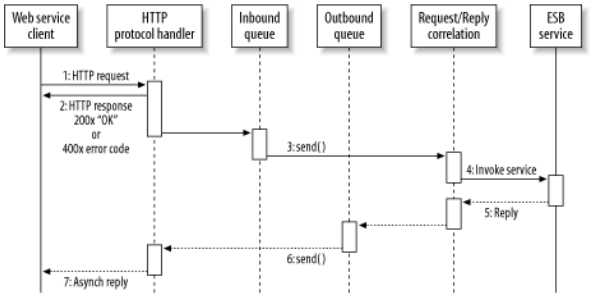
86

Direct Protocol handlers – aHTTP

- Asynchronous R/R may be required since
  - The service represents an interface to an app with regular maintenance phases
  - The service represents an app that still relies on nightly batch processing
  - The process involves some human intervention

87

Direct Protocol handlers – aHTTP



The inbound request must contain a URI indicating where to send the reply

88

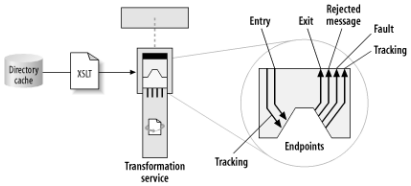
Service Reusability

- In an ESB, services are meant to be reusable
- An ESB service is an instance of a service type
  - Reusability by instantiating a particular type and applying variable data and conditions through parameterization and configuration
    - e.g. transformation service

89

Service Reusability by Config.

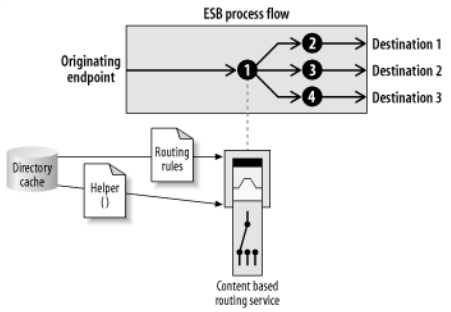
- All the endpoints are configurable
- The nature of the transform can be determined by the docs that get fed into the service



Auditing capabilities may be introduced for certain documents

90

Service Reusability by Config.

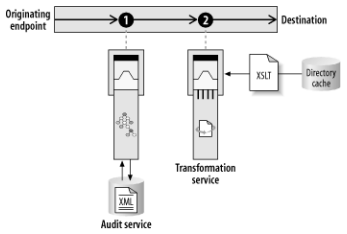


Helper functions include validation and other kind of useful functionality that can be shared among rules

91

Service Reusability by Composition

- Specialized services may be connected to form a composite business process



92

ESB Characteristics

- Pervasiveness
- Highly distributed, event-driven SOA
- Selective deployment of integration components
- Security and reliability
- Orchestration and process flow
- Autonomous yet federated managed environment
- Incremental adoption
- XML support
- Real-time insight

93

ESB – Wrap up

- Integrate across a common backbone to gain real-time access to the business data that is flowing between participants
- An app can rely on the ESB to provide a uniform, consistent approach to sending and receiving data
  - An app plugs into the bus
  - It posts data to the bus
    - It is responsibility of the ESB to get it where it needs to be
    - In the target data format that it needs to be in

94

ESB – Wrap up

- It separates the business process routing logic from the implementation of the apps that are being integrated
  - App owners do not need to worry about integration (e.g. routing, validation, …)
- Message itinerary very powerful and flexible
  - Through configuration and management tools, additional processing steps can be inserted as event-driven services
  - (processing pipeline)

95

ESB – Wrap up

- ESB supports SOA across a series of abstract endpoints
  - Endpoints can represent apps and services of any size
- ESB is a managed environment
- ESB provides facilities that make the development, deployment and maintenance of integration services easy
- Unified view of security
- ESB MOM core as the basic infrastructure to build higher-level SOA
  - The details of creating and managing the messaging channels or setting QoS options are encapsulated in a contained-managed environment
    - Is configured instead of writing application code

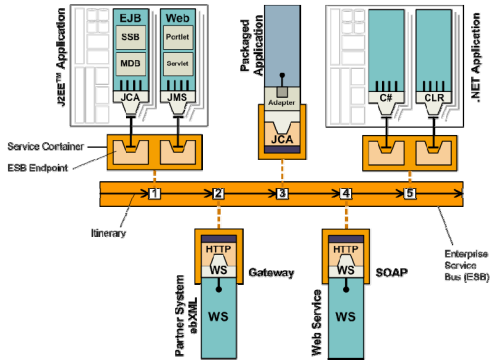
96

ESB – Wrap up

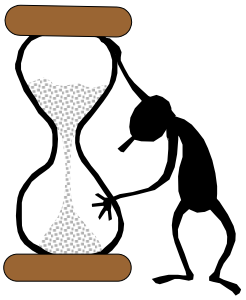
- Traditionally when dealing with MOM apps need to written code that
  - Establishes connection with the message server
  - Creates publishers, subscribers, queue senders and receivers, managing transactional demarcation and recovery from failures
- An ESB removes this complexity by delegating that responsibility to the ESB service container
  - Container relies on configurations

97

ESB – Big Picture



98



99