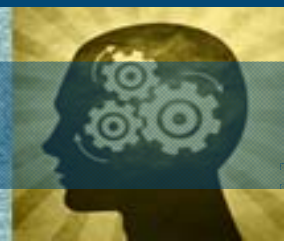
Fiorano

Enabling change at the speed of thought

ESB BEST PRACTICES

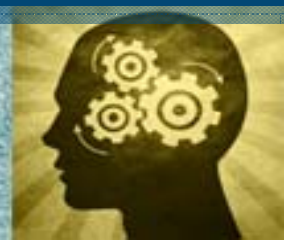


*THREE INTEGRATION SCENARIOS:**

1. **DATA CONSISTENCY:** INDEPENDENT APPLICATIONS “GET THE FACTS STRAIGHT”
2. **MULTI-STEP PROCESS:** INDEPENDENT APPLICATIONS IMPLEMENT A BUSINESS PROCESS
3. **COMPOSITE APPLICATION:** NEW FUNCTIONALITY BY TYING TOGETHER EXISTING APPLICATIONS

•Source : Gartner Inc. Gartner Research ID G00164724.

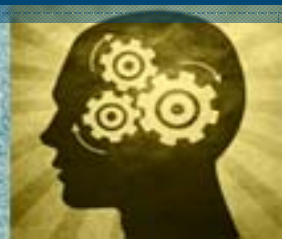
AGENDA

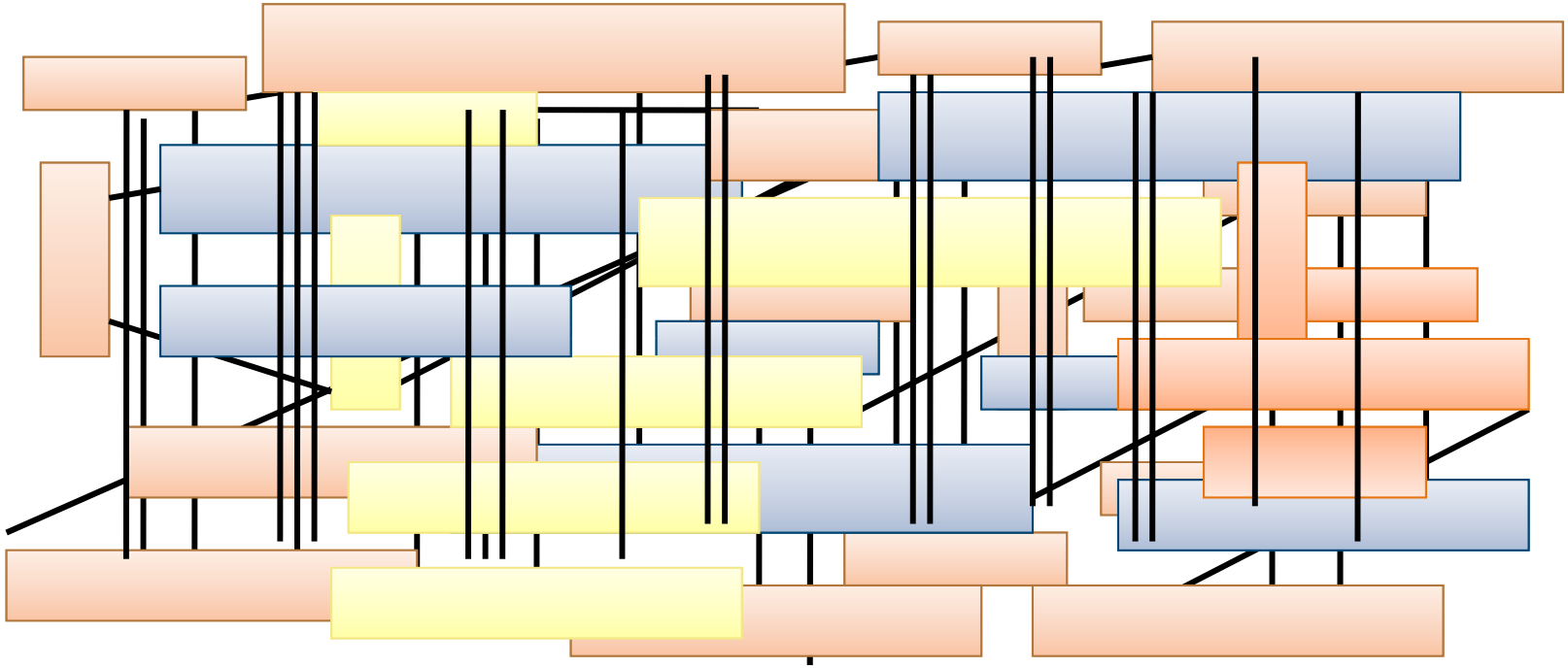


- INTRODUCTION TO SOA
- DEMYSTIFYING ESBs
- ESB/SOA IMPLEMENTATION METHODOLOGY
  - STEP I – REQUIREMENTS GATHERING
  - STEP II – IDENTIFYING “MUST HAVE” FEATURES
  - STEP III – COMPONENTIZING BUSINESS PROCESSES
  - STEP IV – DEFINING DISTRIBUTED ESB PROCESSES
  - STEP V – DEPLOYING ESB PROCESSES
  - STEP VI – LAUNCHING AND MONITORING ESB PROCESSES
  - STEP VII – CHANGE MANAGEMENT, VERSION ROLLOVER
- INTEGRATION SCENARIOS – PUTTING THEORY TO PRACTICE

INTRODUCTION TO SOA

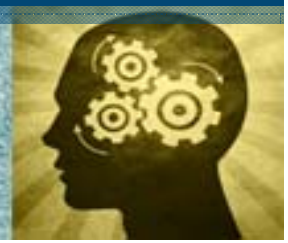
POINT-TO-POINT INTEGRATION





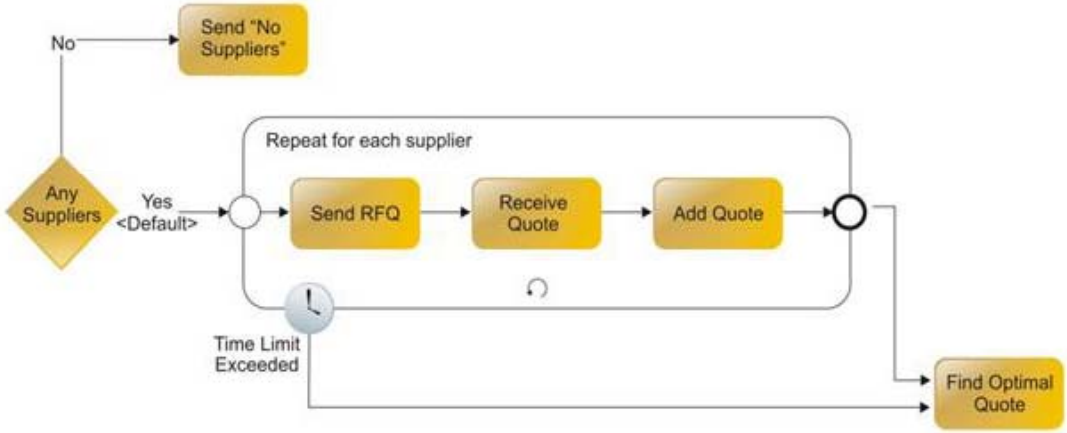
- "Accidental" Architecture (Synchronous, Fine-grained, Not scalable, Many connections and data formats, Extremely difficult to manage, Expensive to maintain and extend)
- Hinders business change: new products, channels, suppliers, etc
- Slow and expensive maintenance, No Reusability
- Expensive to Manage, Monitor and Extend
- Need realistic way to evolve new infrastructure and add value

BUSINESS PROCESSES BECOME AGILE WHEN IMPLEMENTED AS SERVICES



Business Perspectives

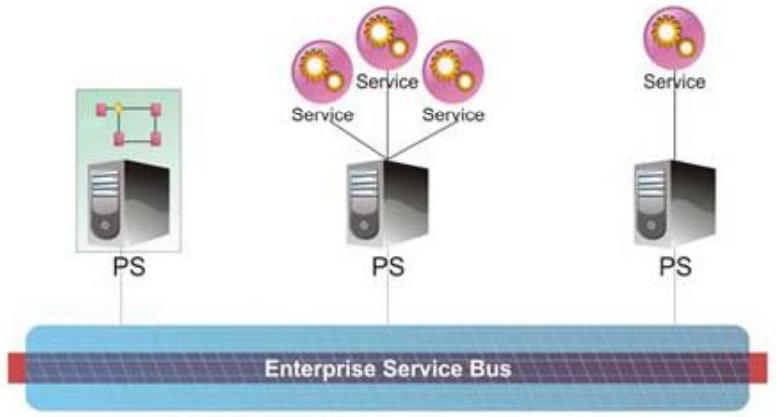
- Simplification
- Elimination
- People
- Process Performance Measurement
- Simulation modeling
- Agility
- etc.



BPMN Notation representing a business process with an expanded Sub-process

Systems Perspectives

- Working Software
- Reliability
- Configuration Management
- Scalability / Performance
- Reusability
- Speed of Delivery
- etc.

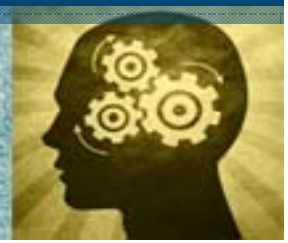


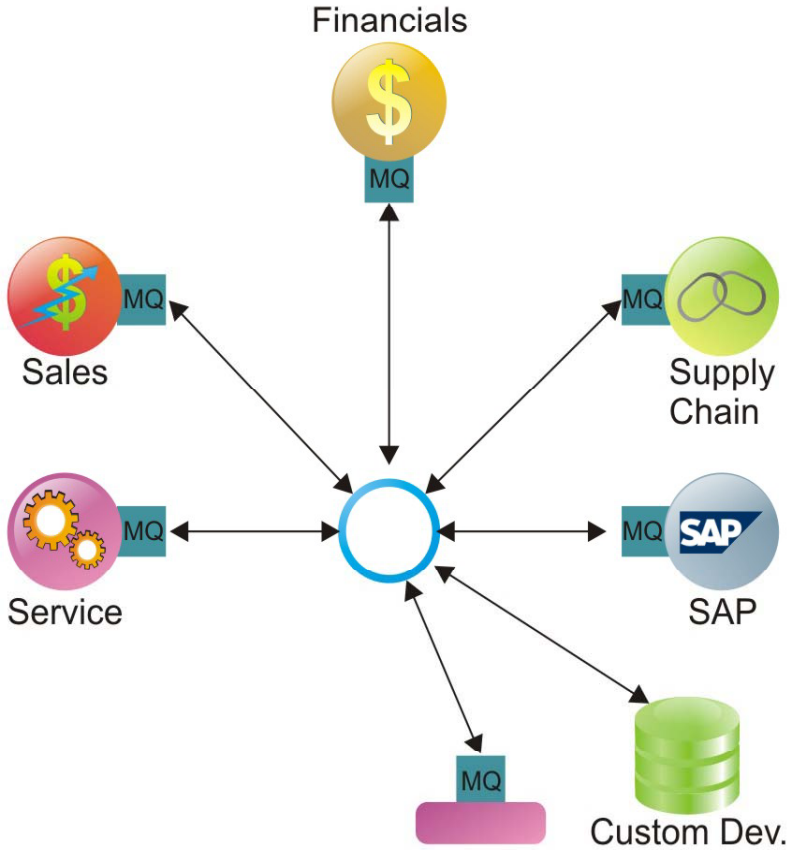
Example: Processes implemented as Services

With SOA – Processes Can Be Layered on Top of Existing Infrastructure

INTRODUCTION TO SOA – Message Oriented Middleware (MOM)

CHALLENGES

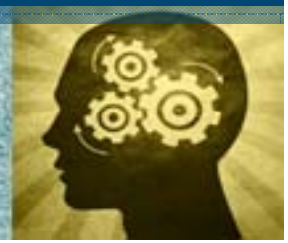




- Easy Inter-Operability
- Common Data Model for integration
- Robust Process Management
- Scalability and High-Availability
- Distributed Services Management – Real time configuration changes
- Reusability of Components & Services in an effective IDE
- Portable Adapters
- Cost, Speed, Quality & Control

INTRODUCTION TO SOA - WEB SERVICES

CHALLENGES

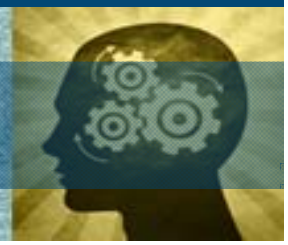


- Suffer from incomplete and competing standards definitions that might not ensure easy interoperability across different implementations
- Interoperabilty
  - ✓ SOAP
  - ✓ Only binding for HTTP; no bindings for SMTP, JMS
  - ✓ “SOAP encoding”: XML ←→ Objects
  - ✓ RosettaNet, EDIINT: no SOAP
  - ✓ SOAP 1.1 → 1.2 (W3C)
- WSDL: RPC/Encoded vs. Document/Literal
- Do not offer a complete package but provide only the functionality of access and invocation leaving majority of the development work to the user
- Business Semantics – Additional tools needed for Orchestration & Choreography
- Security
  - ✓ HTTPS is not enough
  - ✓ No agreement regarding attachments
  - ✓ S/MIME in RosettaNet and EDIINT
  - ✓ Single Signon
- Transactional Integrity
- Reliable Asynchronous Message Handling
  - ✓ WS-Reliability (IBM, MS) vs. WS-ReliableMessaging (OASIS)
- Transformation Services

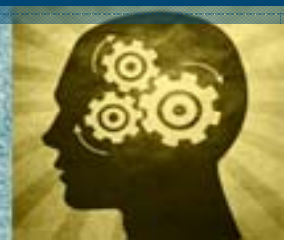
Fiorano

Enabling change at the speed of thought

DEMYSTIFYING THE ESB



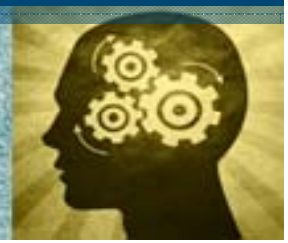
DEMYSTIFYING ESB

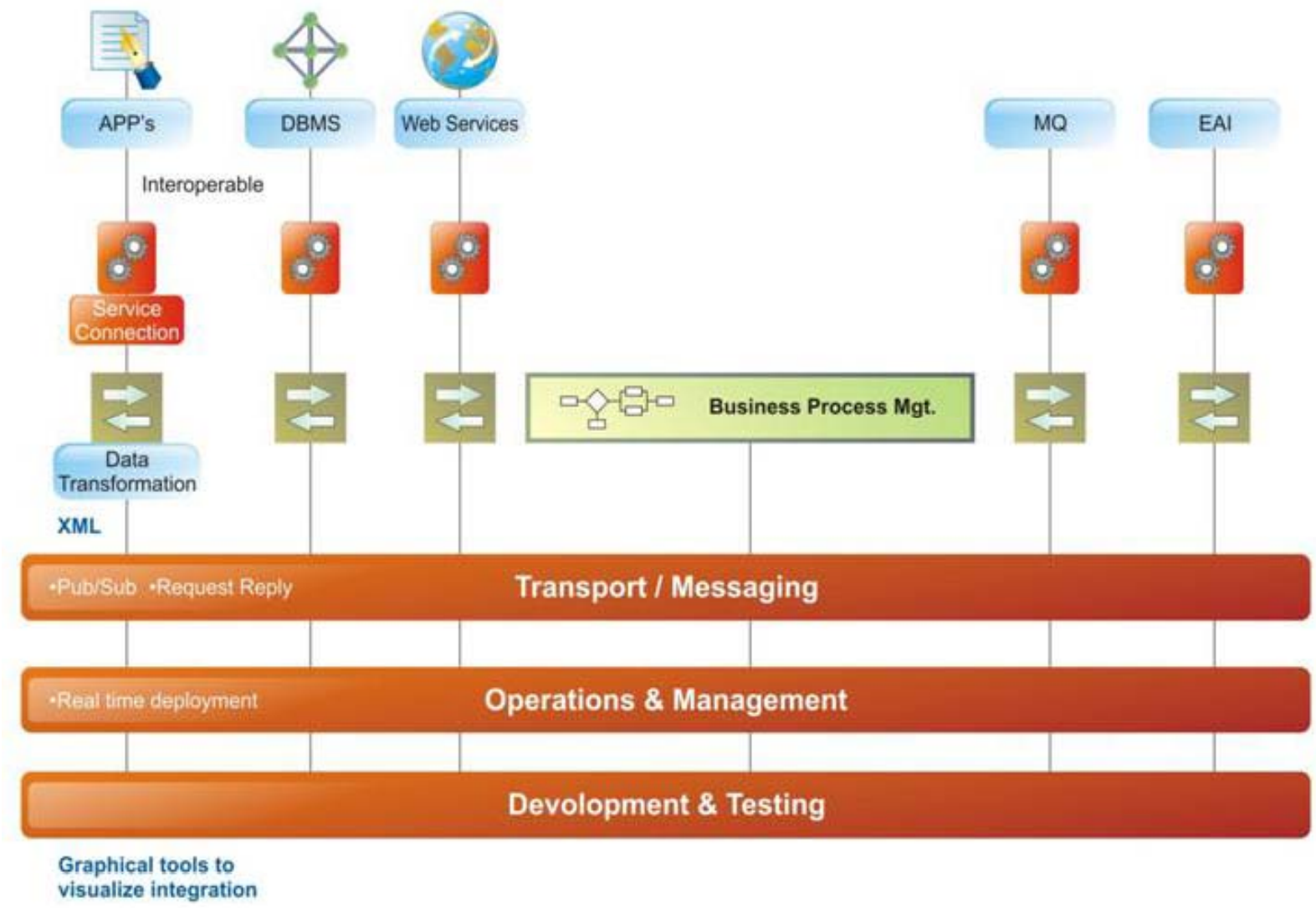


Enterprise Service Bus evolved out of SOA

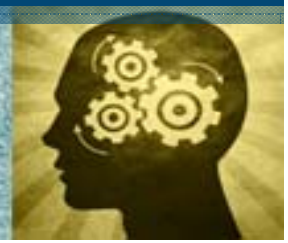
- IT Components can be accessed as services
- Defined form of invocation and entry points to service
- Business process - Event-driven/Asycnhronous Inovcation of Services
- Composite Application – mix of existing and new components

FUNDAMENTALS OF SOA FOR INTEGRATION





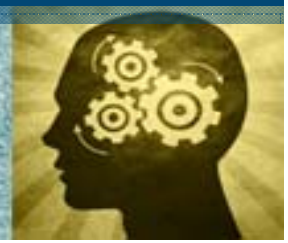
DEMYSTIFYING ESB



Definition

An ESB is a standards-based platform that allows applications on a network to be accessed via Services (including Web-Services and REST-based Services). ESBs combine messaging, Web Services, XML, data transformation and management to reliably connect and coordinate application interaction. The ESB deployment model is an integrated network of collaborating service instances, deployed in distributed service containers.

DEMYSTIFYING ESB



Enterprise Service Bus – Standards based Integration

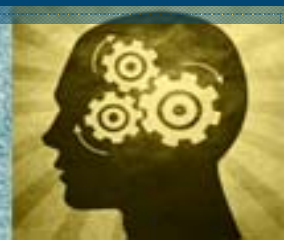
- Communication and data routing (JMS)
- Data protocols (XML)
- Transformation (XSLT)
- Content Based Routing (CBR)
- Connectivity (via JCA, .NET, Web-Services, REST)
- Web-Services; REST-baed Services
- Security
- Pre-built Business Components and Connectors

Related Infrastructure and Concepts - not explicitly part of an ESB

- *Business Process Management (BPM)*
- *Business Process Modelling (BPMN, YAWL or Equivalent)*
- *B2B – trading partner management*

WHY STANDARDS FOR INTEGRATION?

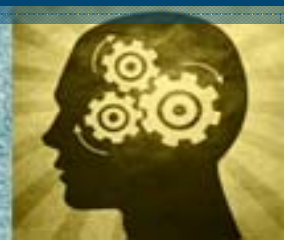
BUSINESS AND TECHNOLOGY DRIVERS



- Increases ability to integrate
  - ✓ No platform or vendor technology dependencies
- Lowers cost of integration
  - ✓ Minimizes need for expensive proprietary adapters
  - ✓ Larger talent pool, broader education offerings
- Integration projects become more predictable

STANDARDS-BASED INTEGRATION

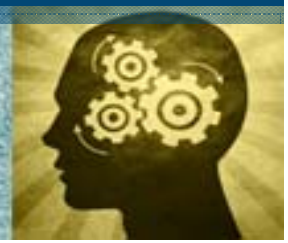
ENTERPRISE BACKBONE



- Deploying a Service-Oriented Architecture (SOA)
- Use of XML
- Rise of JMS as the de-facto standard for underlying application-to-application communication
- Event-flow processes for business process deployment
- Connections to Packaged Apps, Legacy Systems using J2EE Connector Architecture (JCA), Web-Services, and JMS-compliant connectors
- Web Services

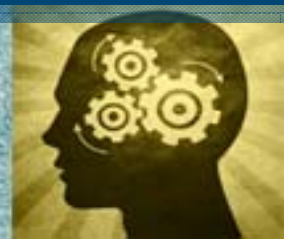
SERVICE-ORIENTED ARCHITECTURE (SOA)

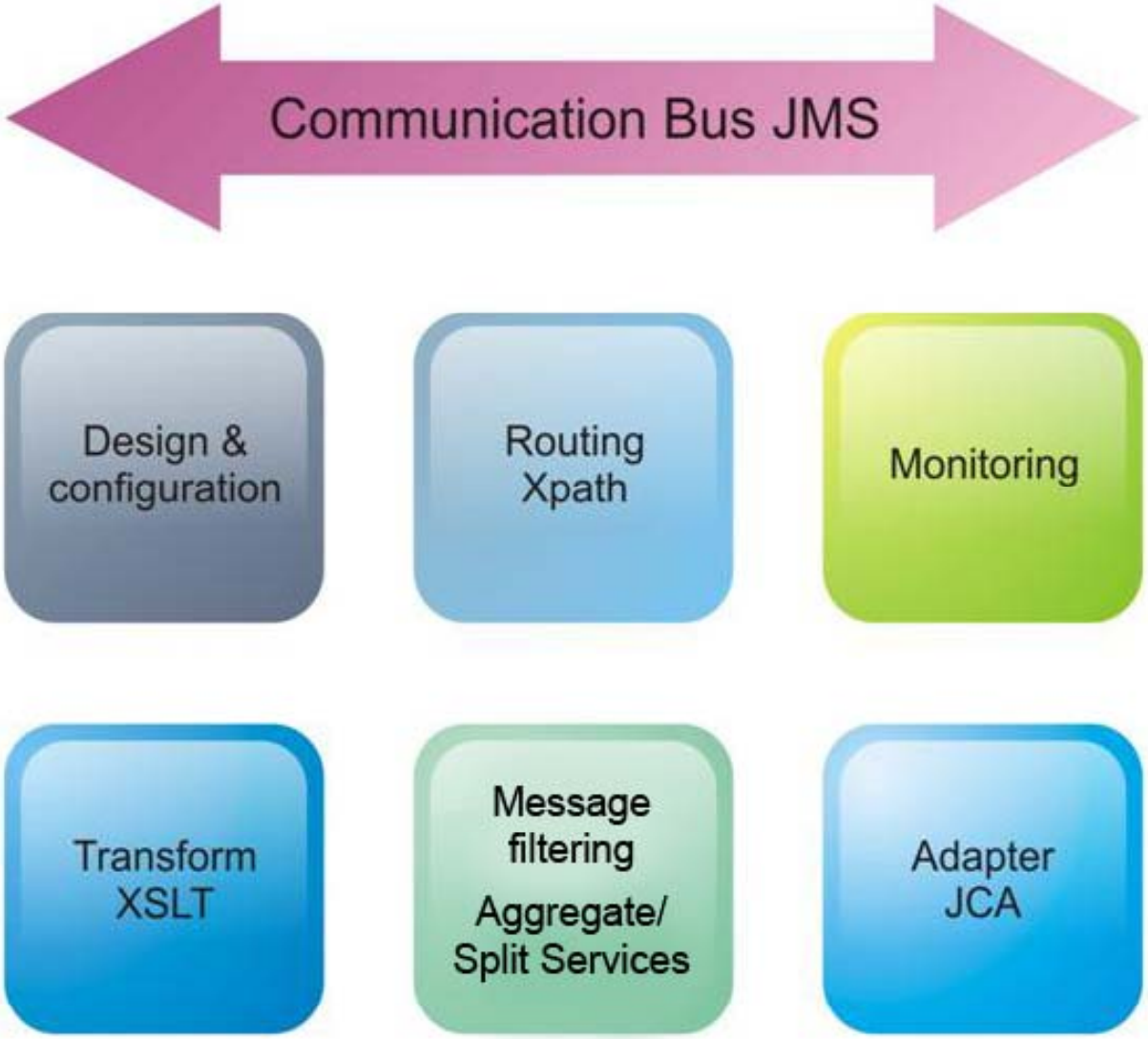
FLEXIBLE, EXTENSIBLE INTEGRATION



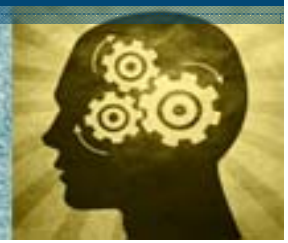
- Design methodology for distributed systems
- Applications expose functionality through service interfaces
- Loosely-Coupled
- Platform and language neutral
- Impervious to implementation changes
- Coarse-Grained- business level Interfaces
- Asynchronous
- No single points of failure

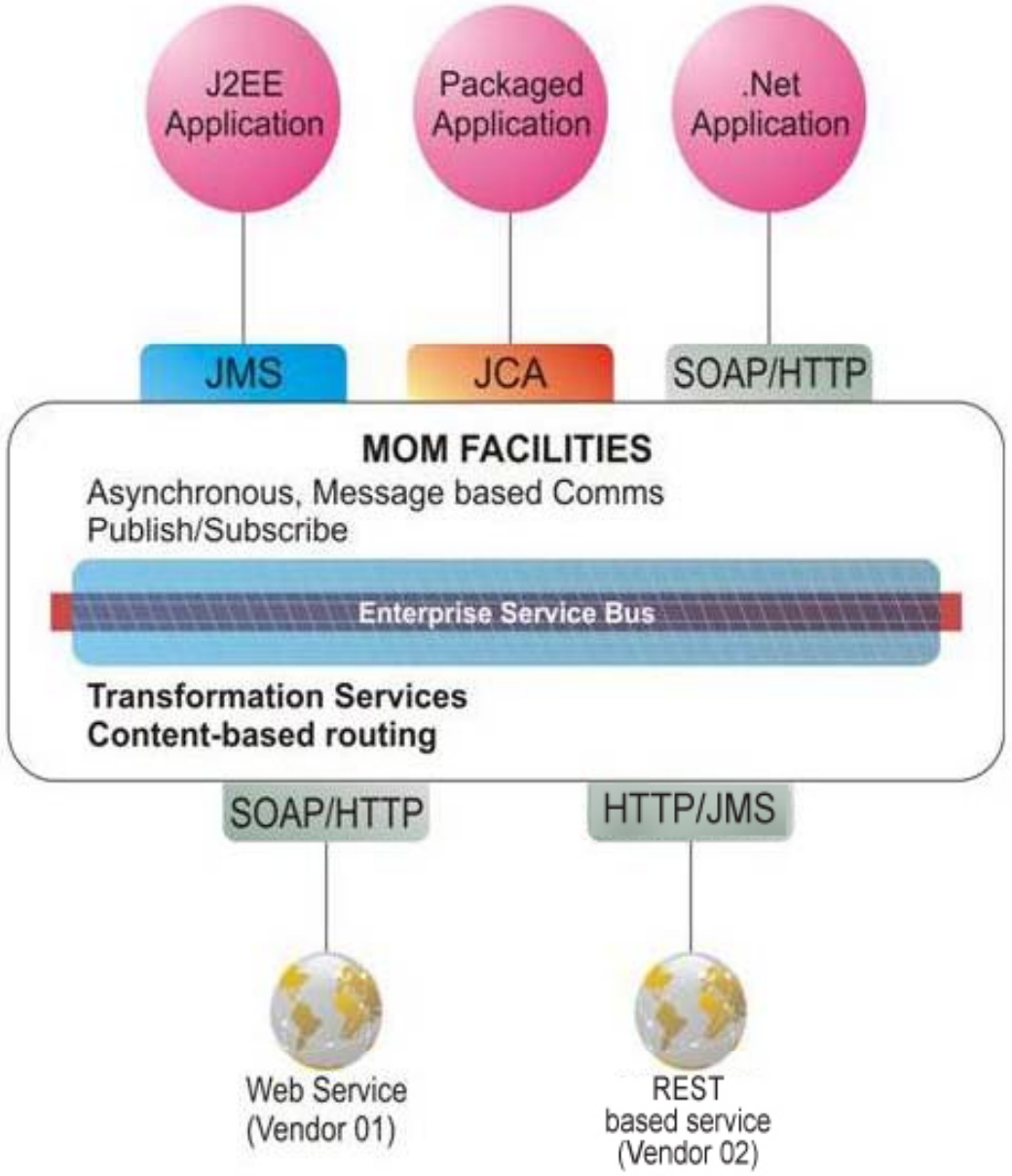
DEMYSTIFYING ESB



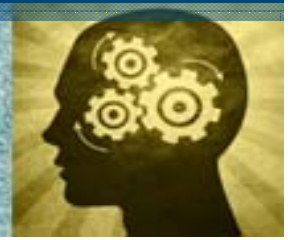


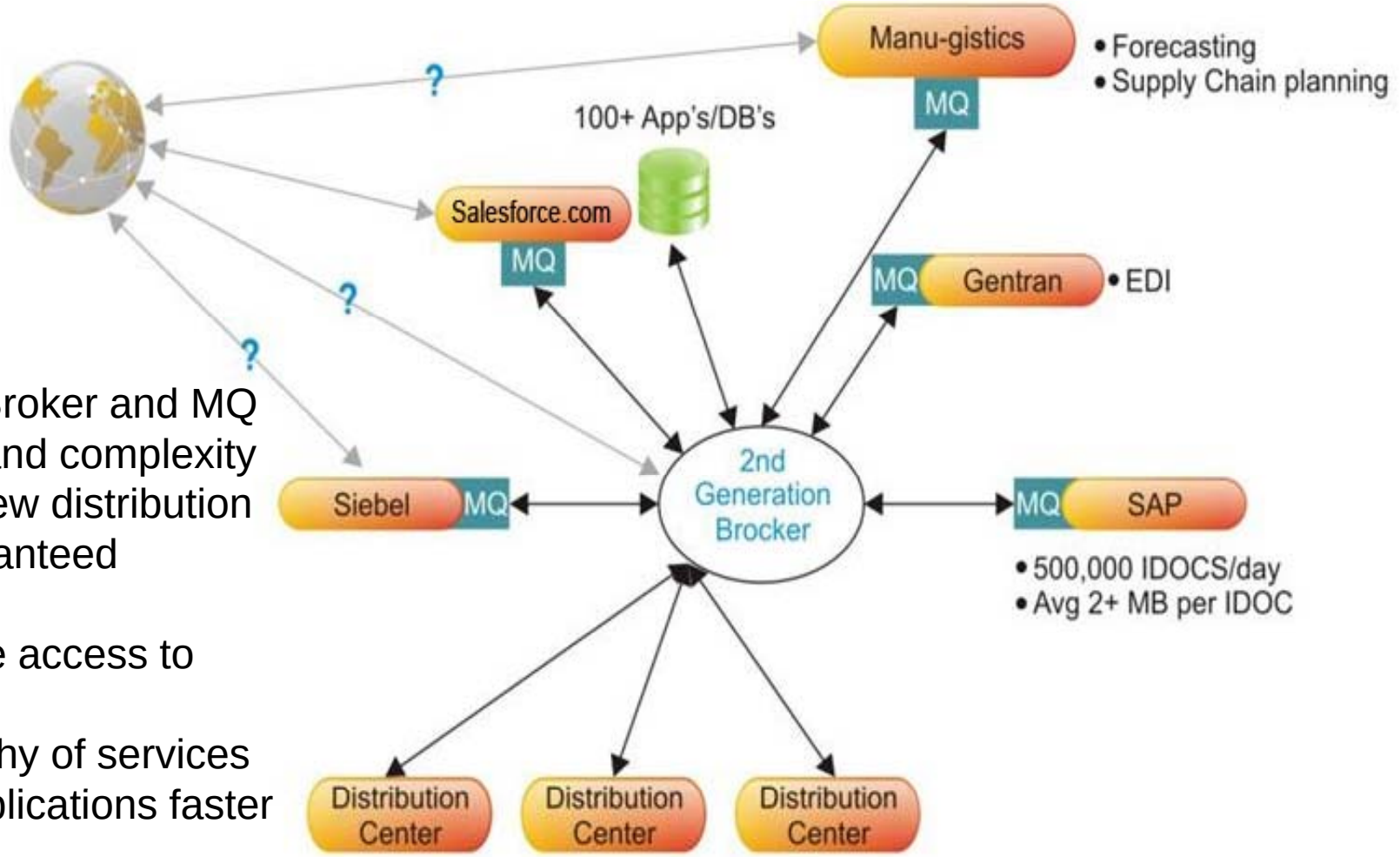
DEMYSTIFYING ESB





EXAMPLE: BUILD ON EXISTING INFRASTRUCTURE

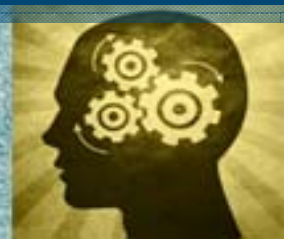




Business Needs

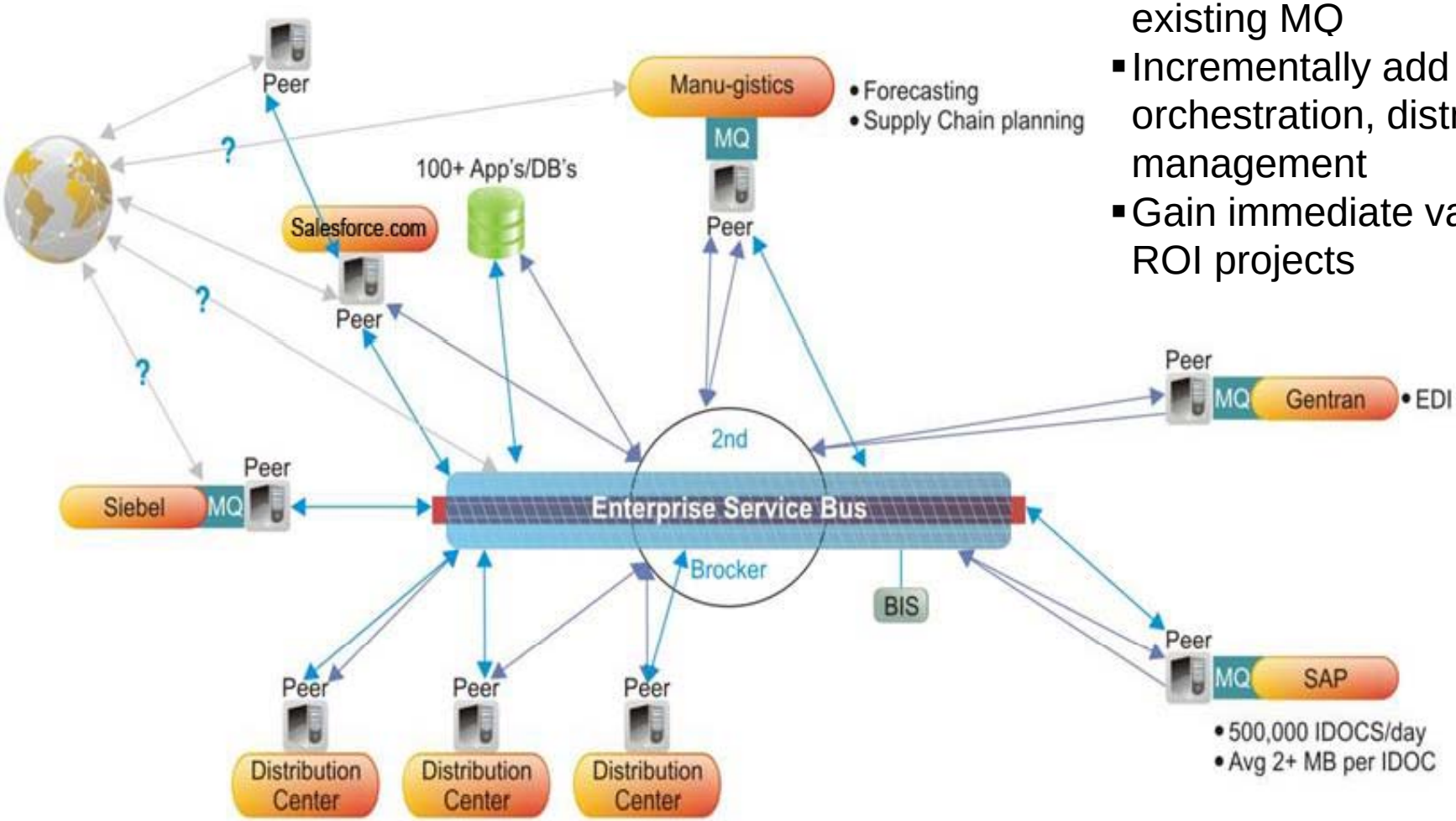
- Want to replace Broker and MQ due to high TCO and complexity
- Reduce cost of new distribution centers, with guaranteed messaging
- Need web-service access to data
- Need choreography of services
- Integrate new applications faster and cheaper

EXAMPLE: BUILD ON EXISTING INFRASTRUCTURE



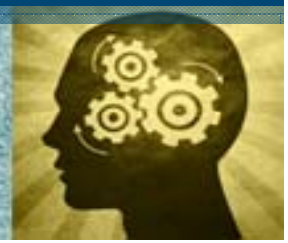
Solution

- Services Oriented Integration
- Reuse messages/events with existing MQ
- Incrementally add transformation, orchestration, distributed management
- Gain immediate value from higher ROI projects



Fiorano
Enabling change at the speed of thought

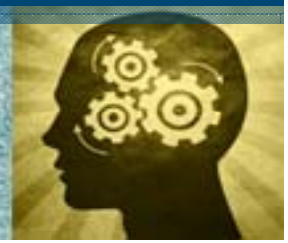
DEMYSTIFYING ESB



ESB's are best suited for:

- Projects that will mix heterogeneous application services (for example, Microsoft or Java portals with disparate Java or Microsoft server back ends)
- Enterprises that want to start with a basic SOA and add other features later, as the implementation evolves
- Enterprises that want to assemble their own best-of-breed comprehensive integration suites
- Mix and match off-the-shelf adapters, BPM, B2B, and BAM tools from other vendors
- Distributed services (written in different programming languages) running on disparate nodes on different operating systems

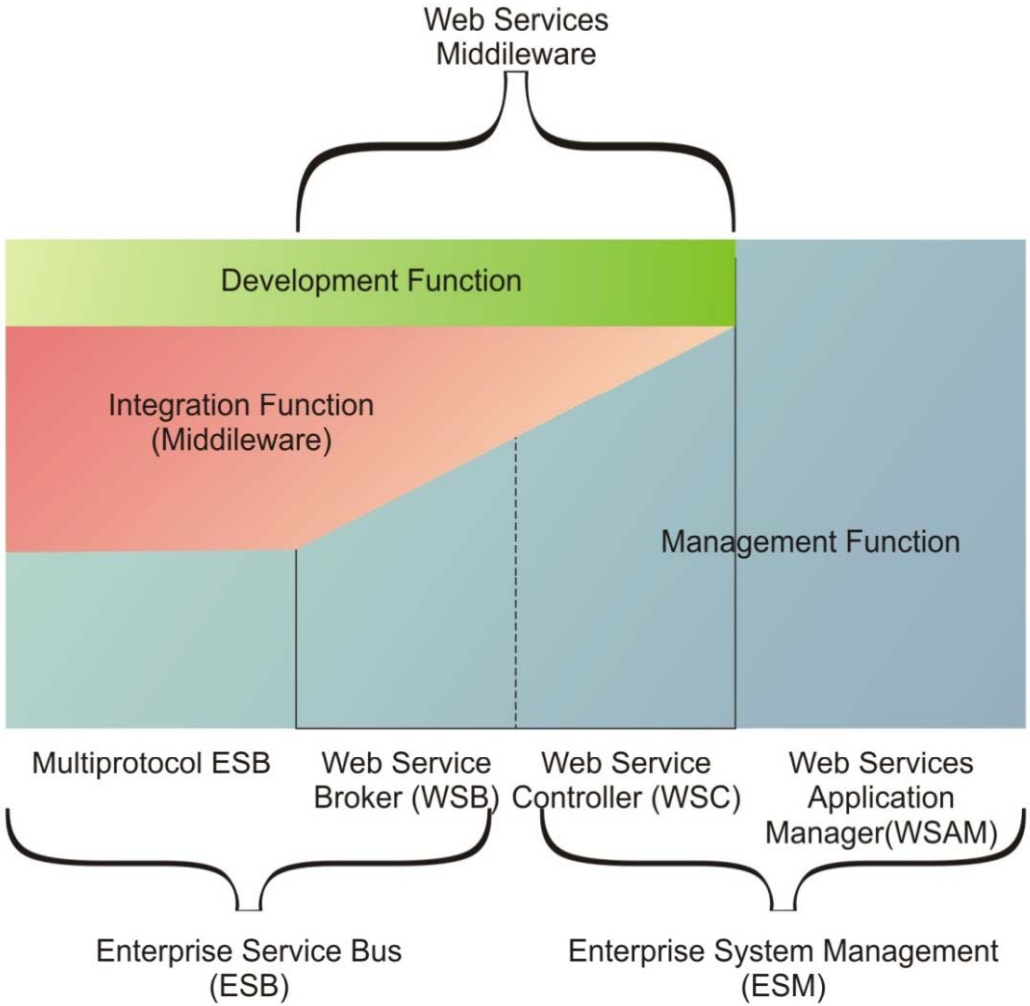
DEMYSTIFYING ESB



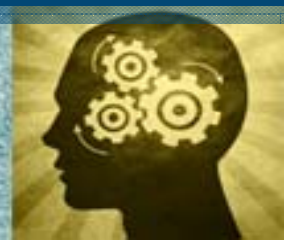
Types of ESB:

1. ESBs based solely on SOAP -Web services brokers (WSBs)

2. Multiprotocol ESBs that support JMS, Web services and other communication mechanisms



ESB - VENDORS (2012)



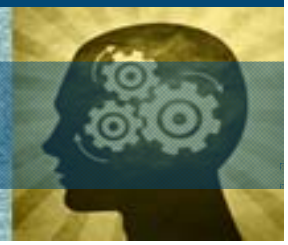
Support SOAP/HTTP and additional protocols guaranteed delivery, publish-and-subscribe often following the JMS standard

ESB	Comments
Fiorano ESB	Native support for JMS, Peer-to-Peer architecture; REST-based Services
IBM WebSphere ESB	BPM-centric Hub/Spoke architecture; Non-native support for JMS
Oracle Service Bus	Based on BEA acquisition, BPM-centric Hub/Spoke architecture
TIBCO ActiveMatrix Service Bus	NatDistributed Architecture, but heavyweight and BPM-centric; native JMS support;
Microsoft BizTalk Server	Monolithic EAI broker - not a true ESB, Microsoft environment centric.

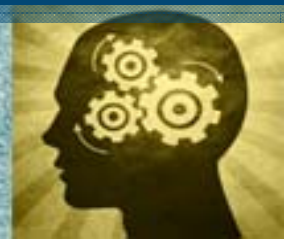
Fiorano

Enabling change at the speed of thought

ESB IMPLEMENTATION METHODOLOGIES

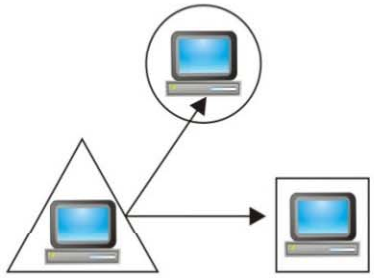


3 BASIC INTEGRATION PATTERNS



DB SYNCHRONIZATION

Data Consistency

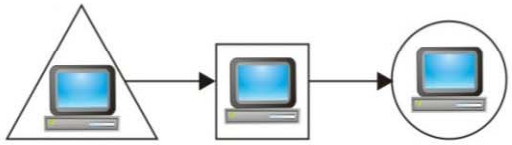


- Multiple Processes
- Parallel unrelated steps
- One-Way, asynchronous
- Batch or Immediate
- Physically Independent
- Logically dependent

Source: Gartner Group

ORDER FULFILLMENT APPLICATION

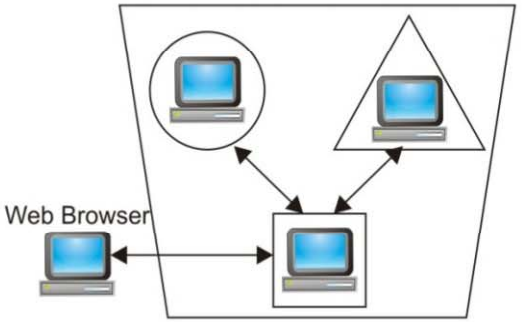
Multistep Process



- One Business Process
- Multi Steps
- One-Way, asynchronous
- Batch or Immediate
- Physically Independent
- Logically dependent

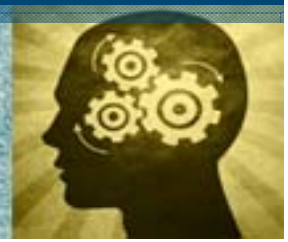
QUOTE TO BIND APPLICATION

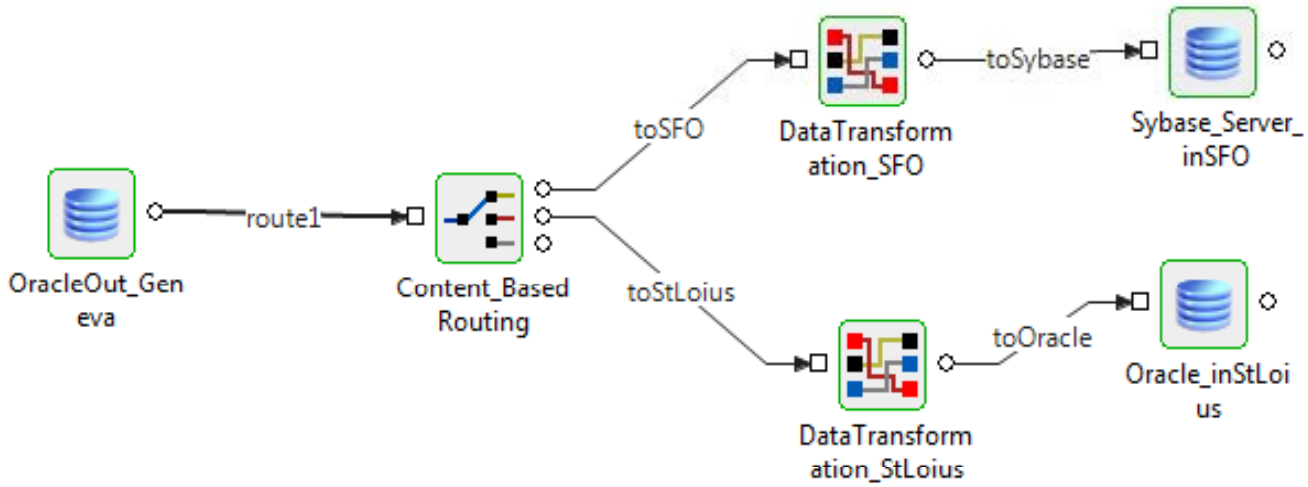
Composite Application



- One Business Process
- One Step
- Two-Way, synchronous
- Physically Independent
- Logically dependent

STEP 1 – REQUIREMENTS GATHERING
DATA CONSISTENCY – DATABASE SYNCHRONIZATION





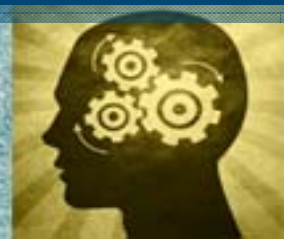
The technical requirements of ABC Corporation can be summarized as

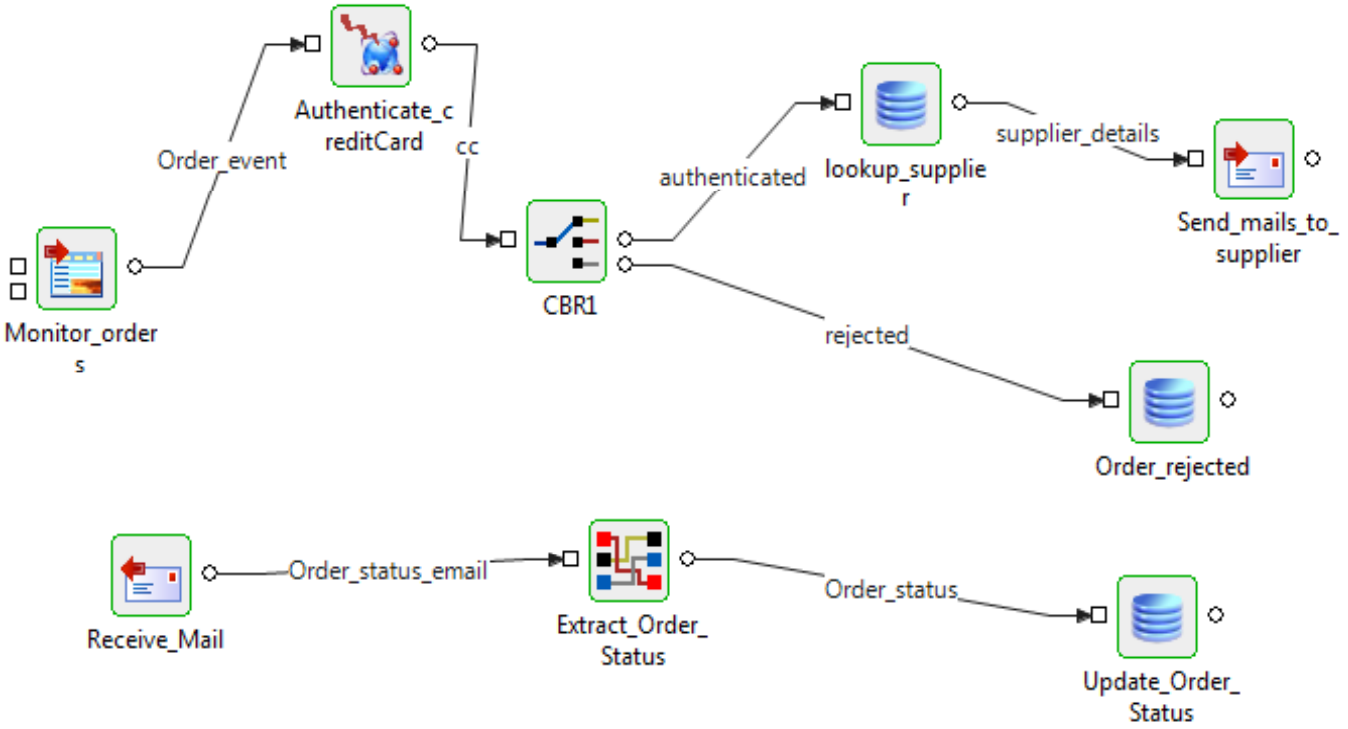
- Data needs to be replicated across multiple data centers asynchronously using a Message Bus.
- Any changes made to the Oracle Database instance in Geneva need to be reflected (based on certain selection criteria) in either the Sybase database instance or the MSSQL database instance located in San Francisco and St.Louis respectively.
- The data needs to be transformed from the source table data format to target table data format(s).
- All data transfers need to be secure

Key Characteristic: the data flow between steps is asynchronous and one-way

STEP 1 – REQUIREMENTS GATHERING

MULTI STEP PROCESS – ORDER FULFILLMENT



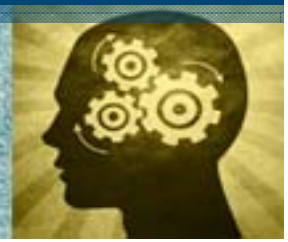


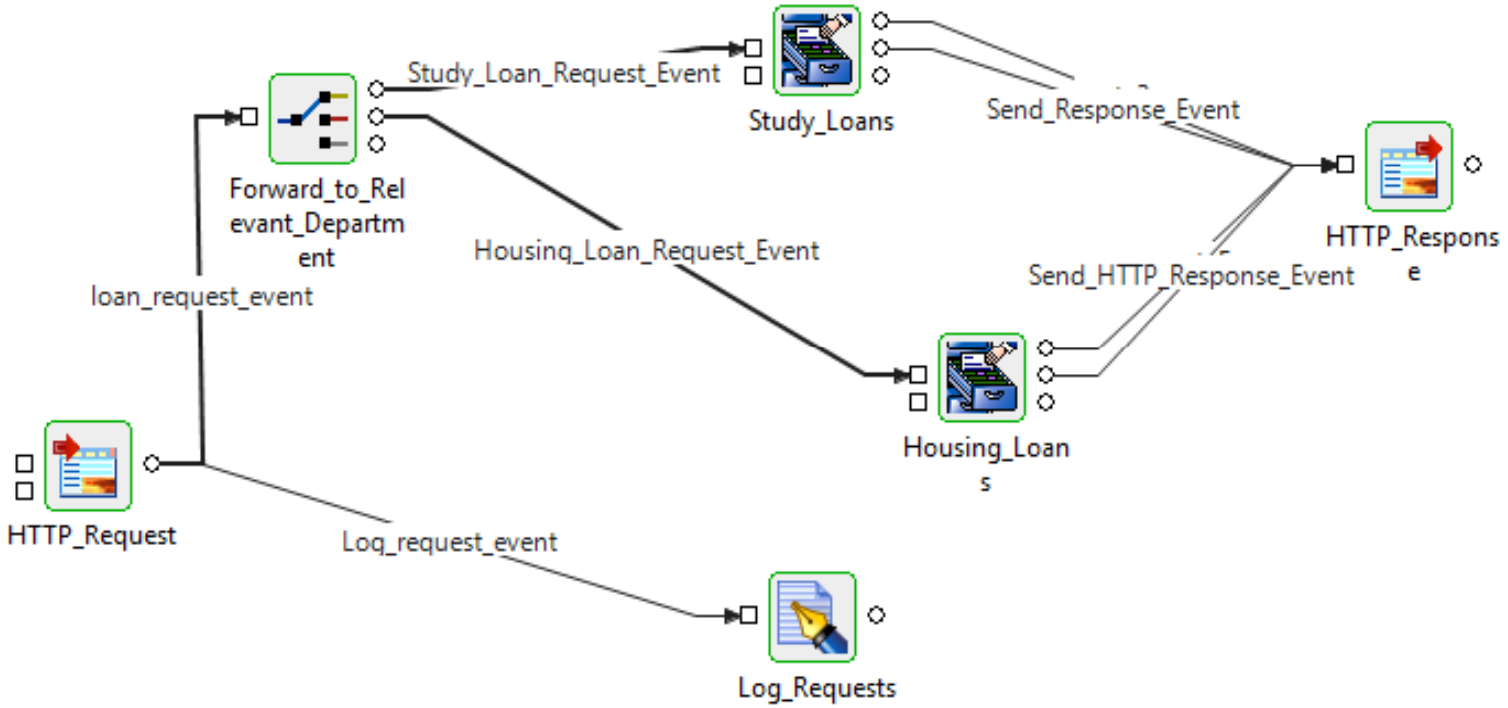
Company X runs an online market place for electronics products. Orders are accepted over the web and saved in an Oracle DB. To process an order, the company needs to perform credit-card verification using a third party hosted credit card gateway and then send out orders to suppliers over email. The supplier sends back an acceptance or rejection notification of the order (along with expected delivery schedule) by a return email. This information needs to be updated in the Oracle DB, so that the customer can track the status of the order online.

Key Characteristic: the data/event flow between steps is asynchronous and one-way

STEP 1 – REQUIREMENTS GATHERING

COMPOSITE APPLICATION – QUOTE TO BIND



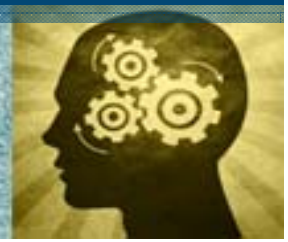


End to End business process for opening a new insurance policy that includes Capture of customer information, Risk Evaluation requestor, Processing of the Application, Development of a Price Quote and finally, a Response (Quote) to the user.

Key Characteristic: The completion of each step is predicated on the completion of the next step and the calling sequence is request/reply between steps.

STEP 1 – REQUIREMENTS GATHERING

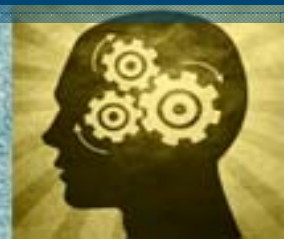
BUSINESS – DATA SYNCHRONIZATION



- Data synchronization for Disaster recovery planning and business continuity, Content distribution, Backup consolidation, and Server migration
- Completely eliminate all manual data synchronization without having to stop the system to accommodate new outlets or changes to the tables in the source or target systems
- Business process monitoring Dash Board
- Add new retail outlets
- Business users should be able to rapidly create and deploy business processes
- Timelines – immediate
- Reduce development, maintenance costs

STEP 1 – REQUIREMENTS GATHERING

TECHNICAL/SYSTEMS – DATA SYNCHRONIZATION

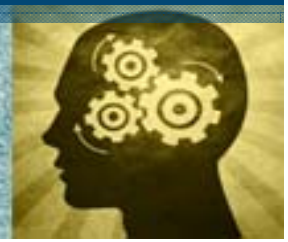


- TECHNICAL
  - High Level Business process representation – ideally, the model should map as rapidly as possible to the final application solution
  - Databases (SQL, Oracle, DB2 and files) across disparate OS
  - Event/Data flow representation
  - Intended users –
    - Business Analyst (Choreography and Execution)
    - System Administrator (setup, configuration, data transformation)

- SYSTEM
  - Platforms and Operating Systems
  - Overall network topology

STEP 1 – REQUIREMENTS GATHERING

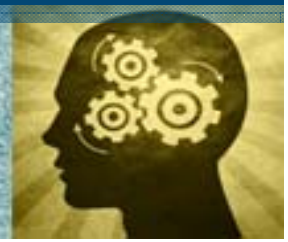
OUTPUT DOCUMENTS



- High-Level Design documents that address the integration requirements (business & technical)
- Document all the identified data and process flows
- Create a list of different networks, and the servers available in each of these networks
- Recommended hardware specifications for the specific ESB product
- Any 3rd party software that is essential to the functioning of the ESB

STEP 2 – IDENTIFYING "MUST HAVE" FEATURES

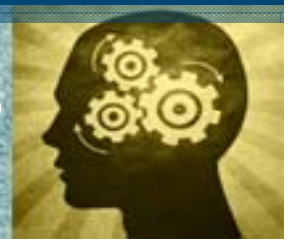
DATA CONSISTENCY



- Architecture
  - SOA ? (Hub & Spoke, Peer to Peer, Brokered Peer to Peer)
    - ✓ Brokered P2P with store and forward at end points of the network
    - ✓ Automatic reconnects in case of network failures
    - ✓ In-built store-and-forward
- Supported Standards
    - ✓ XML, JCA, CBR
- Message Bus (Asynchronous, Transformation, etc)
    - ✓ Asynchronous, Transformation (at source or destination)
- Support for Distributed Applications (Compose, Execute and Monitor distributed Apps)
    - ✓ Location and Technology transparency, Intelligent routing, Single point of control, Deployment support
- Connectivity services (Web services,J2EE Connectors, JMS, WebSphere MQ)
    - ✓ JMS, Database Connectivity (JDBC)

STEP 2 – IDENTIFYING "MUST HAVE" FEATURES

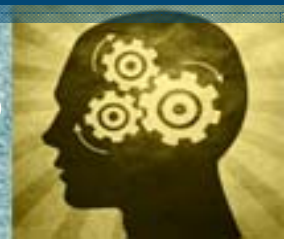
DATA CONSISTENCY



- Administration / Deployment
  - ✓ Single point of control
  - ✓ Dynamic changes to application processes
  - ✓ Single point of control
  - ✓ Remote access capability
  - ✓ Start / stop facilities
  - ✓ Manual routing support
  - ✓ Tracing
  - ✓ Message editing
- Monitoring
  - ✓ Problem determination
  - ✓ Problem prediction
  - ✓ Internal and external support
  - ✓ Support for enterprise management frameworks

STEP 2 – IDENTIFYING "MUST HAVE" FEATURES

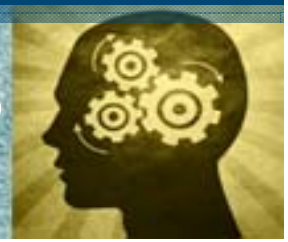
DATA CONSISTENCY



- Robustness (Service and the Infrastructure level)
  - ✓ Fault avoidance
  - ✓ Fault tolerance
- Scalability and Performance (Service and the Infrastructure level)
  - ✓ Asynchronous messaging
  - ✓ Multi-threading
  - ✓ Load balancing
  - ✓ Large data handling
- Security (Infrastructure and Application level)
  - ✓ Access control
  - ✓ Information security
  - ✓ Tools usage

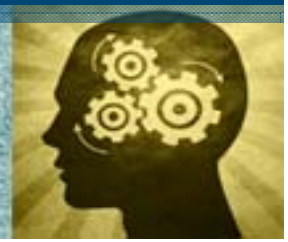
STEP 2 – IDENTIFYING "MUST HAVE" FEATURES

DATA CONSISTENCY



- Breadth of Connectivity (Configure, Modify, Connect with other Adapters)
  - ✓ DBMS access (Oracle, SQL, DB2, etc)
  - ✓ Legacy systems (Mainframe Applications)
  - ✓ Application servers
  - ✓ .NET
  - ✓ COM / CORBA - Multi-Language Adapters
  - ✓ WebServices (Publisher and Consumer)
- Tools
  - ✓ Configuration (Business Processes, Services, Infrastructure)
  - ✓ Incremental deployment
  - ✓ Lifecycle support

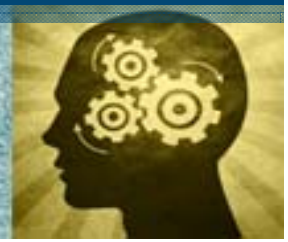
STEP 3 – COMPONENTIZING BUSINESS PROCESSES



- Component/Service/Business Service – Definition
- Fine-Grained Components - Issues
- Coarse Grained Components – Advantages
- Pre-built Components - Ease the rapid deployment of processes
- User Defined Components – Best practices
- Reusing existing resources

STEP 3 – COMPONENTIZING BUSINESS PROCESSES

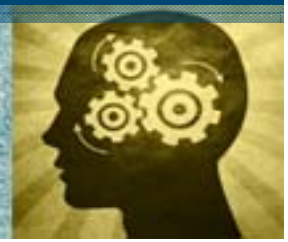
FINE GRAINED COMPONENTS - ISSUES/PROBLEMS



- Typically represent function call (static input and outputs)
- Low reusability (impossible to build a general purpose Database Adapter using a “Fine Grained” component design approach)
- Development overheads (tight coupling between client and component)
- Change of service functionality needs re-coding (Static data formats for input and output)
- Does not map well to business process composition
- Synchronous invocation of function calls
- Skilled developers needed to use fine-grained components

STEP 3 – COMPONENTIZING BUSINESS PROCESSES

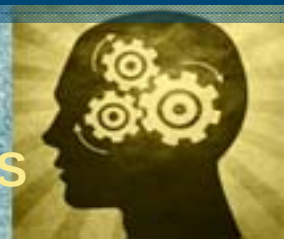
COARSE GRAINED COMPONENTS - ADVANTAGES



- Represents a high level business component
- Easy to Orchestrate and/or Choreograph business processes using these components
- Reusable across business process (changes required only in design time properties)
- Dynamic data formats for input and output
- Synchronous & Asynchronous invocation
- Low development cost (middleware is hidden)
- Business Analysts can orchestrate business processes without IT Intervention

STEP 3 – COMPONENTIZING BUSINESS PROCESSES

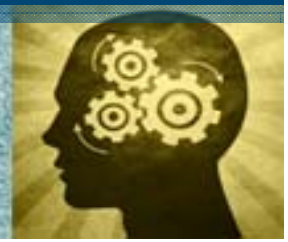
ENTERPRISE SERVICES - REUSABLE, BUSINESS LEVEL BUILDING BLOCKS

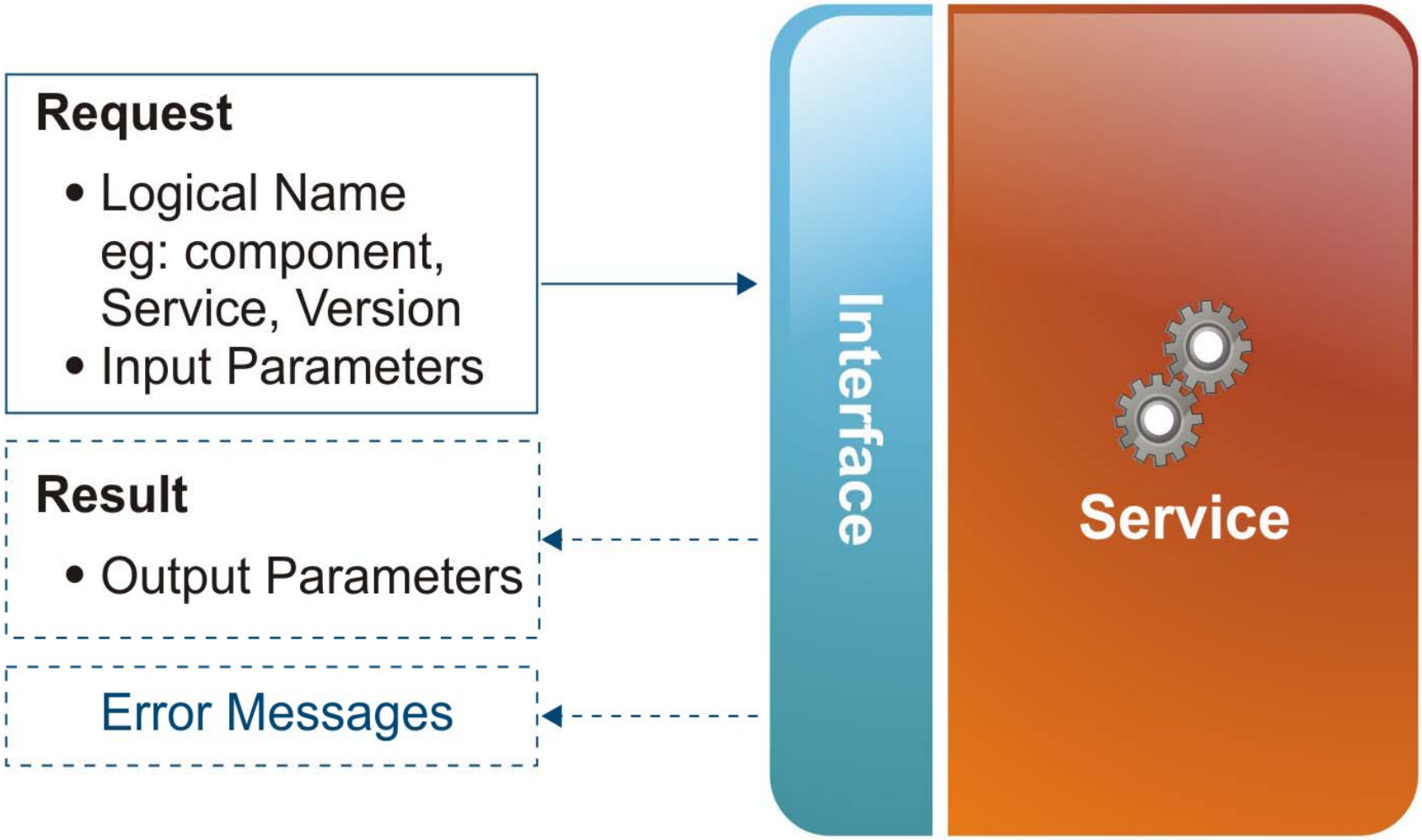


- Coarse-Grained
  - ✓ Level of abstraction easily understood by business people
- Event-Enabled Interfaces
  - ✓ Easily composed into Distributed, Event-driven Processes
- Multi-Language
  - ✓ No development restrictions
- Generalization of Web-Services
  - ✓ Standards-based WSDL interfaces
- REST- based Services
  - ✓ The document *is* the interface
  - ✓ No central state-management; all state carried in messages/documents

STEP 3 – COMPONENTIZING BUSINESS PROCESSES

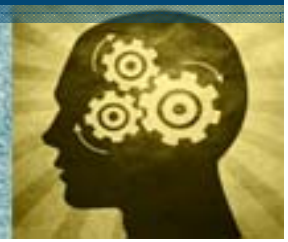
SERVICE → INTERFACE





STEP 3 – COMPONENTIZING BUSINESS PROCESSES

COARSE GRAINED COMPONENTS



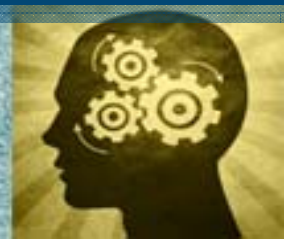
"A component is a non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces."

- Philippe Krutchen, Rational Software

- Well defined interfaces & contracts
- Provides implementation of interfaces
- Coarse grained

STEP 3 – COMPONENTIZING BUSINESS PROCESSES

COARSE GRAINED COMPONENTS



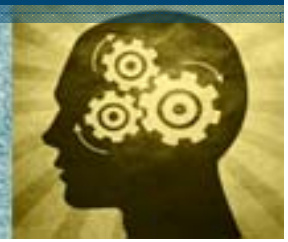
"A software component is a unit of composition with contextually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition."

- Clemens Szyperski, Component Software

- Developed, tested and deployed in complete isolation
- Can be configured for different environments at deployment time
- Supports contextual dependencies

STEP 3 – COMPONENTIZING BUSINESS PROCESSES

COARSE GRAINED COMPONENTS



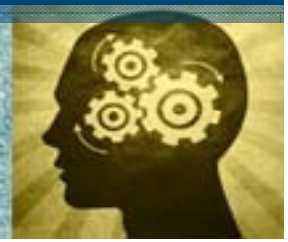
"A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces...typically represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations."

- Grady Booch, Jim Rumbaugh, Ivar Jacobson

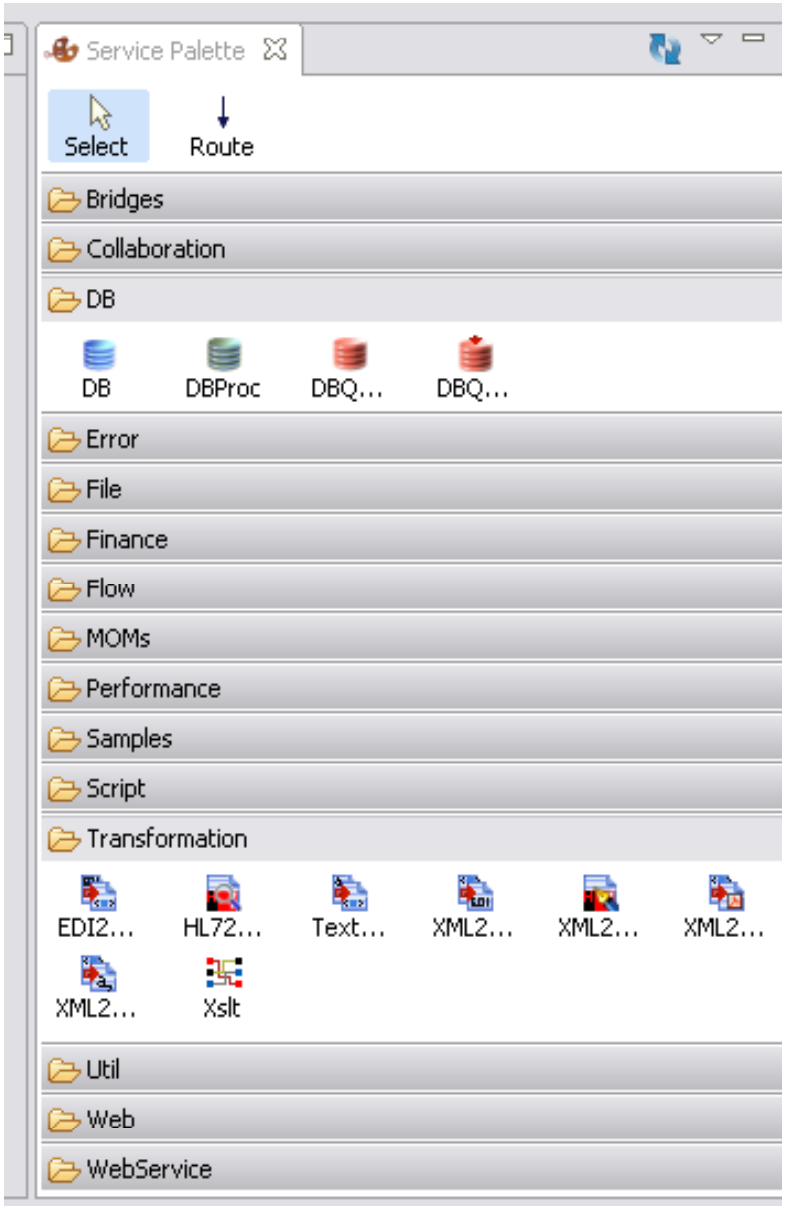
- Includes physical packaging of files, executables, jars, dlls, etc
- Aggregation of classes & functions into a complete, reusable functionality
- Higher level of reusability than Classes

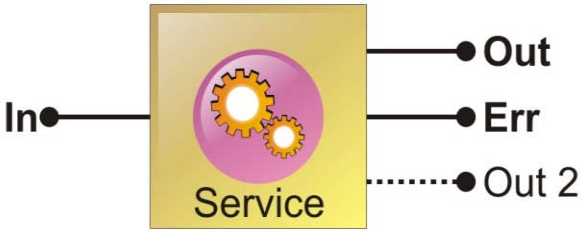
STEP 3 – COMPONENTIZING BUSINESS PROCESSES

PRE-BUILT SERVICES



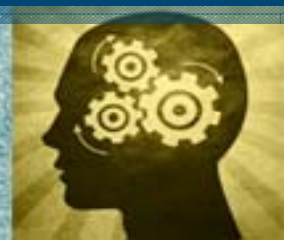
- Services
  - ✓ Bridges
  - ✓ Database/File Adapters
  - ✓ Web/WebService Adapters
  - ✓ Routing (flow services)
  - ✓ Transformation
  - ✓ Signing, encryption, encoding Services
- Adapters
  - ✓ Middleware
  - ✓ Application





STEP 3 – COMPONENTIZING BUSINESS PROCESSES

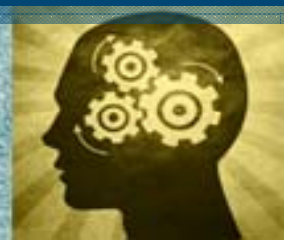
USER DEFINED COMPONENTS – BEST PRACTICES



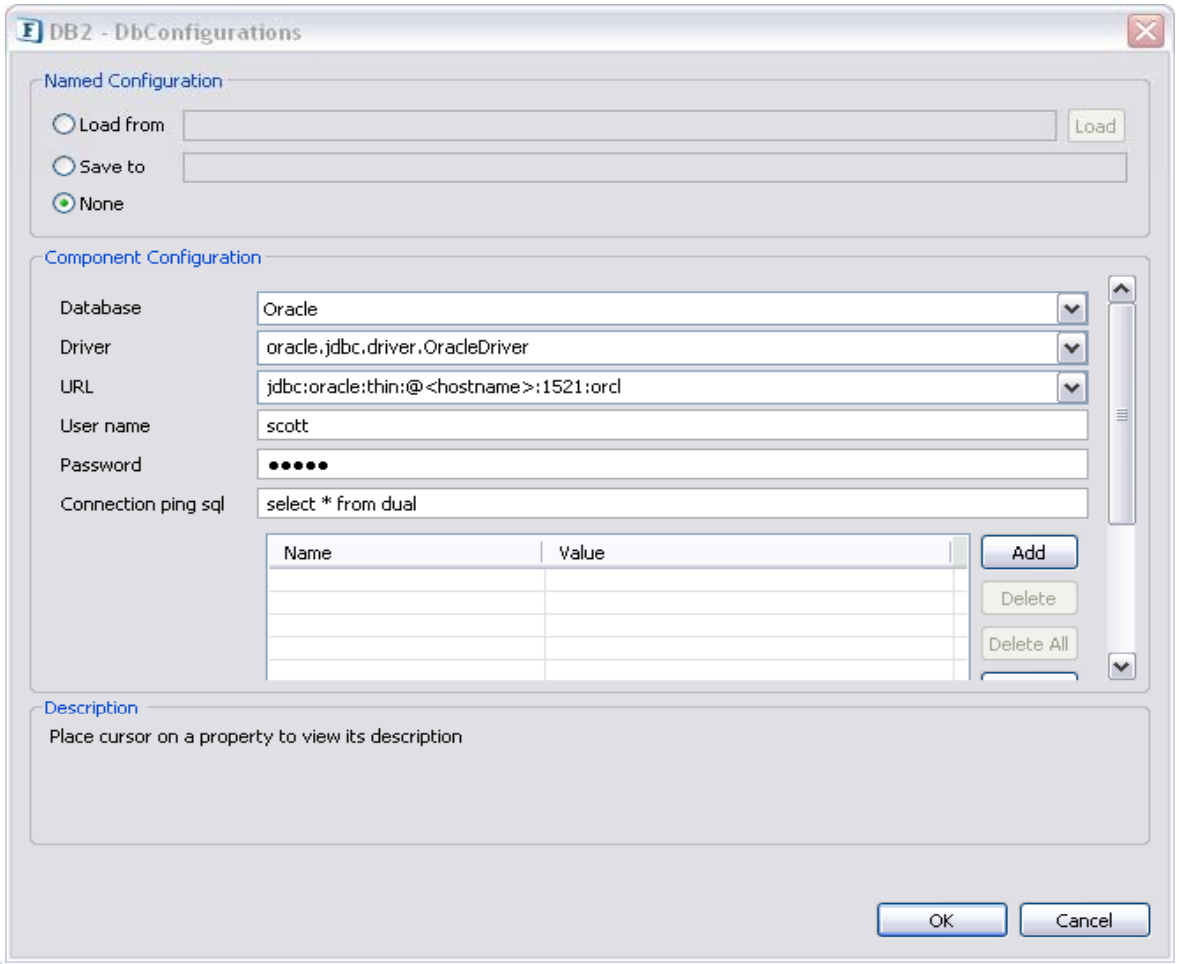
- Identifying set of user defined services
- Choosing Appropriate programming language
  (Java, C, C++, C#, Perl, Python, JavaScript etc)
- Identifying input and output ports
  (dynamic or static)
- Custom Property Sheet
  (Design & runtime properties)
- Exception and Error Handling mechanisms
- Tracing and Logging support
  - Trace Modules and trace levels
  - Logging details
- Transaction support
  - Local
  - Global

STEP 3 – COMPONENTIZING BUSINESS PROCESSES

USER DEFINED – BEST PRACTICES

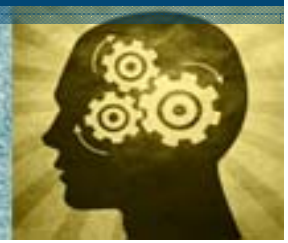


- Each component has specific design and runtime properties that need to be properly configured, without which the integration process will fail
- E.g. Database configuration illustrated below.

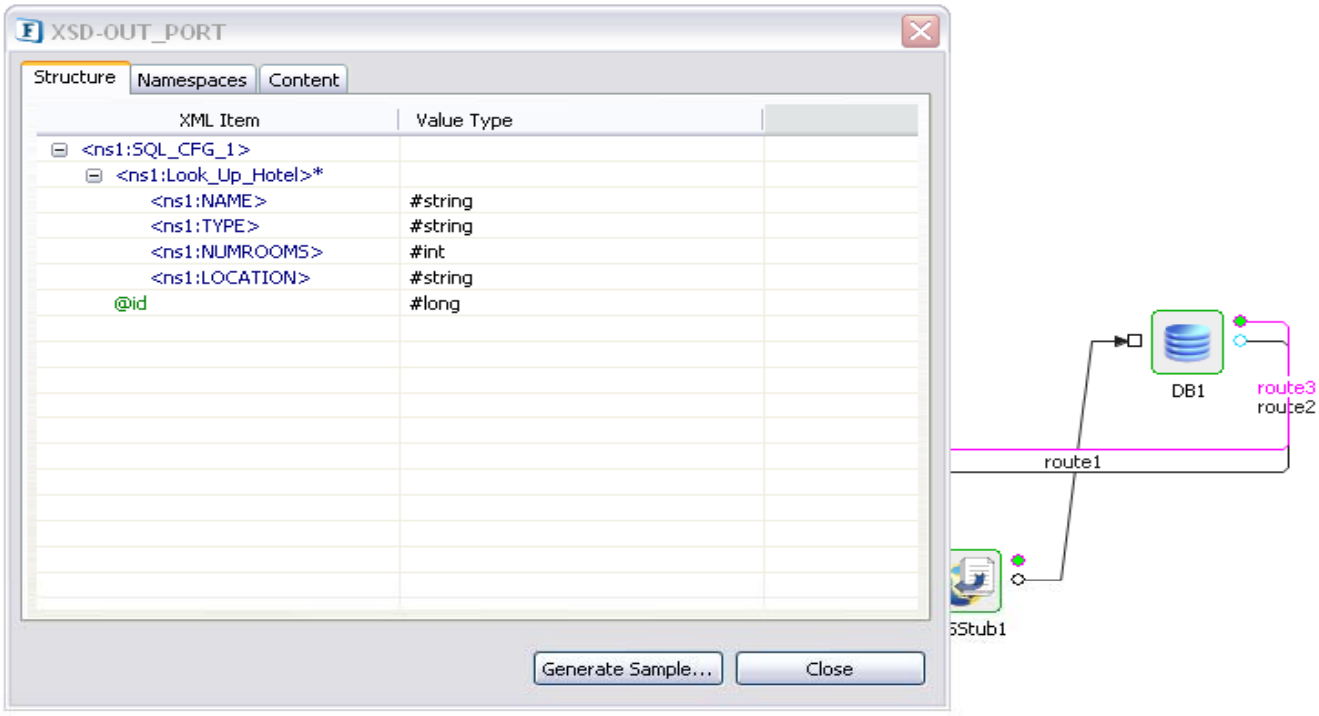


STEP 3 – COMPONENTIZING BUSINESS PROCESSES

USER DEFINED – BEST PRACTICES

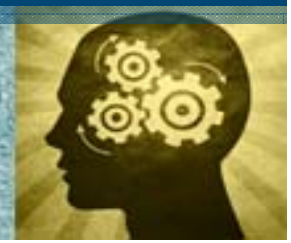


- Identify and configure input and output ports. Based on the configuration of the CPS (Component property sheet), each component generates schemas on both input and output ports.
- The schema defines the data format (XSD, DTD, XML) for input and output messages.

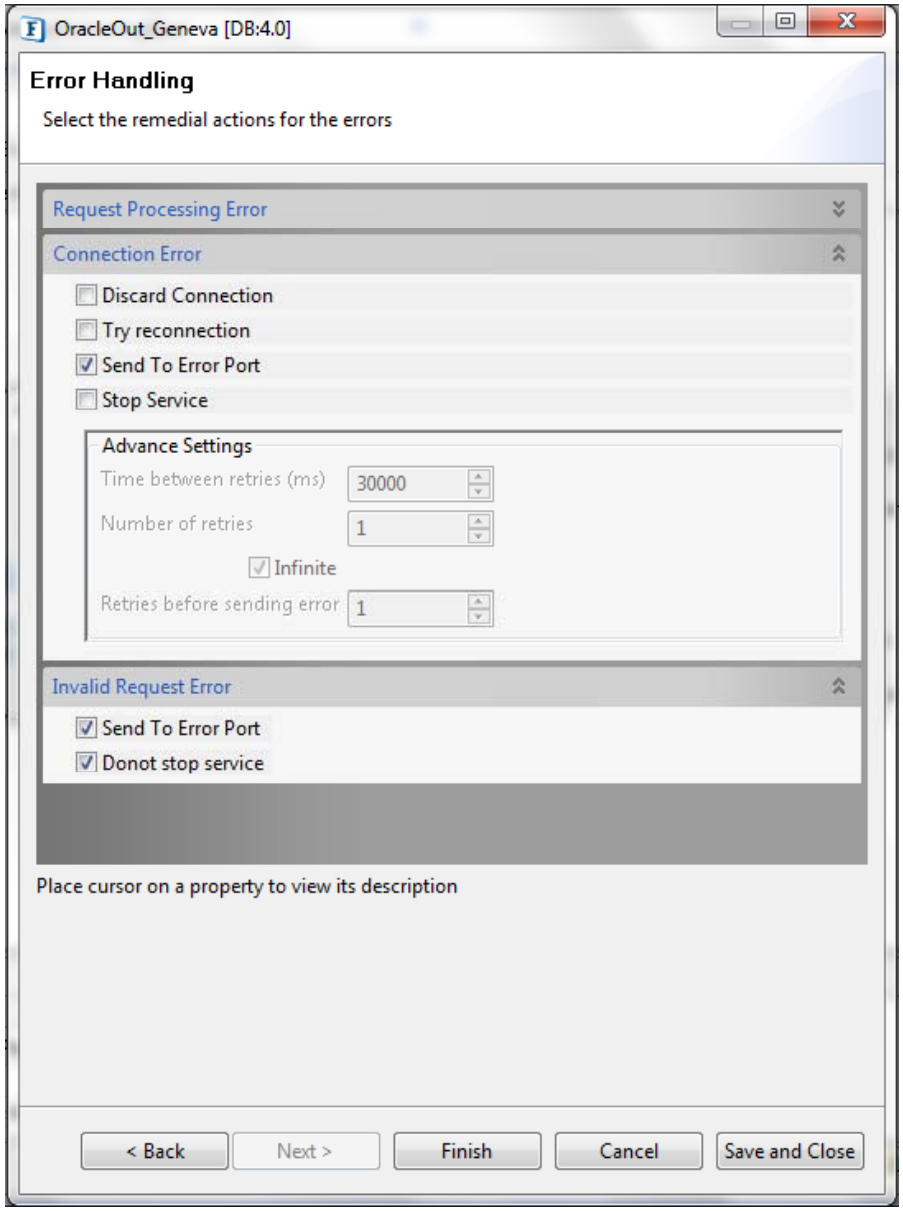


STEP 3 – COMPONENTIZING BUSINESS PROCESSES

USER DEFINED – BEST PRACTICES

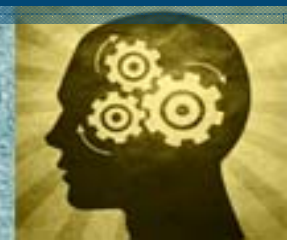


- Ensure the ESB supports error-handling at the component (adapter) level
- Each component should have an error-handling configuration (configured via the CPS) to allow predefined actions to be set for each type of error-condition
- For instance, if you wish to stop message processing when an error occurs, set the “Stop Service” action: this ensures that incoming messages are queued up at the input port; such messages are processed only once the component is restarted

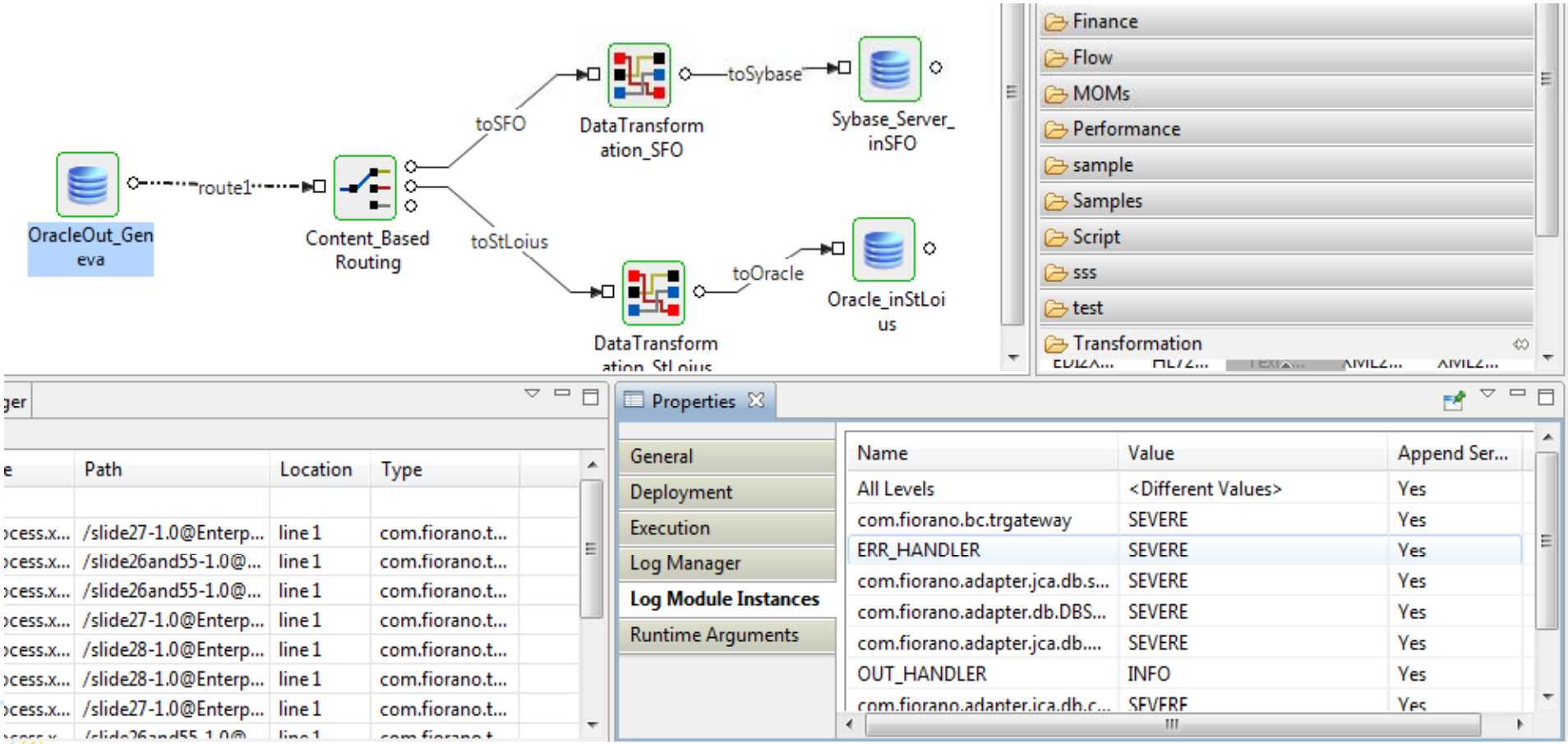


STEP 3 – COMPONENTIZING BUSINESS PROCESSES

USER DEFINED – BEST PRACTICES

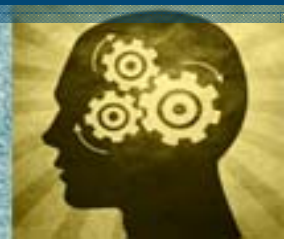


- Tracing and Logging support – essential to debug integration processes in a distributed environment.
  - ❖ Identify modules to set trace/log levels
  - ❖ View/change via Studio or Web-based Dashboard



STEP 3 – COMPONENTIZING BUSINESS PROCESSES

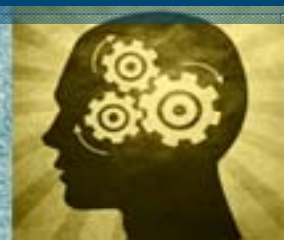
USER DEFINED – BEST PRACTICES



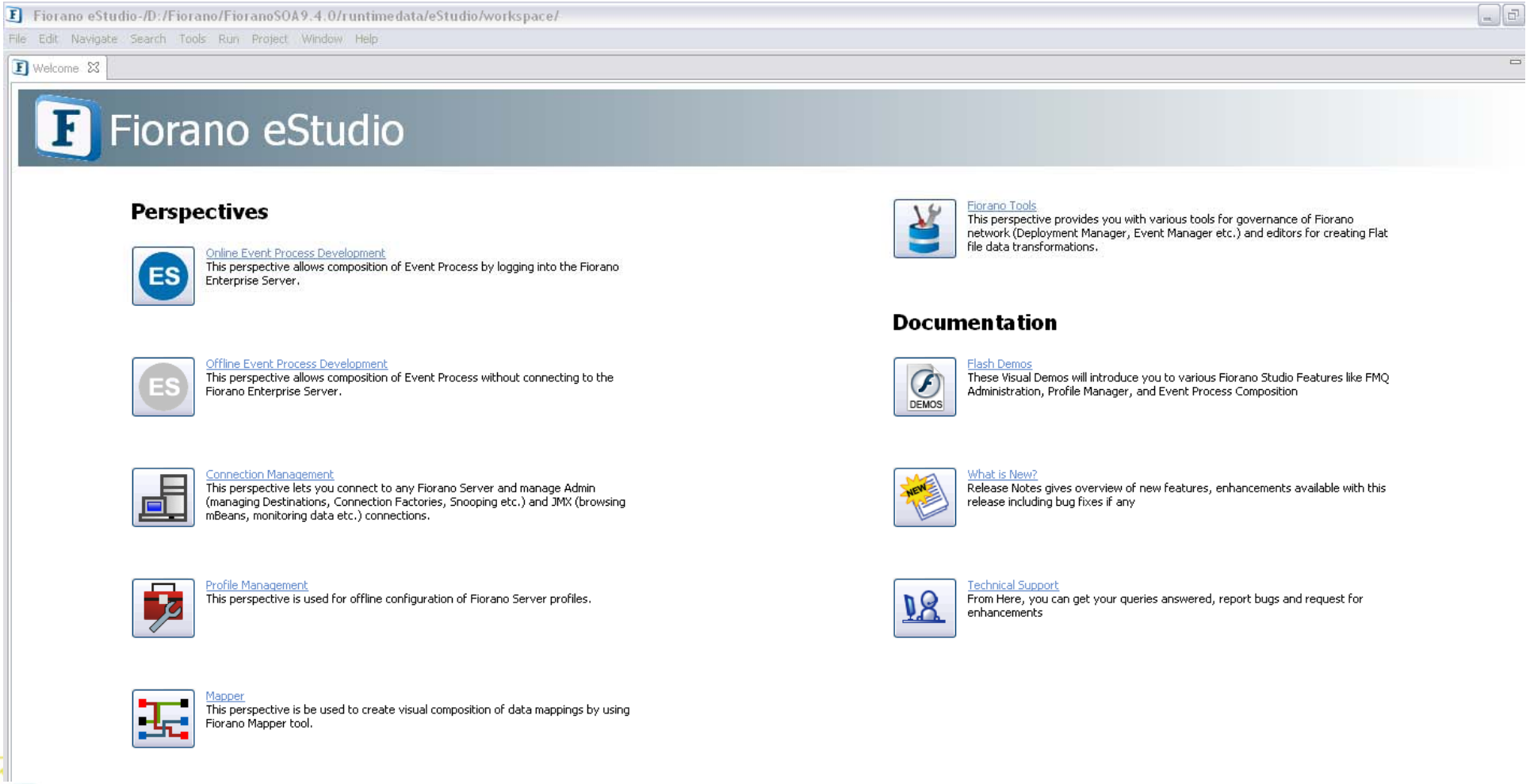
- Transaction support
  - ✓ Local (Client level transactions): typically associated with databases, allowing users to define boundaries across a set of queries and/or input messages
  - ✓ Global (XA): must be handled at the component level
  - ✓ Compensating transactions: mapped onto an ESB flow; on failure of a certain operation, a compensating transaction undoes the previous changes
- Performance Requirements
  - ✓ Expected transaction throughput: can be done by running flows in parallel on different peer servers and load balancing incoming messages across these peers
  - ✓ Multi-threading support: supported at the component level by increasing the number of JMS Sessions on the input port of a component
- Synchronous/Asynchronous; the ESB must handle both synchronous and asynchronous end-points, both for inbound and outbound requests (JMSIn, JMSOut, WSStub, and WSConsumer)

STEP 3 – COMPONENTIZING BUSINESS PROCESSES

USER DEFINED – BEST PRACTICES

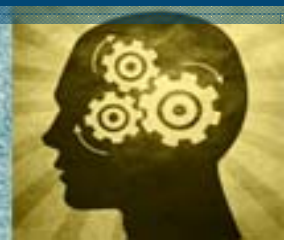


- Tools for service development – check for “online” (connected to the server) and “offline” (disconnected mode) development support
- Ideally, the ESB will support IDE plug-ins



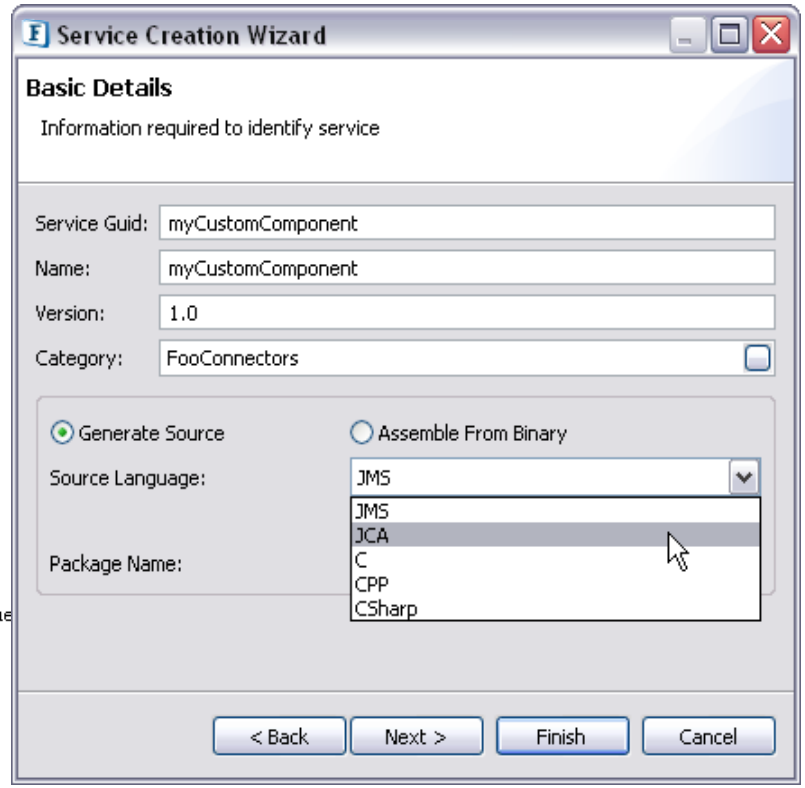
STEP 3 – COMPONENTIZING BUSINESS PROCESSES

REUSING EXISTING SERVICES – BEST PRACTICES

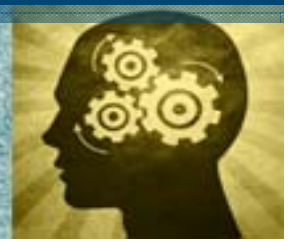


- The ESB must allow custom-component development in multiple languages (Java, C, C++, .NET, scripts, etc), together with:
- Adapters to Web-Services, REST, COM, EJB, CORBA, RMI, etc
- Reuse of business processes

```
/**
 * creates an instance of Request Processor.
 * @param schema If there's any schema on the input port to be validated.
 * @param logger logger used for logging.
 * @param serviceConfiguration configuration object.
 */
public RequestProcessor(ESBRecordDefinition schema, Logger logger,
                        IServiceConfiguration serviceConfiguration) {
    super(schema, logger, serviceConfiguration);
    this.logger = logger;

}

/**
 * Processes the input request. By default it returns the input message.
 * @param request request string
 * @return response message.
 * @throws ServiceExecutionException if there is any exception in processing the request
 */
public String process(String request) throws ServiceExecutionException {
    logger.log(Level.INFO, RBUtil.getMessage(Bundle.class, Bundle.REQUEST_PROCESSED, new Object[]{reque
    // business logic goes here
    return request;
}

/**
 * Processes the input request. By default it returns the input object.
 * @param request request object
 * @return response object.
 * @throws ServiceExecutionException if there is any exception in processing the request.
 */
public Object process(Object request) throws ServiceExecutionException {
    logger.log(Level.INFO, RBUtil.getMessage(Bundle.class, Bundle.REQUEST_PROCESSED, new Object[]{request}));
    return request;
}
```



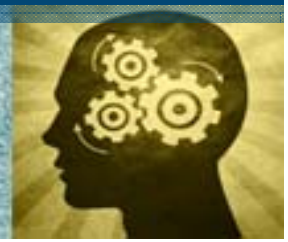
STEP 4 – DEFINING DISTRIBUTED ESB PROCESSES



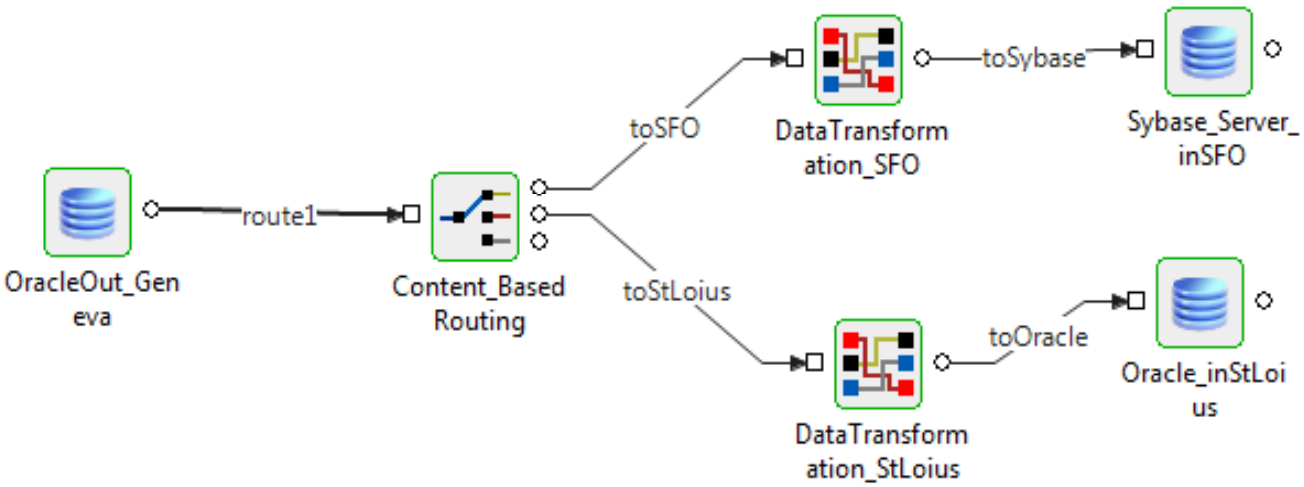
- Control vs. Data Flow
  - Data flow incorporates both synchronous and asynchronous message-flow; more general purpose than pure control flow
- Business Process Orchestration
  - Typically implies a central orchestration engine
  - For distributed ESBs, central orchestration is a bottleneck; Choreographed Event-flow processes with a graphical representations that map directly to the physical implementation offer greater flexibility and generality
- Event Warehousing
- Security
- Logging, Tracing & Alerts
- Data format impedance mismatch
- Configuring Business processes for failover
- Configuring Business processes for performance

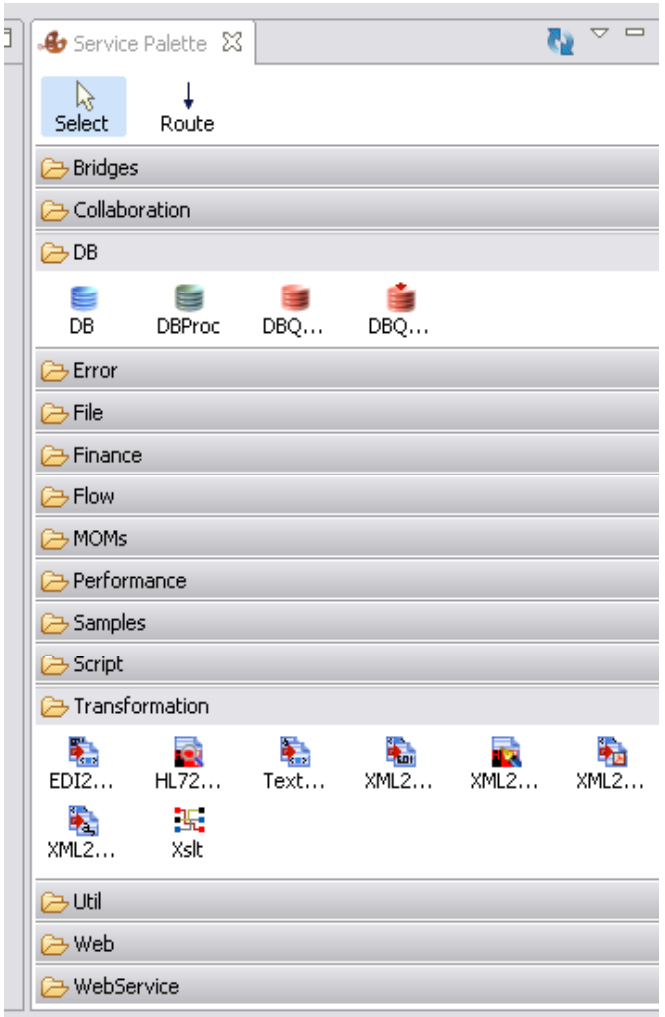
STEP 4 – DEFINING DISTRIBUTED ESB PROCESS

EVENT PROCESS



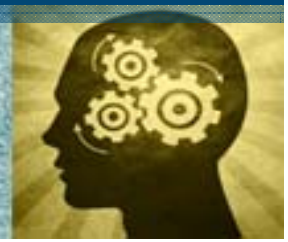
- Event Driven Business Process or simply “Event Process”
- Drag and drop approach





STEP 4 – DEFINING DISTRIBUTED ESB PROCESSES

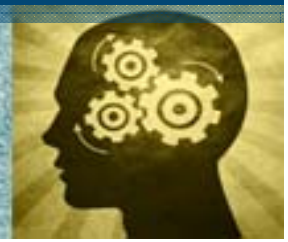
BUSINESS PROCESS CHOREOGRAPHY



- Composition using pre-built services
- Data routing
- Control information
- Data transformation
- Identifying Node Names
- Configuring Service Design time properties

STEP 4 – DEFINING DISTRIBUTED ESB PROCESSES

DOCUMENT TRACKING



- Document tracking is a critical requirement essential to audit the incoming and outgoing data flowing through each component; message warehousing.

lhost:1980/ESBDashboard/index.html#Application%20Documents

Fiorano SOA 9.4.0, Build No. 9550, OS: Windows 7 32-bit, JVM: 32-bit,

English Home Logou

SBW WorkFlows Count

Application Name	In Transit	Completed	Exception Documents
CTMGGATEWAYSTATICDATA	0	0	0
CTMMOCKUP	0	0	0
DATABASE_REPLICATION	0	0	0
DATABASE_UPDATE	0	0	0
DB_DEMO	1	1	2
DB_SAM			
DB_SAMPLE			
DB_SHOW			

Page 2 of 7 | Refresh SBW

DB_DEMO ▸ fps_1315494768488_1 ▸

SBW Documents

Document ID	Component
DB1_DB_DEMO_1315494768488_1	DB1

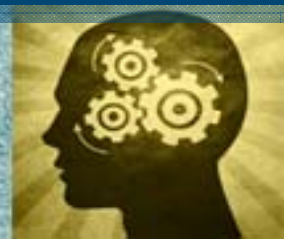
Document Details for DB1_DB_DEMO_1315494768488_1

Properties | Attachments | General | Message Body | Application Context | Carry Forward Properties | Source Contexts

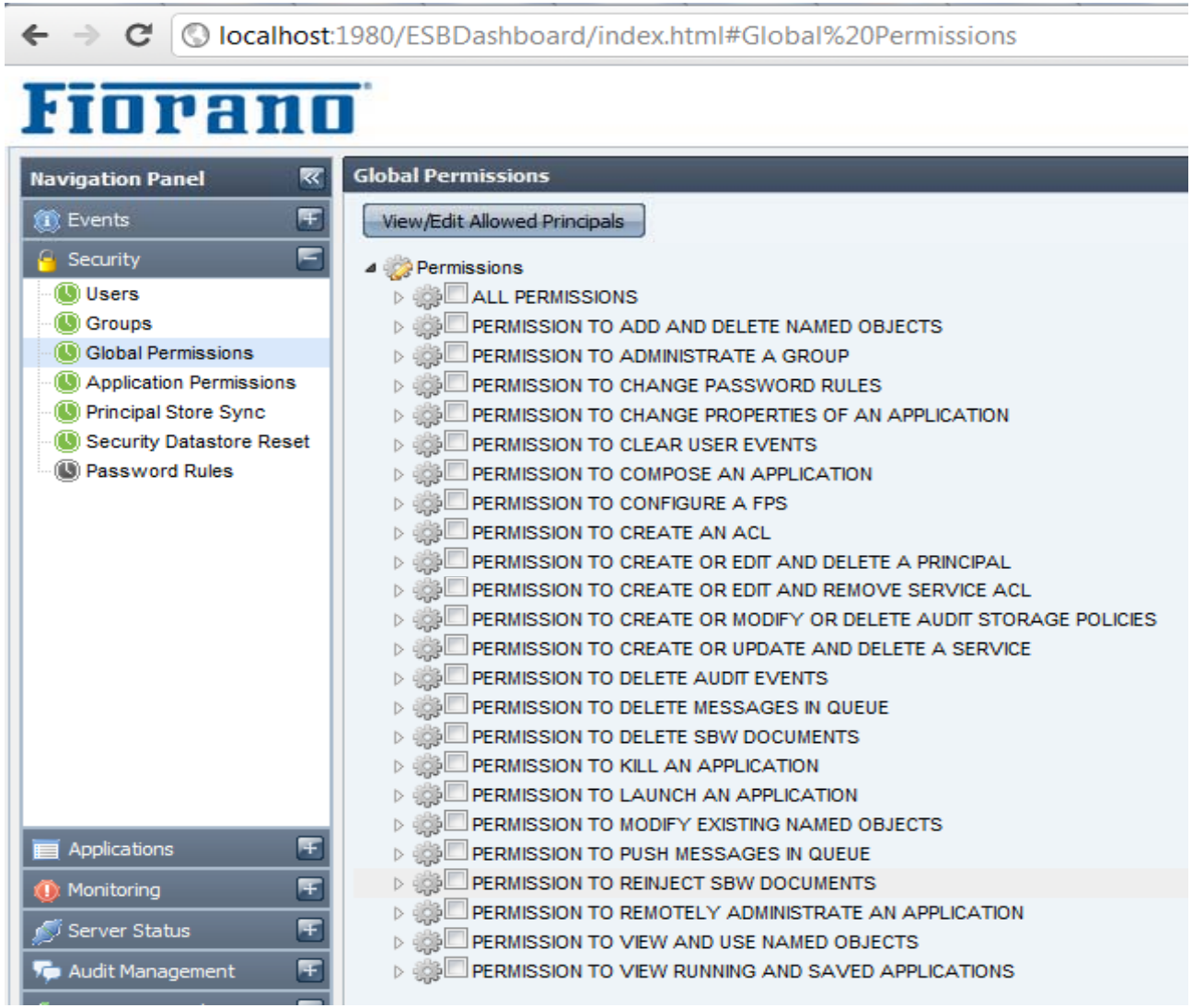
Name	Value
ESBX__SYSTEM__WORK_FLOW_INST_ID	fps_1315494768488_1
ESBX__SYSTEM__WORKFLOW_STATUS	EXECUTING
ESBX__SYSTEM__USER_DEFINED_DOC_ID	
ESBX__SYSTEM__TOTAL_TIME	0
ESBX__SYSTEM__STATE_ID	ON_EXCEPTION
ESBX__SYSTEM__SOURCE_DESTINATION_NAME	DB_DEMO__DB1__ON_EXCEPTION
ESBX__SYSTEM__SOURCE	fps
ESBX__SYSTEM__SINK	FES
ESBX__SYSTEM__PORT_NAME	ON_EXCEPTION
ESBX__SYSTEM__OUT_TIME	1315495273555
ESBX__SYSTEM__IN_TIME	1315495273555
ESBX__SYSTEM__EXECUTING_INCR	1
ESBX__SYSTEM__EVENT_TYPE	1
ESBX__SYSTEM__EVENT_SCOPE	Tifosi

STEP 4 – DEFINING DISTRIBUTED ESB PROCESSES

ROLE BASED SECURITY

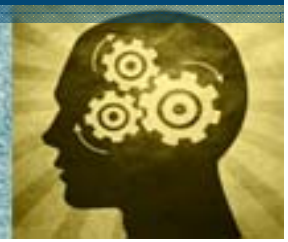


- The ESB must support ACL-based security. Users can be assigned different roles and each role has different permissions

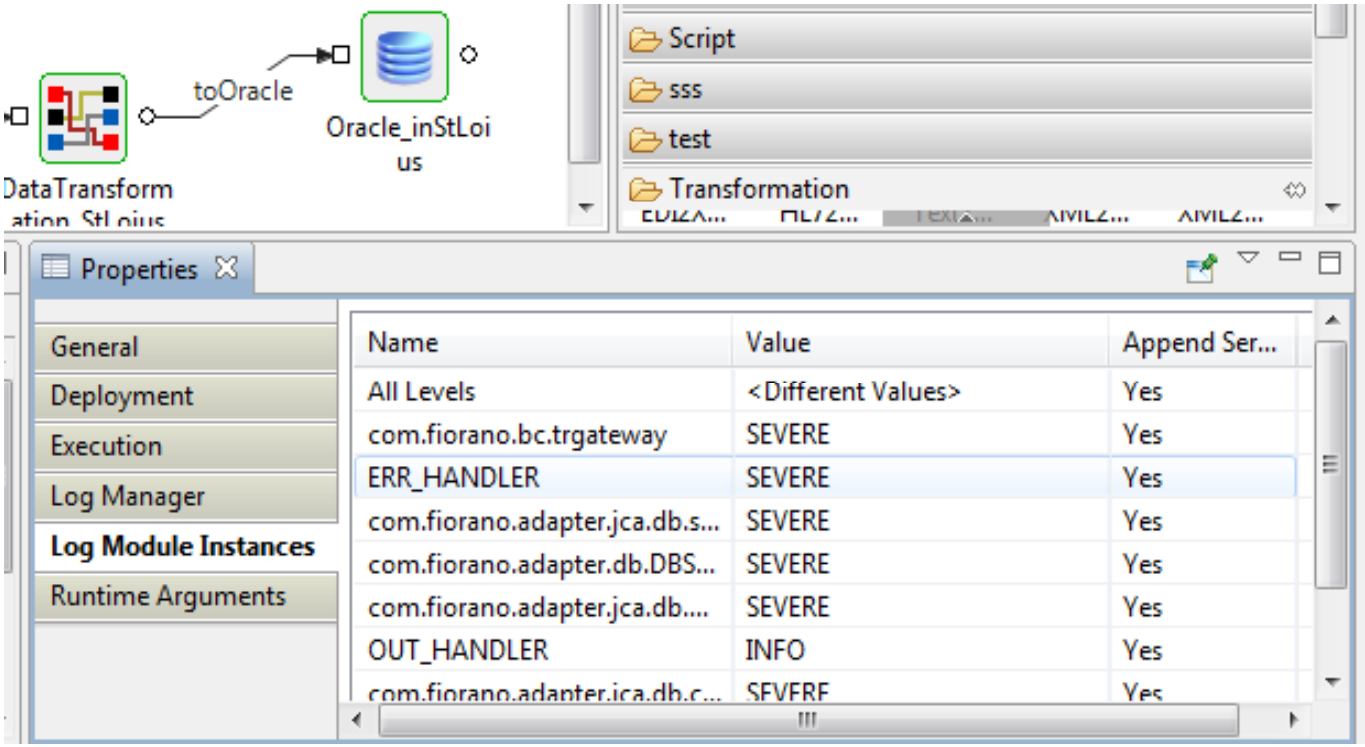


STEP 4 – DEFINING DISTRIBUTED ESB PROCESSES

LOGGING, TRACING & ALERTS

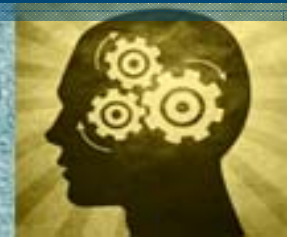


- To facilitate debugging in a distributed environment, the ESB should ideally allow alerts to be set depending on trace and log levels

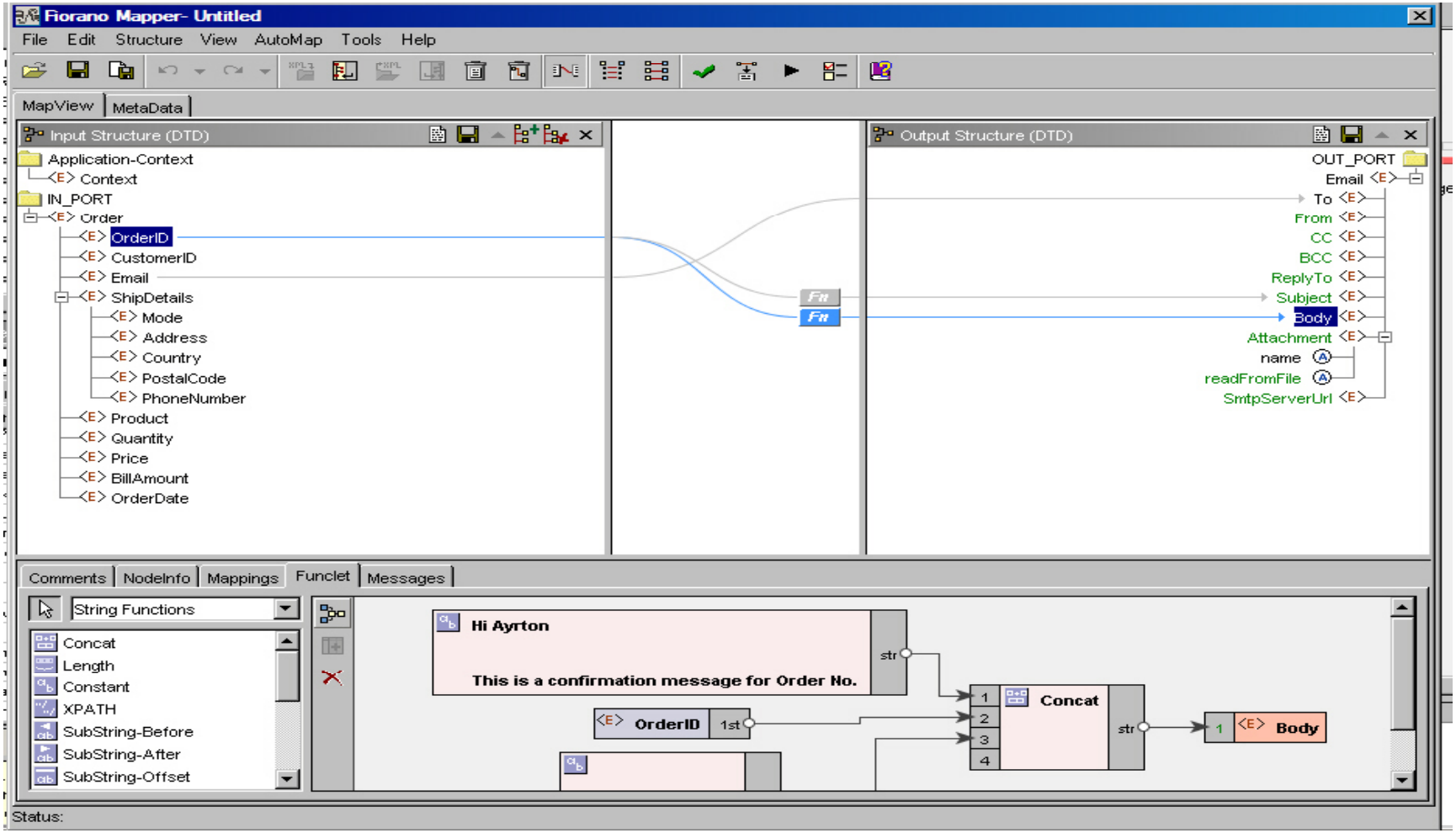


STEP 4 – DEFINING DISTRIBUTED ESB PROCESSES

DATA FORMAT IMPEDANCE MISMATCH

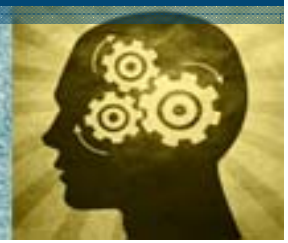


- Transformation is fundamental to ESB processes; powerful XSLT visual transformation tools are mandatory; manual XSLT injection is a strong plus

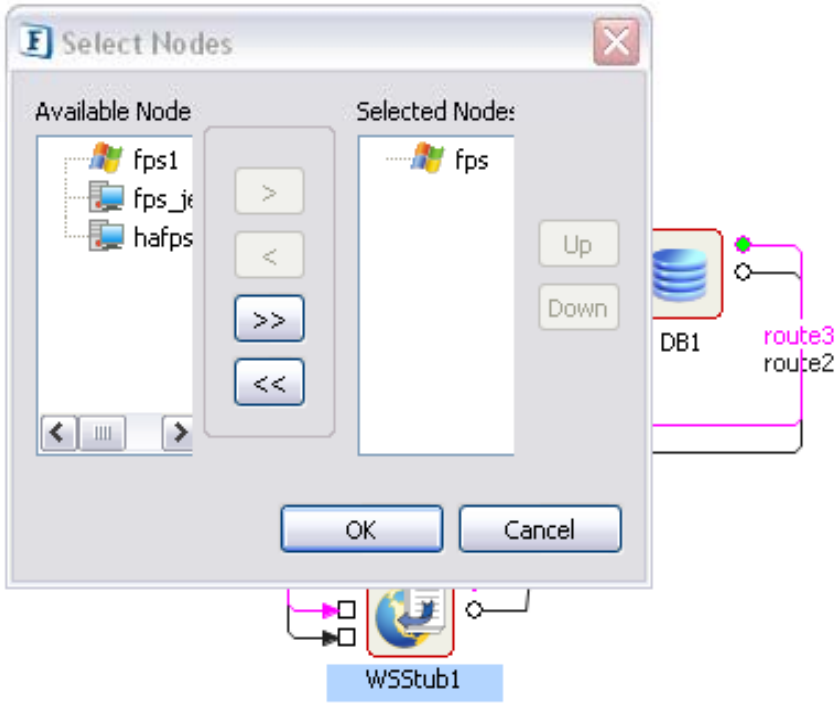


STEP 4 – DEFINING DISTRIBUTED ESB PROCESSES

CONFIGURING BUSINESS PROCESSES FOR FAILOVER

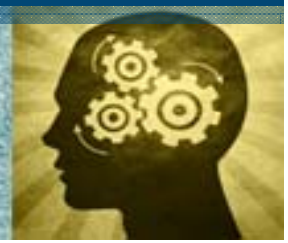


- State failover for services: if a Service/Component fails on a given machine, it should be relaunched on another (failover) machine
- Server level failover: if the ESB server fails, a pre-configured backup should seamlessly take over



STEP 4 – DEFINING DISTRIBUTED ESB PROCESSES

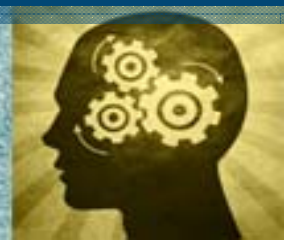
CONFIGURING BUSINESS PROCESSES FOR PERFORMANCE



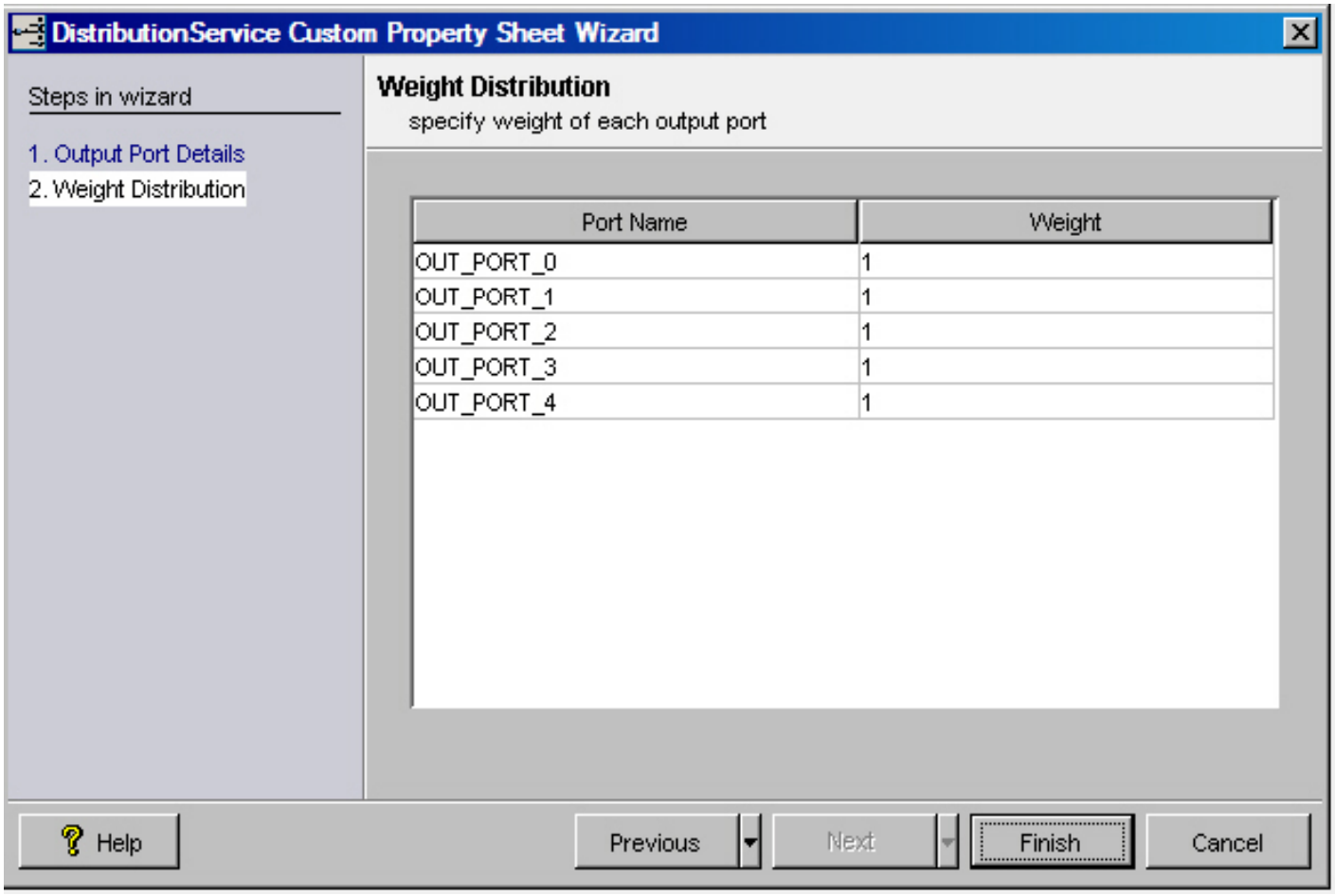
- Identifying Parallel data flows
- Dynamic rerouting of data based on load
- Identifying Heavy-weight services (80/20 Rule)
- Running multiple instances of “heavy weight” services on different nodes
- Sub-Flows and Sub-Processes for effective business process execution
- Log/Trace level optimization
- Event tracing optimization

STEP 4 – DEFINING DISTRIBUTED ESB PROCESS

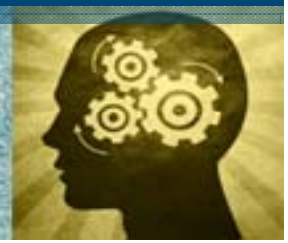
CONFIGURING BUSINESS PROCESSES FOR PERFORMANCE



- Load-balancing should be possible at the Service/Component level
- E.g. based on a preconfigured weightage, incoming messages are distributed to the output ports of a distribution component



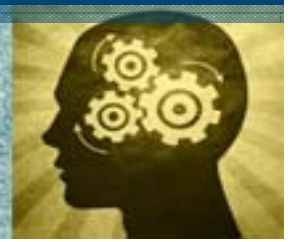
STEP 5 – DEPLOYING ESB PROCESSES



- Deployment - what does it mean ?
- Identifying Network Domain/Topology
- Manual Services vs. Auto-Launched Services
- Security issues for Deployment
- Service Development Languages and Platforms

STEP 5 – DEPLOYING ESB PROCESSES

WHAT DOES IT MEAN?



- Deployment Descriptors – are typically used to hold execution parameters and deployment information of components, including dependent services, resources, etc. The ESB uses such descriptors to deploy components at runtime.

EAI Demo | ServiceDescriptor.xml

Deployment

General Information

Label:

Production | QA | Development | Staging

Supported OS:

Linux | Windows | Macintosh | Solaris

All OS

AutoInstallable

Resource

Name	For Cor
fesb-ra.xml	true
ra.xml	false
fbc-comp-DB.jar	true
estudio-db.jar	true
dbconnection32.png	false
dbconnection.png	false
version.properties	false

Add... | Remove | Up | Down

Service Dependencies

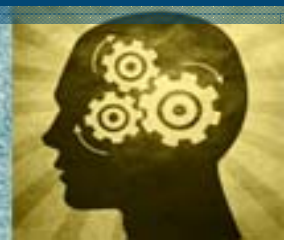
Dependency Name
base64
log4j
jdbc
dmlparser
BCGateway
xalan

Add/Remove... | Up | Down

Overview | Execution | Deployment

STEP 5 – DEPLOYING ESB PROCESSES

MANUAL VS AUTO LAUNCH

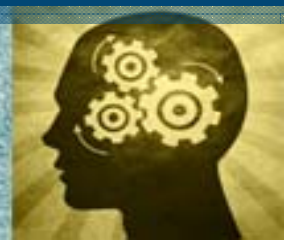


Manual Services

- Executed externally to the ESB (Servlets, EJBs etc)
- Combination of managed and unmanaged components
- Managed Components like:
  - ✓ A Webservice deployed in a Webservice container
  - ✓ An EJB deployed in J2EE container
  - ✓ A COM Object deployed in COM+ server
  - ✓ A CORBA based server Object deployed in an ORB
  - ✓ Windows Service
- Unmanaged Components like:
  - ✓ Java executable archive
  - ✓ A C/C++ executable
  - ✓ Legacy Application running in a mainframe environment
  - ✓ A Unix shell program (functioning within a pipe-and-filter style architecture)

STEP 5 – DEPLOYING ESB PROCESSES

MANUAL VS AUTO LAUNCH

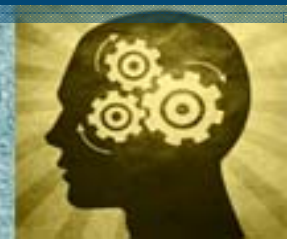


Auto Launched Services

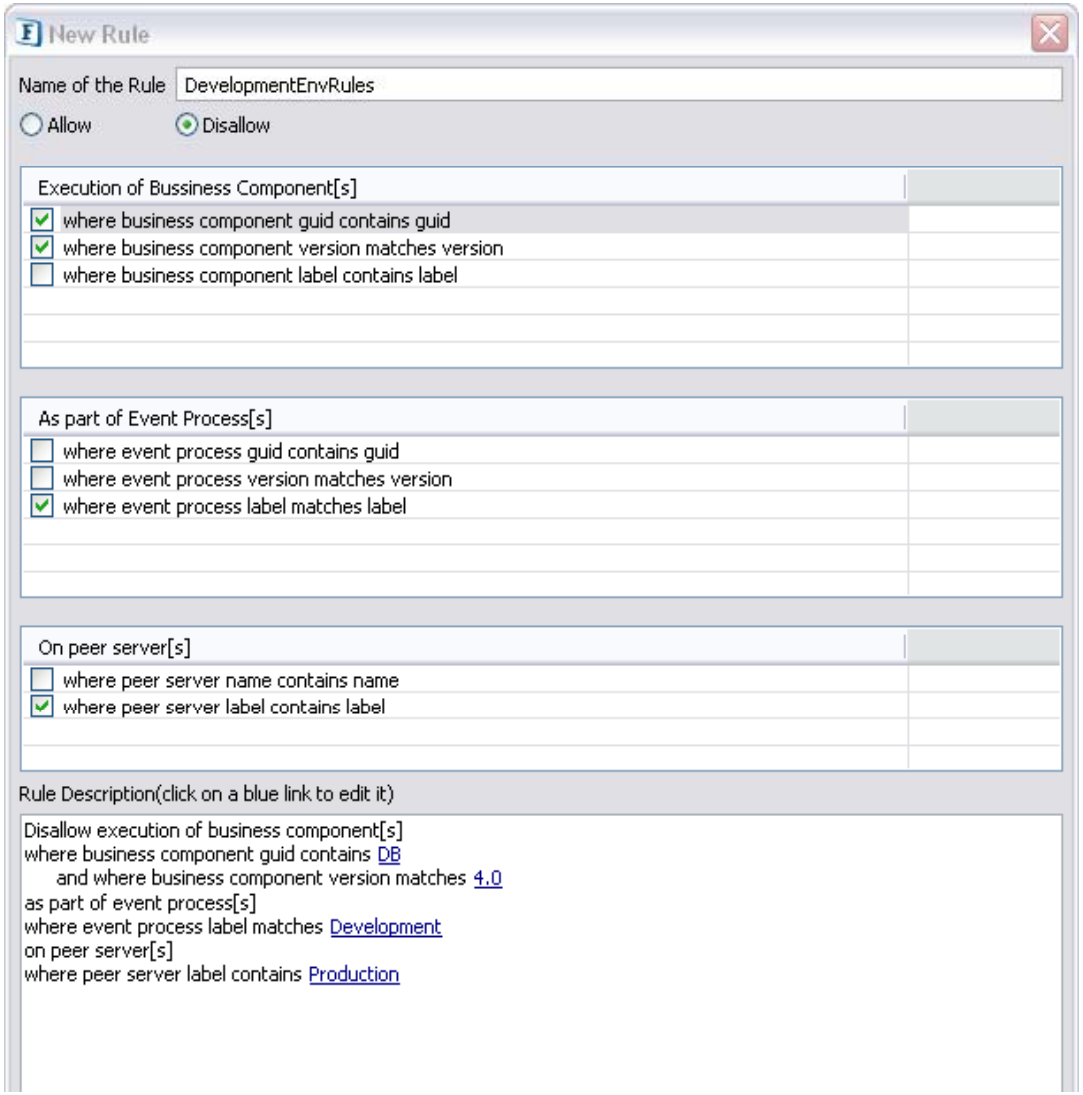
- Native ESB services: managed and launched by ESB containers
  - Auto start/stop and restart of these services
  - Connectivity management
  - Fine grained monitoring

STEP 5 – DEPLOYING ESB PROCESSES

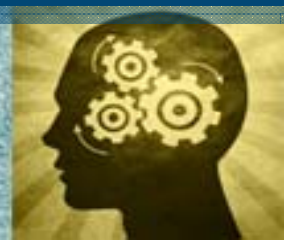
SECURITY ISSUES



- Ideally, the ESB should allow flexible Deployment Manager to control and manage service deployment
- For instance, the ESB should enable users to allow/restrict the execution of components that belong to certain integration processes on certain machines across the network
- Useful for fine-grained control of the integration process as well as for Security

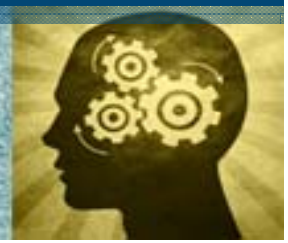


STEP 6 – LAUNCHING & MONITORING ESB PROCESSES



- Remote Launch of Business Process
- Monitoring state of Application and all its associated service instances
- Runtime hooks to determine state of service
- Real-time data debugging
- Business Process Monitoring
  - ✓ SNMP, JMX etc
  - ✓ Monitoring of Servers

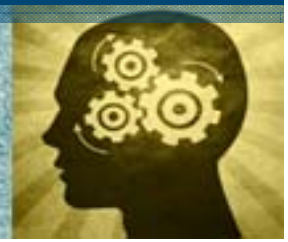
STEP 6 – LAUNCHING & MONITORING ESB PROCESSES



Launch of ESB processes

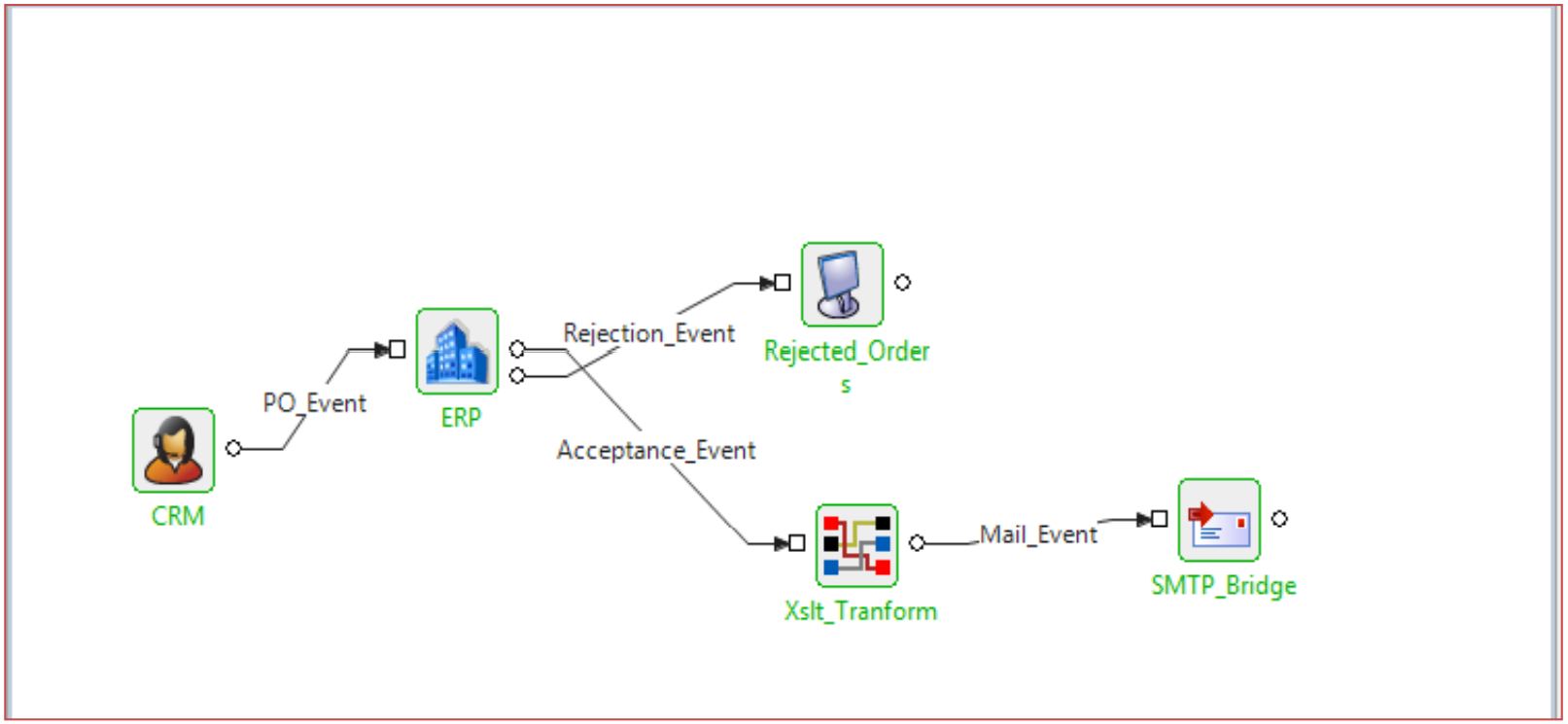
- Remote Launch of ESB Services
- Service Instantiation
  - ✓ On-demand/On-Event Instantiation
  - ✓ Auto Instantiation
- Re-Launch of Business Process
  - ✓ Auto re-launch on different set of nodes
  - ✓ Rules based re-launch of business process

STEP 6 – LAUNCHING & MONITORING ESB PROCESSES

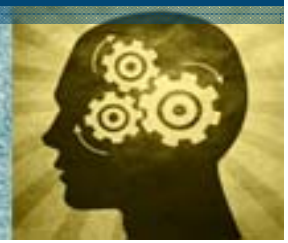


Monitoring the state of an Integration flow and its associated service instances

- ✓ Local Queue (message) Monitoring and Management
- ✓ Identify performance/scalability bottlenecks in real time by checking the number of backed up messages on input queues
- ✓ Identify parallel flows in the Integration

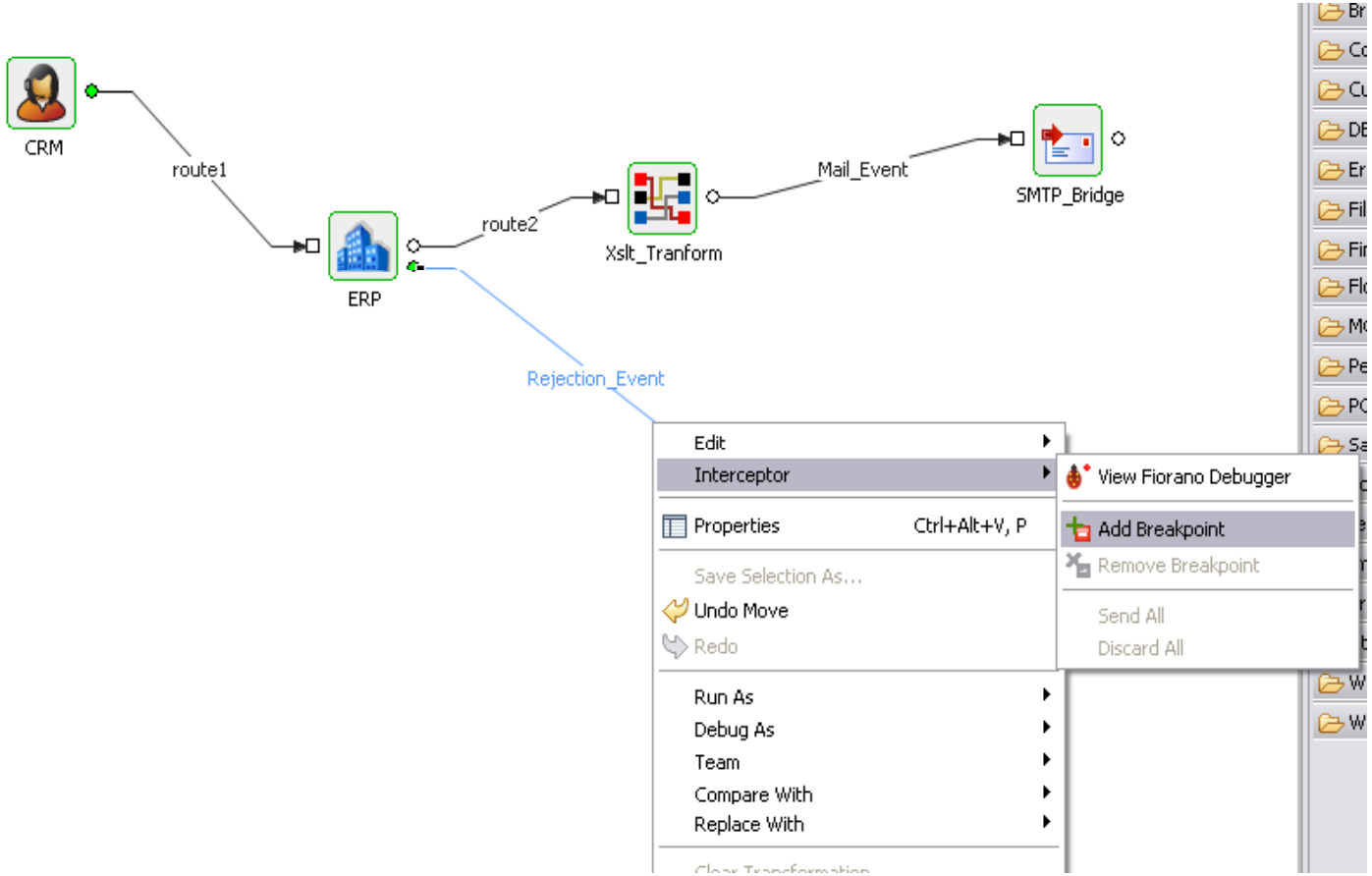


STEP 6 – LAUNCHING & MONITORING ESB PROCESSES

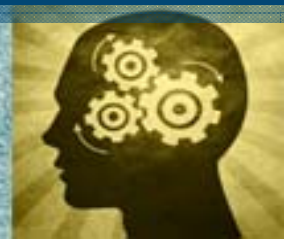


Ensure the ESB supports runtime hooks to determine the state of a particular service

- ✓ Runtime tracing and Logging
- ✓ Sub-Flows to handle Error conditions
- ✓ Alert Handlers

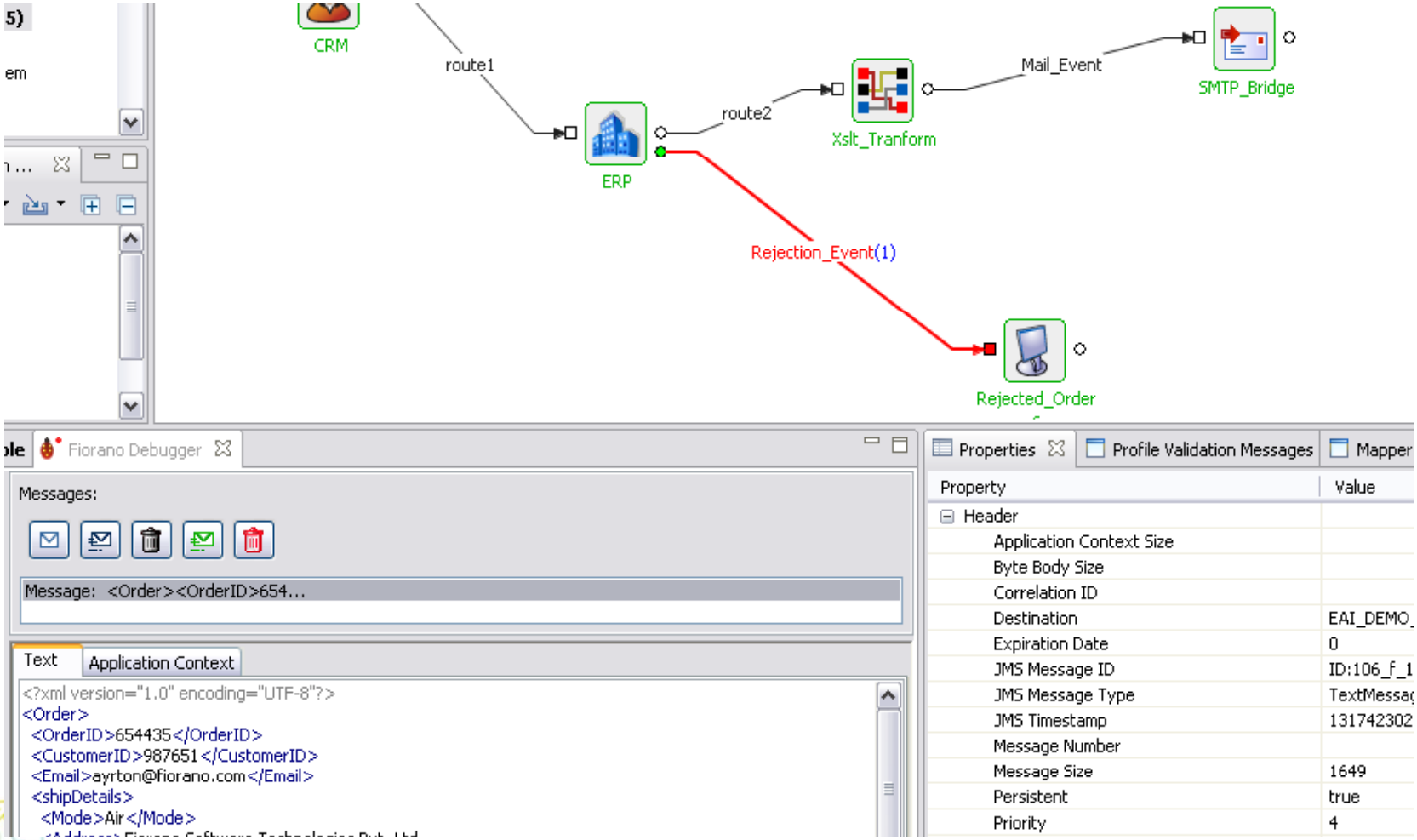


STEP 6 – LAUNCHING & MONITORING ESB PROCESSES

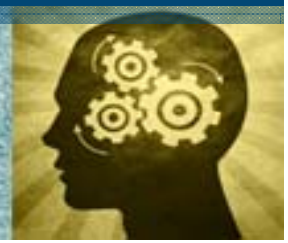


Real-time Message Interception

- breakpoints can be set to view content of ‘in-flight’ messages (payload, properties)
- facilitates debugging of distributed integration processes



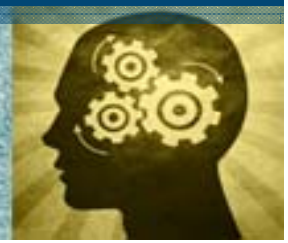
STEP 6 – LAUNCHING & MONITORING ESB PROCESSES



Business Process Monitoring

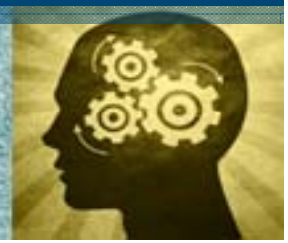
- SNMP, JMX etc
- Monitoring of Servers
  - Heap usage
  - Backlog monitoring
  - Performance monitoring

STEP 6 – CHANGE MANAGEMENT & VERSIONING



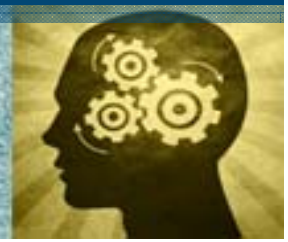
- Dynamic Extensions to Business processes
  - ✓ Static vs. Dynamic
  - ✓ Impact Analysis
  - ✓ Extend Business process to include new services
  - ✓ Extend Business process to include data services with different data formats
  - ✓ Optimize Business process for performance, scalability
  - ✓ Extend Business processes to handle network configuration changes
- Service and Application Versioning

STEP 6 – CHANGE MANAGEMENT & VERSIONING



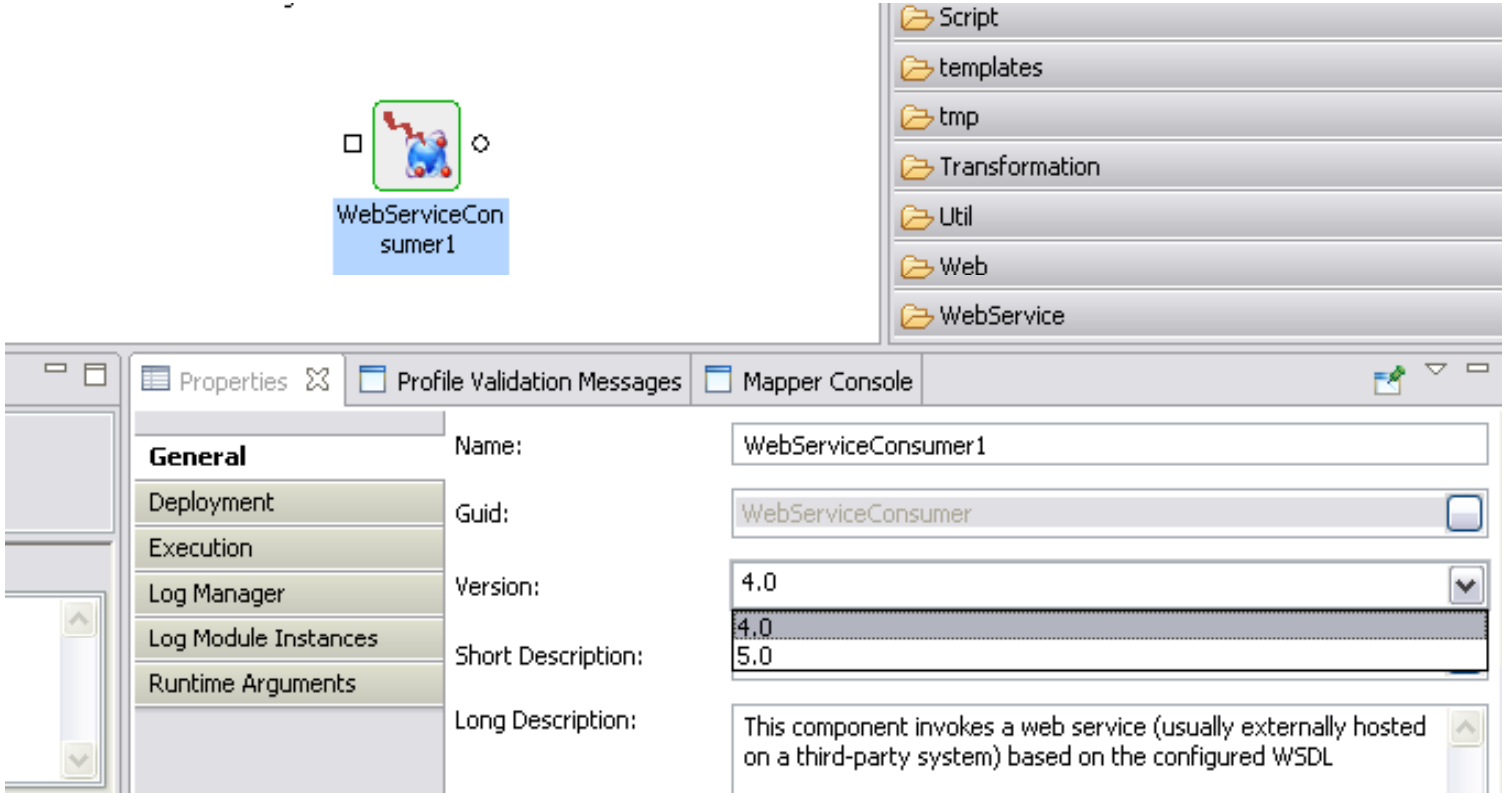
- Configuration Management
  - ✓ Moving from one stage to another (QA to Deployment)
  - ✓ Moving from one environment to another (customer to internal testing)
- Extension to Business process
  - ✓ Updating Data Consistency Application to include a different data center
- Change to Business Process
  - ✓ Flexibility to adapt to new technologies (WebService instead of a C++ stand alone Application)

STEP 6 – CHANGE MANAGEMENT & VERSIONING

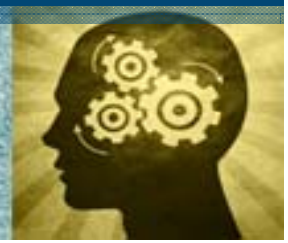


Service and Application Versioning

- Manage and maintain multiple versions of services along with Labels
- Quickly allow migration from one service version to another

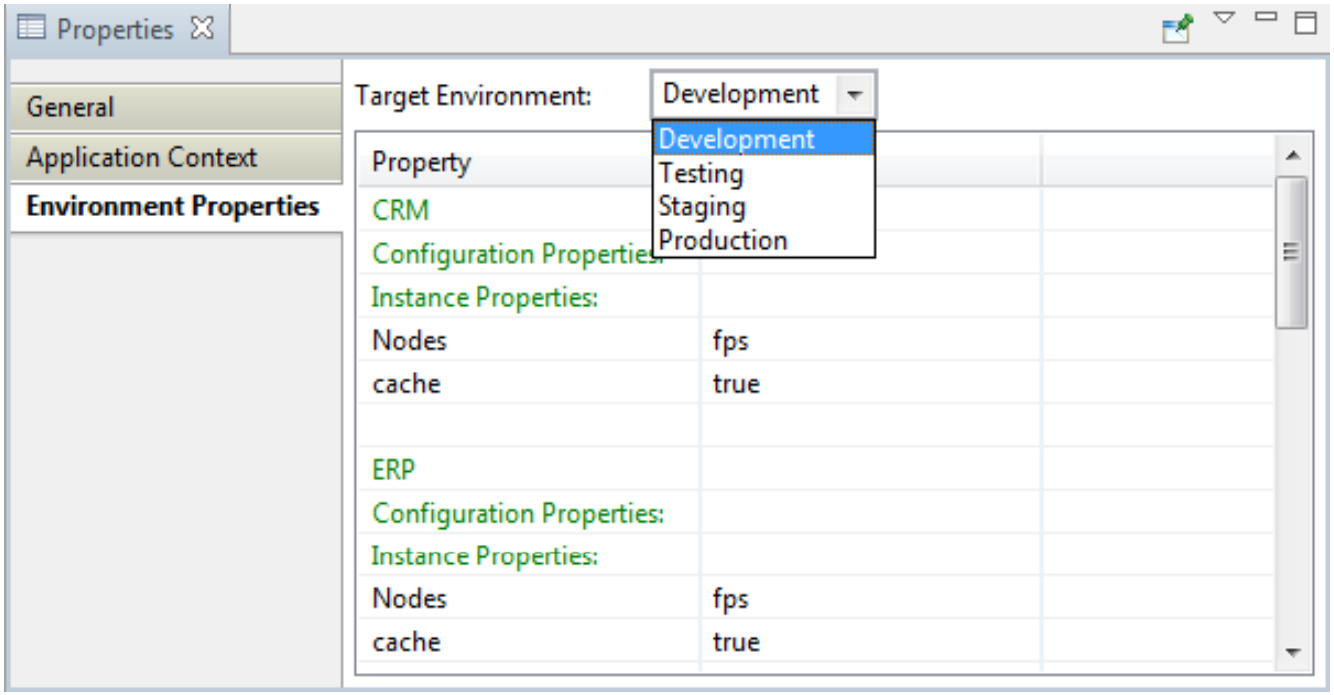


STEP 6 – CHANGE MANAGEMENT & VERSIONING



Configuration Management

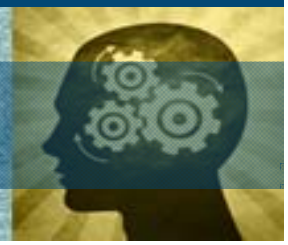
- Moving from one stage to another (e.g. Development to QA, QA to Staging, etc.)
- Point and click-operation: the user should be allowed choose the target environment configuration in a single operation, without using multiple scripts



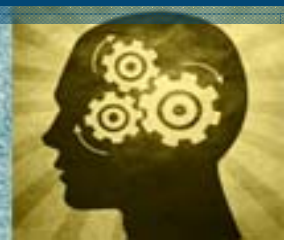
Fiorano

Enabling change at the speed of thought

PUTTING THEORY TO PRACTICE



TECHNOLOGY HYPE IS CONFUSING THE ISSUES WITHOUT PROVIDING A CLEAR PATH FORWARD



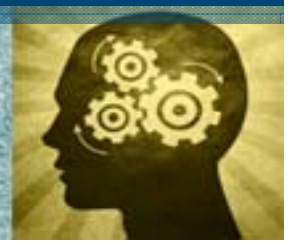
Example:ZapThink view

ZapThink view

- Need to choose standards that work today and will evolve for future
- Solve the real integration problem – more than a Proof of Concepts
- Must build incrementally on top of existing systems

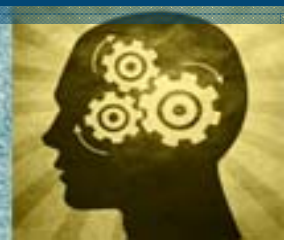
STANDARDS REDUCE COSTS BUT…

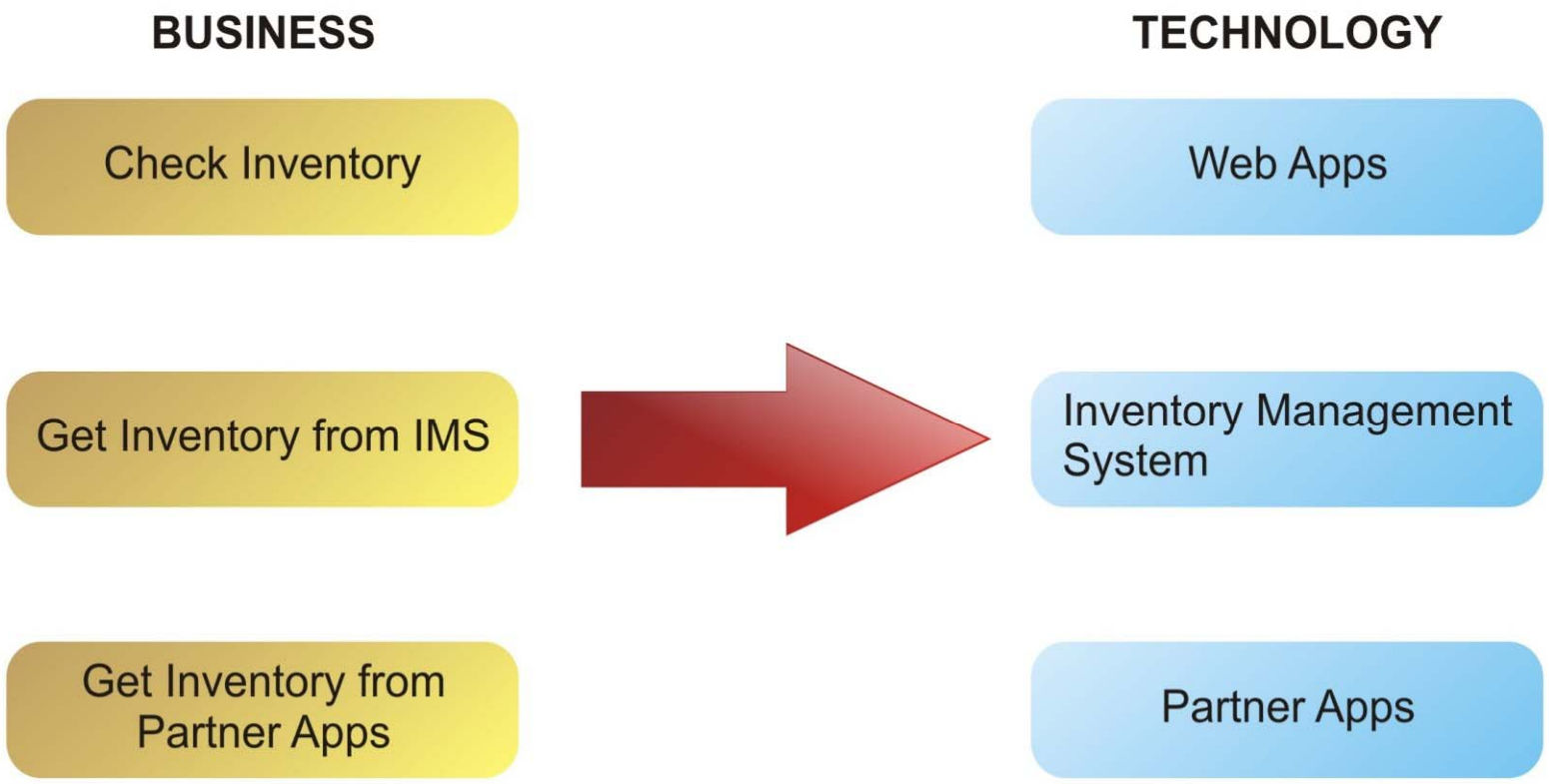
FUNDAMENTAL PROBLEMS REMAIN!



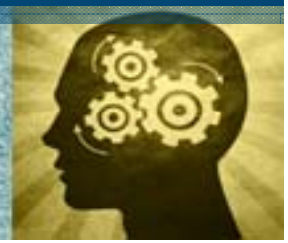
- Business Person's View
  - ✓ High-level model of business process flow
- IT Level View
  - ✓ Implementation flow differs substantially from business process view
- Impedance mismatch creates fundamental problems
  - ✓ Implementations have too many "moving parts"
  - ✓ Business-level change requirements difficult and time-consuming to implement

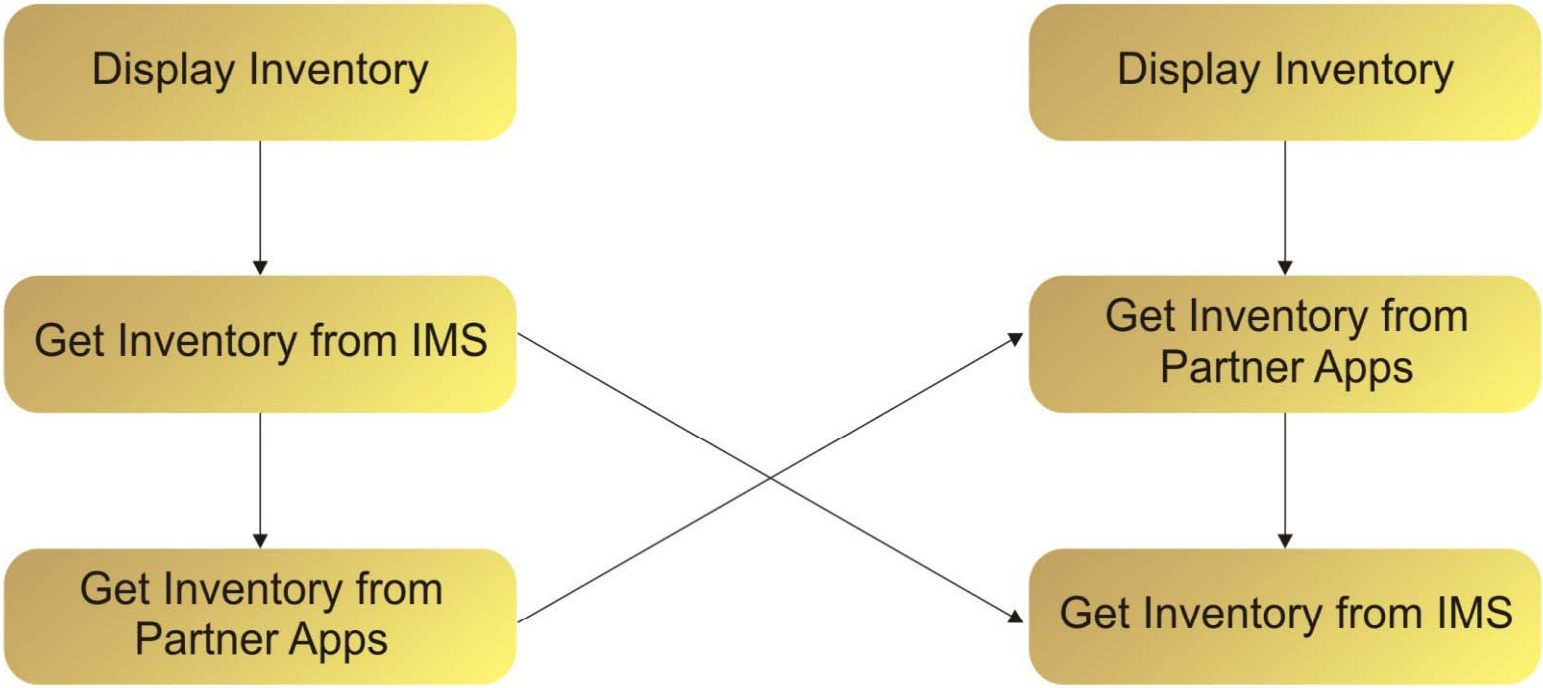
THE FUNDAMENTAL PROBLEM – DIVERGENT BUSINESS TECHNOLOGY



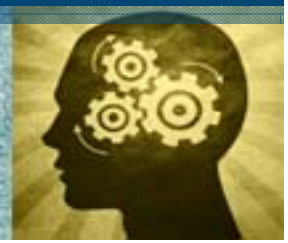


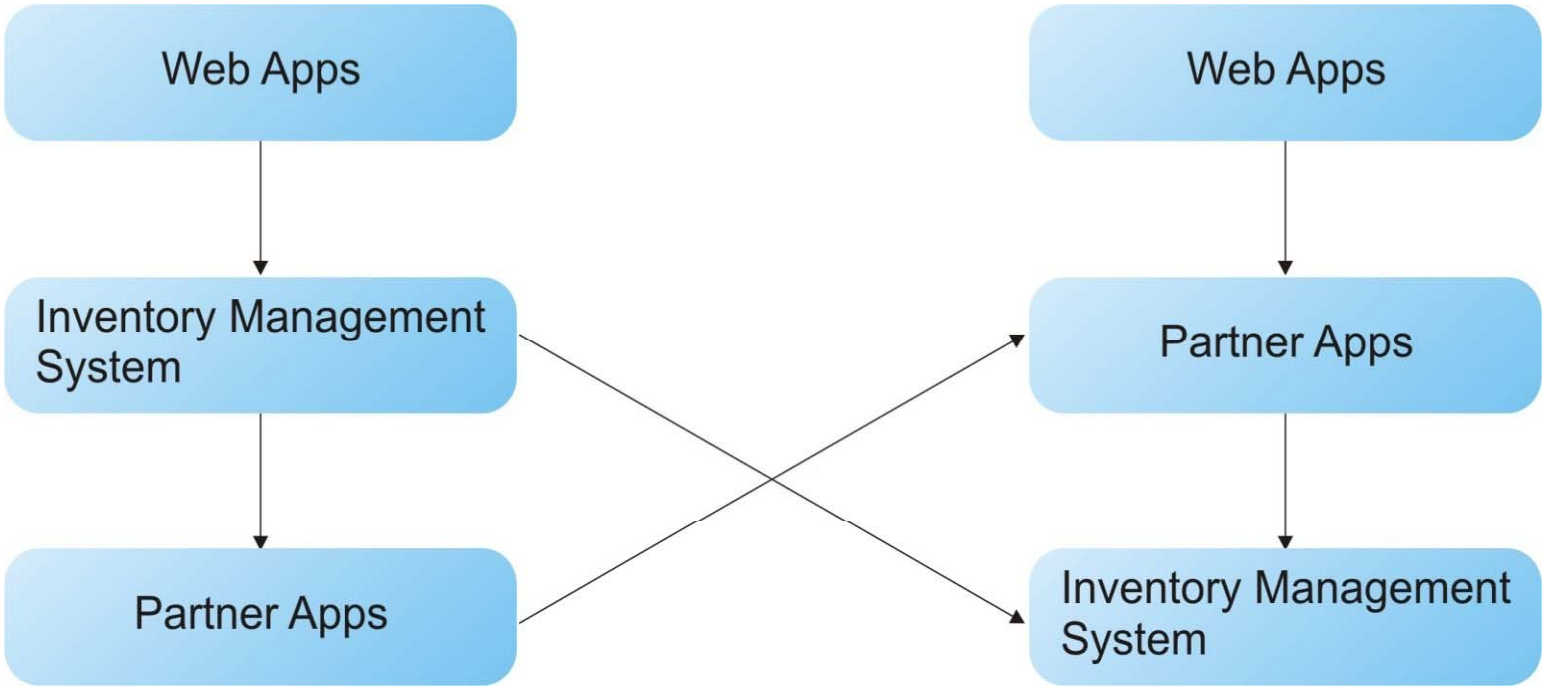
A CHANGE IN BUSINESS PROCESS



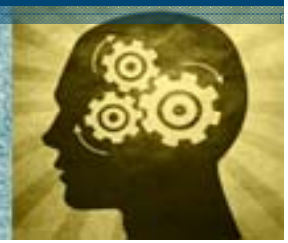


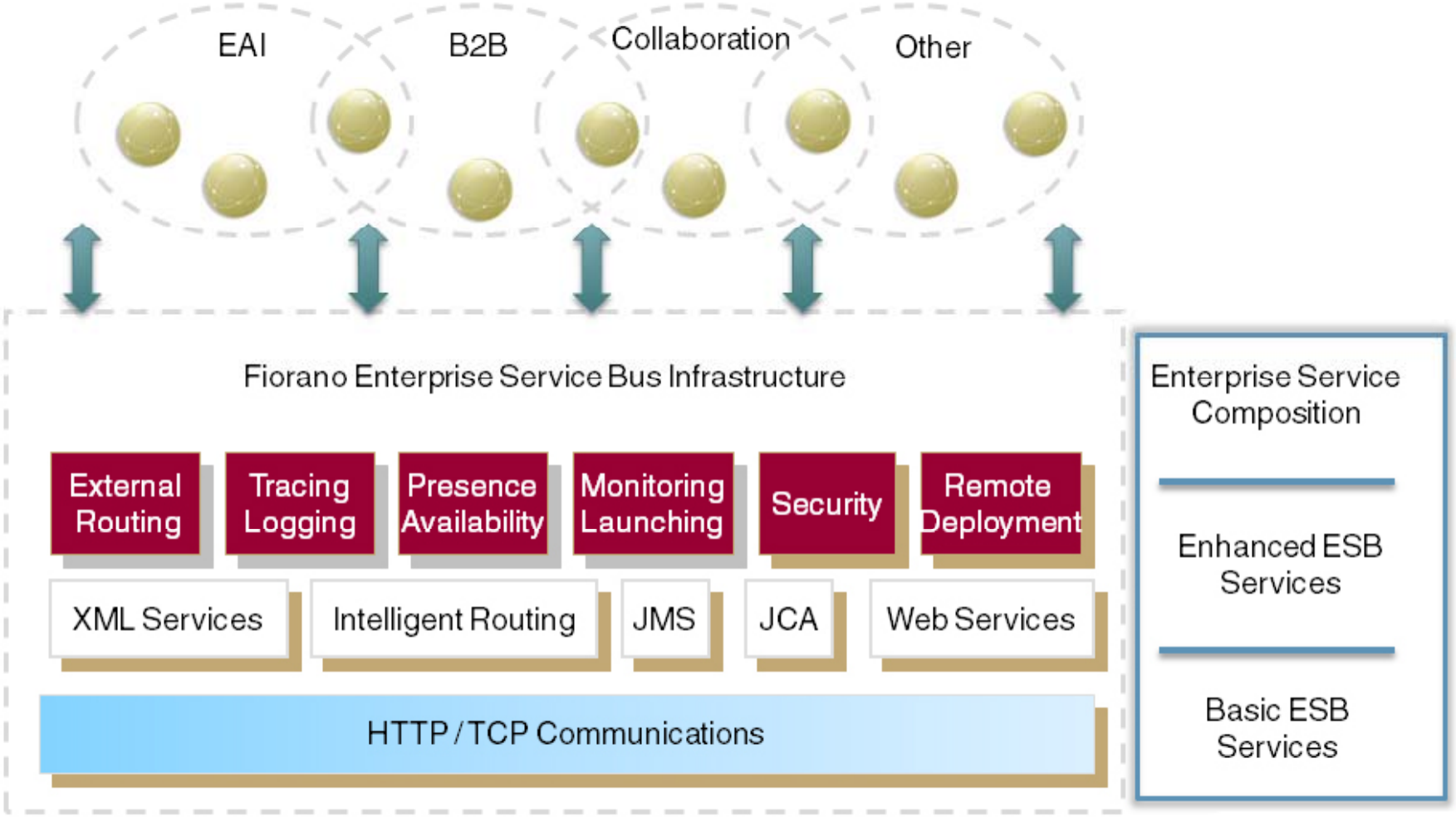
IS NOT EASILY MAPPED TO IMPLEMENTATION LEVEL

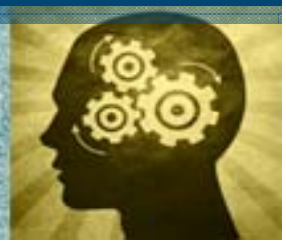




FIORANO ESB
PEER-TO-PEER ESB







THANK YOU!

For more details please visit: www.fiorano.com