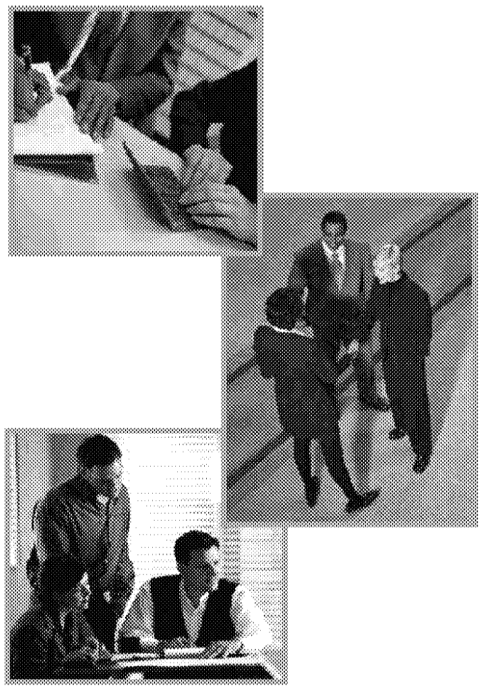
POL00137239

Gartner



Gartner Consulting

Royal Mail Group / Post Office Ltd.

Horizon On-Line Design Review

Gartner Findings

Version 1.1

03 October 2008

Engagement: 222387830

POL00137239

Horizon ⟹ Horizon On-Line

Royal Mail Group Ltd.

Post Office Limited (POL) has 13,500 branches throughout the UK. *Each year some 1.6 billion transactions are carried out across the counters*. These are varied in nature, from on-line banking, to complex postal services, to straightforward retail sales.

The current Horizon system was implemented to support the operation of the counters 10+ years ago. At a high level Horizon has the following functions:

1. Transaction capture and support (an ePOS system)
2. Maintaining accounts for the Sub Postmaster
3. Stock management in the branches
4. Necessary technical support functions such as Estate Management, User access control, security, etc.

POL has commissioned Fujitsu Services to upgrade Horizon to Horizon On-Line.

POL00137239

Objectives of Horizon On-Line

Objectives of the new Horizon On-Line, or "Horizon Next Generation" (HNG):

- The primary objective of the upgrade is to **reduce the costs** of the Horizon system to POL.
  1. Reduction of IT operational costs in the data centre, support, etc.
  2. Reduction of enhancement costs going forward
  3. Reduction of staff time to handle transactions in the branches
- There is no aim to provide additional business capabilities at this time; the concept of 'business equivalence' has been agreed – the new Horizon On-Line system must provide the same business outcomes as Horizon.
  - However the HNG design architecture should align with POL's emerging IT strategies for future growth (see next slide)
- Upgraded training facilities (for Counter Training Offices) are required.

POL00137239

Project Team

Scott Lewis, Group Vice President

Gary Long, Vice President Emeritus

Gartner Analyst Community

- Jess Thompson (United States)
- Massimo Pezzini (Italy)
- Kimihiko Iijima (Japan)

Engagement Manager

Consultant

Integration and SOA Analysts

POL00137239

Approach & Methodology: Gartner Value

Objectivity	■ Our objectivity ensures our approach and findings are based on sound data. ■ Objective approaches and findings foster outcomes geared towards your organisation — not solution driven by vendors or system integrators.
Technology Understanding	■ Our understanding ensures that we acknowledge the obvious, but more importantly understand the subtle, complex influences and interrelationships among market trends that will, together, materially affect your strategy. ■ Using proven data means that analyses are rooted in facts, not based on assumptions and extrapolation.
Methodology Expertise	■ We develop logical and rigorous approaches that leverage our knowledge. ■ Our methodologies allow us to incorporate a range of data, ask the right questions and develop traceable, defendable results.
Worldwide Scope	■ Worldwide scope ensures that proven capability exists for all regions. ■ Our worldwide scope also allows us to understand when technology trends in one area have application to those of another region, or where regional differences exist that will affect market strategies.
Credibility and Experience	■ Market analysis often works with incomplete, forecasted data; credibility and experience means we stand by our results — our name makes it easier to compel/defend decision-making within your organisation.

POL00137239

Objectives of This Review

POL00137239

Objectives of the Design Review

The purpose of the review is to assess the proposed Horizon On-line design against Post Office Limited's (POL's) emerging IT strategies.

1. Emerging Strategy: New Customer-Facing Devices

- *Does the design allow additional devices in the branches to be integrated with Horizon On-Line efficiently and effectively?* Note that POL may wish to source such devices and the software running on them from alternative suppliers.

2. Emerging Strategy: Ability to Use 3rd Party Services

- *Will it be possible to integrate software components from other suppliers into the Horizon On-Line system?* Note that for this to be of practical value the components would have to deliver significant business functionality that does not exist in Horizon On-Line, so to make this relevant examples of the functionality that POL could require in the future will need to be identified.

3. Emerging Strategy: Use of POL Services by Partners

- *Are the types of [HNG] component that POL may wish to exploit from other channels sufficiently well encapsulated and independent that they will be exploitable?*

POL00137239

Objective 1: New Customer-Facing Devices

Does the design allow additional devices in the branches (and elsewhere) to be integrated with Horizon On-Line efficiently and effectively?

Examples:

- Another retailer's terminal
- Kiosks – which use central services for pricing and payment
- Hand-held devices
- Smart card handlers
- Call Centres
- Consumer access to "virtual branch" services over the Internet [1]

[1] Many such services are provided today (see www.postoffice.co.uk) but do not use the same software or infrastructure.

POL00137239

Objective 2: Ability to Use 3rd Party Services

Will it be possible to integrate software components from other suppliers into the Horizon On-Line system?

Examples:

- Perform identity validation using third party services, such as credit agencies.
- B2B "cross-checking": Mr. Jones just deposited GBP 20,000 – is this reasonable?
- Post offices enter a brand new line-of-business with a more-complex multi-step business model
  - For example, consider selling health insurance where a physical examination is required
- A post office terminal or a kiosk might be fitted out with a software abstraction of a PIN pad.

POL00137239

Objective 3: Use of POL Services by Partners

Will new HNG components (services) be independent and well-encapsulated, so that they may be "consumed" from new channels?

Examples:

- Major partners could develop their own applications (clients) that invoke HNG components (services)
- The Royal Mail could build totally new products, that utilize some POL services

POL00137239

Analysis of the Review Objectives

POL00137239

What is Needed to Meet the Design Goals?

Objective 1: **New Customer-Facing Devices**

- There must be reasonable expectations that most HNG services can be "plugged in" to the applications running on the new (unknown) devices.

Objective 2: **Ability to Use 3rd Party Services**

- There must be a reasonable hope that the 3rd party services could be used, even though they (a) would belong to many different organizations, (b) would be written in undetermined computer languages and (c) execute on a variety of hardware platforms.

Objective 3: **Use of POL Services by Partners**

- This is essentially the inverse of Objective 2.

+ For all of the above, the cost and time to integrate must be reasonable.

+ Existing (well-designed) services should not have to be changed much if at all.

+ Security barriers must be scaled.

POL00137239

Architectural Challenges

1. Application Integration

Application integration is defined as "making independently designed application systems work together." It encompasses everything from tightly coupled, request/reply exchanges among interdependent systems to simple, arms-length batch file transfers between separate systems.

The two primary forms of application integration are:

- "application to application" (A2A) integration of the enterprise's applications
- "business to business" (B2B) integration, between the applications of disparate organizations.

2. Multi-Channel Architecture

Gartner defines multi-channel integration for retail delivery as the ability to manage all transactions, data and workflows among customers, staff and back-office systems across several channels. Multichannel integration includes at least two (and potentially all) customer channels — including self-service devices such as kiosks and interactive voice response (IVR), branch, offices, contact centers, online banking and mobile devices.

Source: Gartner Research

POL00137239

What is Needed … (continued)

What is needed is a "Lego Block" approach … *but the blocks are coming from different parties*.

"Plug and play" is another way to think of the problem … but Windows plug and play *happened only because Microsoft was dominant* (and wealthy) enough to make it happen.

By necessity, POL must be able to cope with sourcing hardware and software from disparate, as-yet-unknown suppliers.

The best chance of realizing the goals lies in:

1. Use of a "service-oriented architecture" (SOA) to govern the design
2. Use of Internet (Web) technology and protocols
3. Reliance on industry standards that are vendor-neutral and relevant to SOA and the Web
4. Use of robust computer-to-computer "middleware" to make integration easier and more reliable.

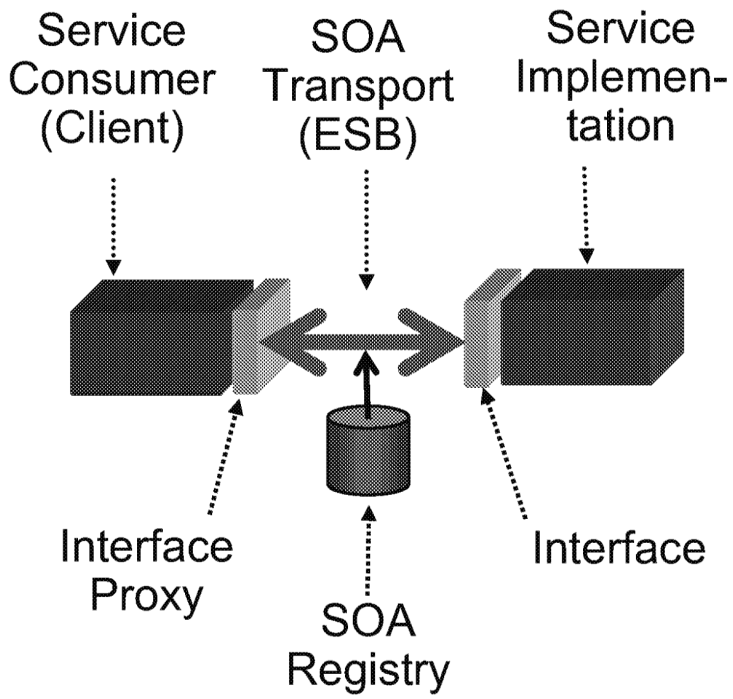
POL00137239

Service-Oriented Architecture

A Quick Review of SOA Basics

POL00137239

How Do You Know SOA When You See It?



- ✓ **Back end separated from front end**
- ✓ **Separately standing interface definitions**
- ✓ **Loosely coupled deployment**
- ✓ **Usable and useful across applications**

Source: Gartner Research

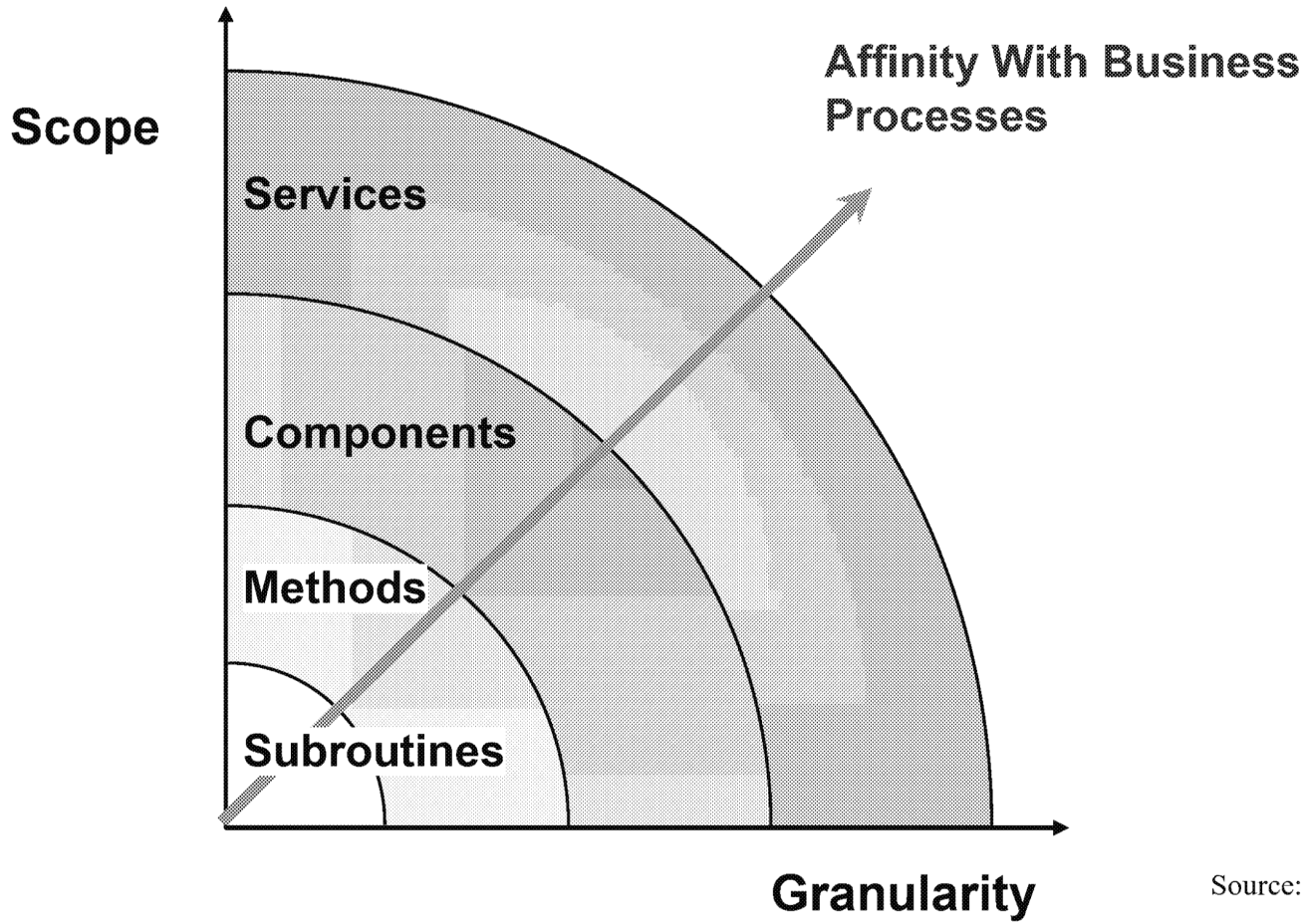
A service (in software) is a software component that's suitable for cross-application access via a separately-defined interface. Unlike the other kinds of software modules, a service represents a business function, although it's implemented as technical software components. Some application architectures also break out separate "data services".

Services are designed to be building blocks for larger processes, transactions or applications. SOA "request/reply services" are the most common and easily understood. Advanced SOA also includes "event services".

POL00137239

Evolution of Software Modularity

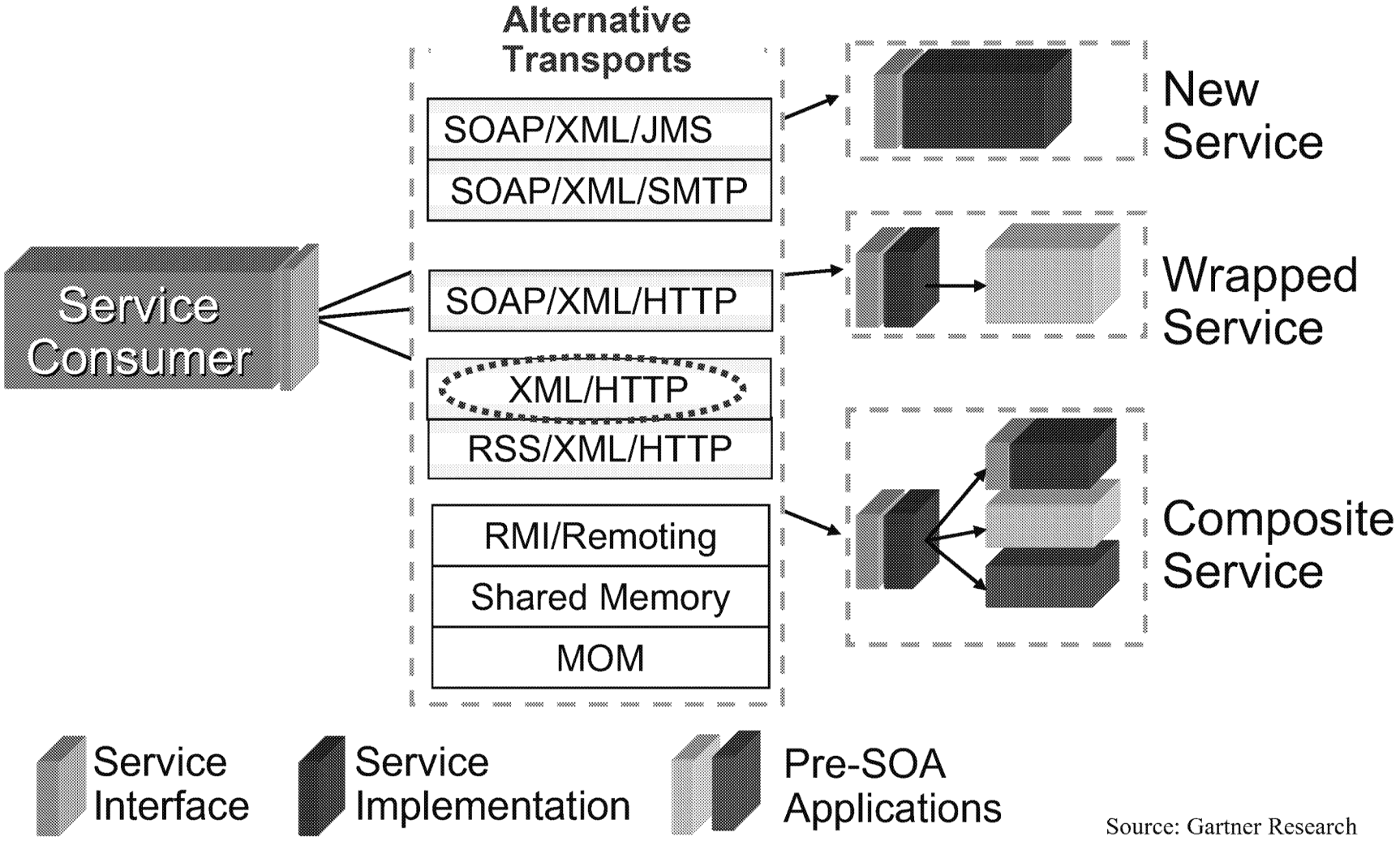
From Subroutines To Services



Source: Gartner Research

POL00137239

Inside SOA: Behind the Interface

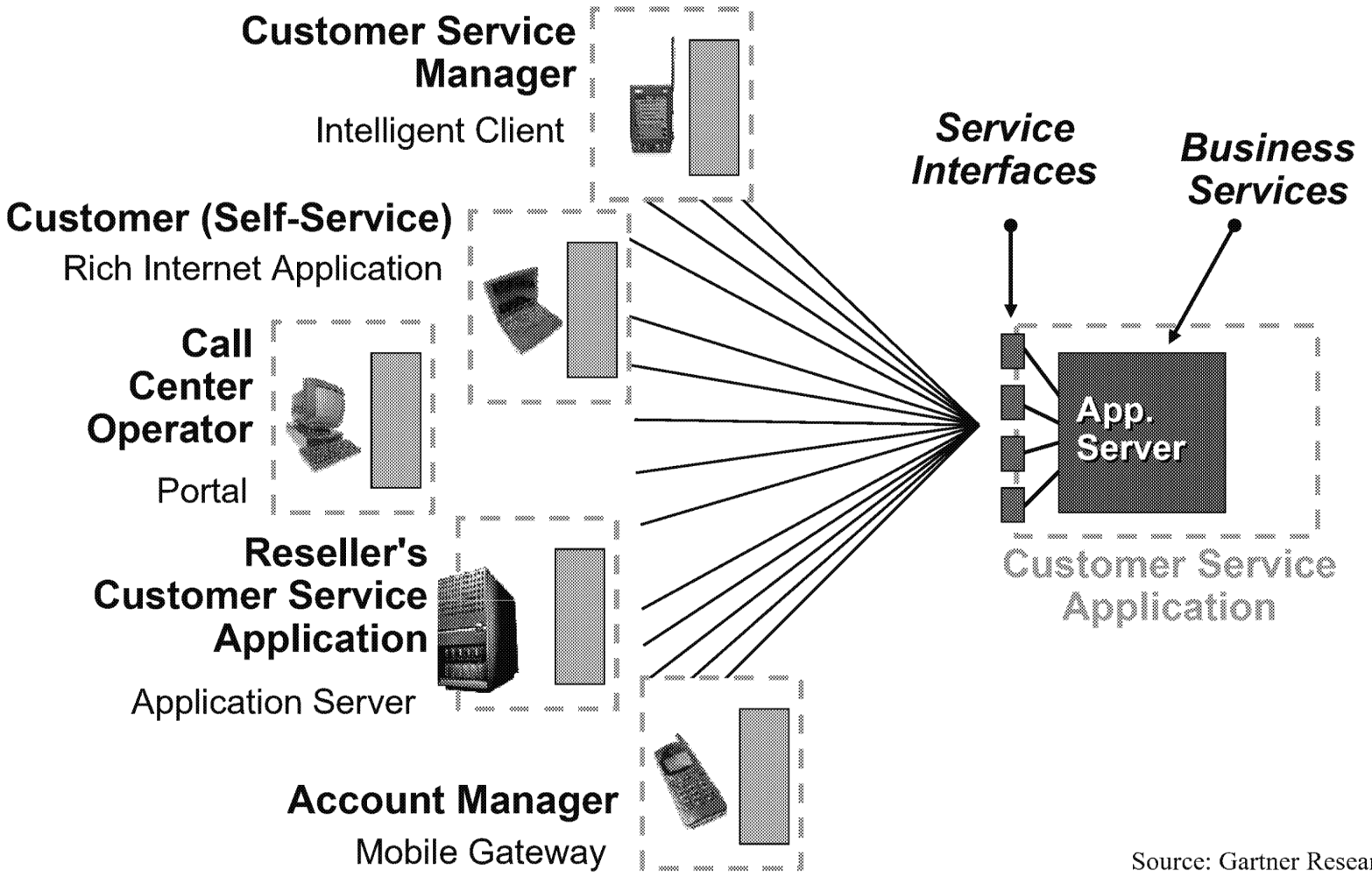


Source: Gartner Research

POL00137239

Royal Mail Group Ltd.

SOA R/R: Perfect for Multichannel Applications

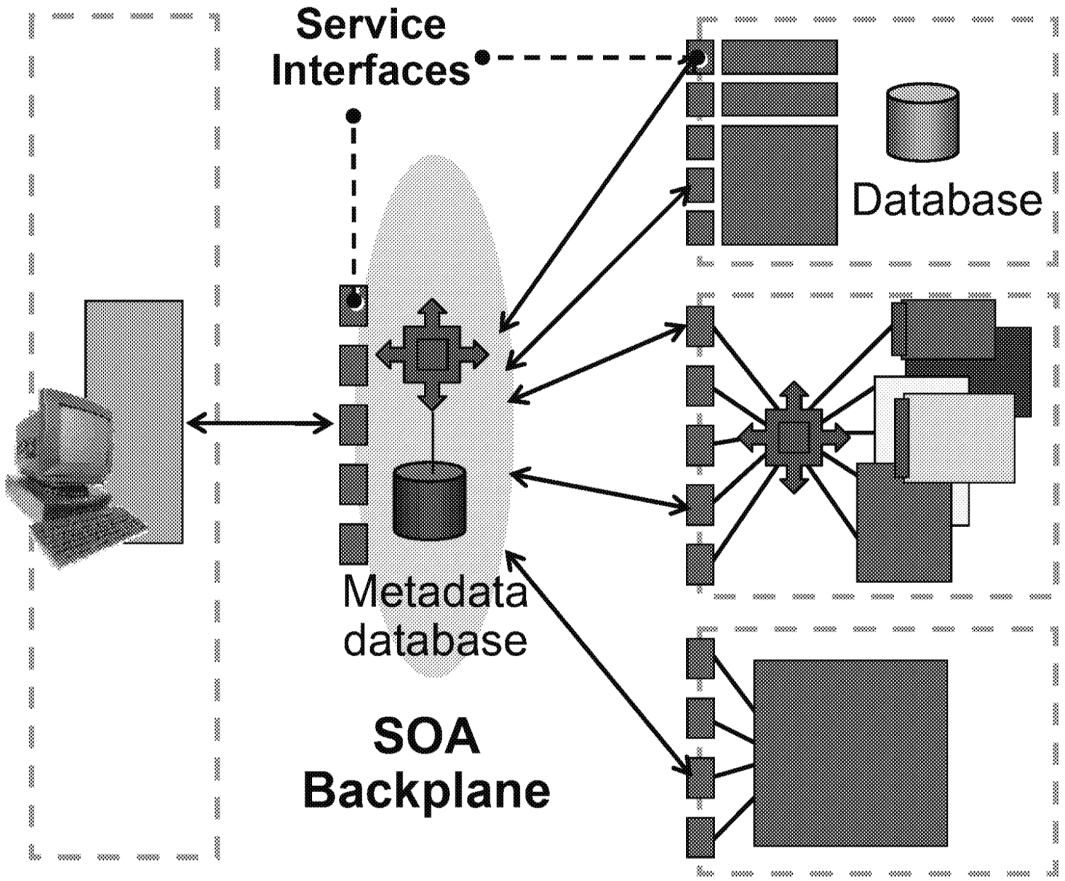


Source: Gartner Research

POL00137239

SOA and Composite Applications

Potential future use at POL – after Horizon Online is deployed



Checking Account
(new SOA application)

Customer Record
(integration of multiple SOA and pre-SOA applications: purchased, legacy, outsourced)

Credit Bureau
(external service provider, unknown architecture)

Example: Checking Overdraft Approval Transaction

Source: Gartner Research

POL00137239

Royal Mail Group Ltd.

The Seven Golden Rules for the Perfect First SOA Project

1. **Set Goals and Collect Business/IT Requirements**
2. **Segregation of Duties**
   - ✓ Application Design Teams:
     - Service consumers
     - Service implementations
   - ✓ Infrastructure Design Team (the future SOA CoE)
3. **Joint Design/Independent Implementations**
   - ✓ Services jointly designed by application teams
   - ✓ Technically validated by the infrastructure team
4. **Deliver Infrastructure (SOA Backplane) *First***
   - ✓ Design, implementation, testing
   - ✓ Service registry
   - ✓ Validation against an agreed proof-of-concept
5. **Deliver Services *Before* Consumer Applications**
   - ✓ Plan for services to be available and tested before relevant consumers
6. **Test, Test and Test**
   - ✓ Plan for at least 25% of development effort on integration testing
7. **Log, Log and Log**
   - ✓ Multiple turn-on/off logs on the "borderlines"

Source: Gartner Research

POL00137239

Middleware

The "Plumbing" for Modern Applications

POL00137239

What is Middleware?

SOA Middleware:

- Application Server
- Enterprise Service Bus (ESB)
- Integration Suite

Middleware is the software "glue" that helps programs and databases work together, even though they may be on different computers.

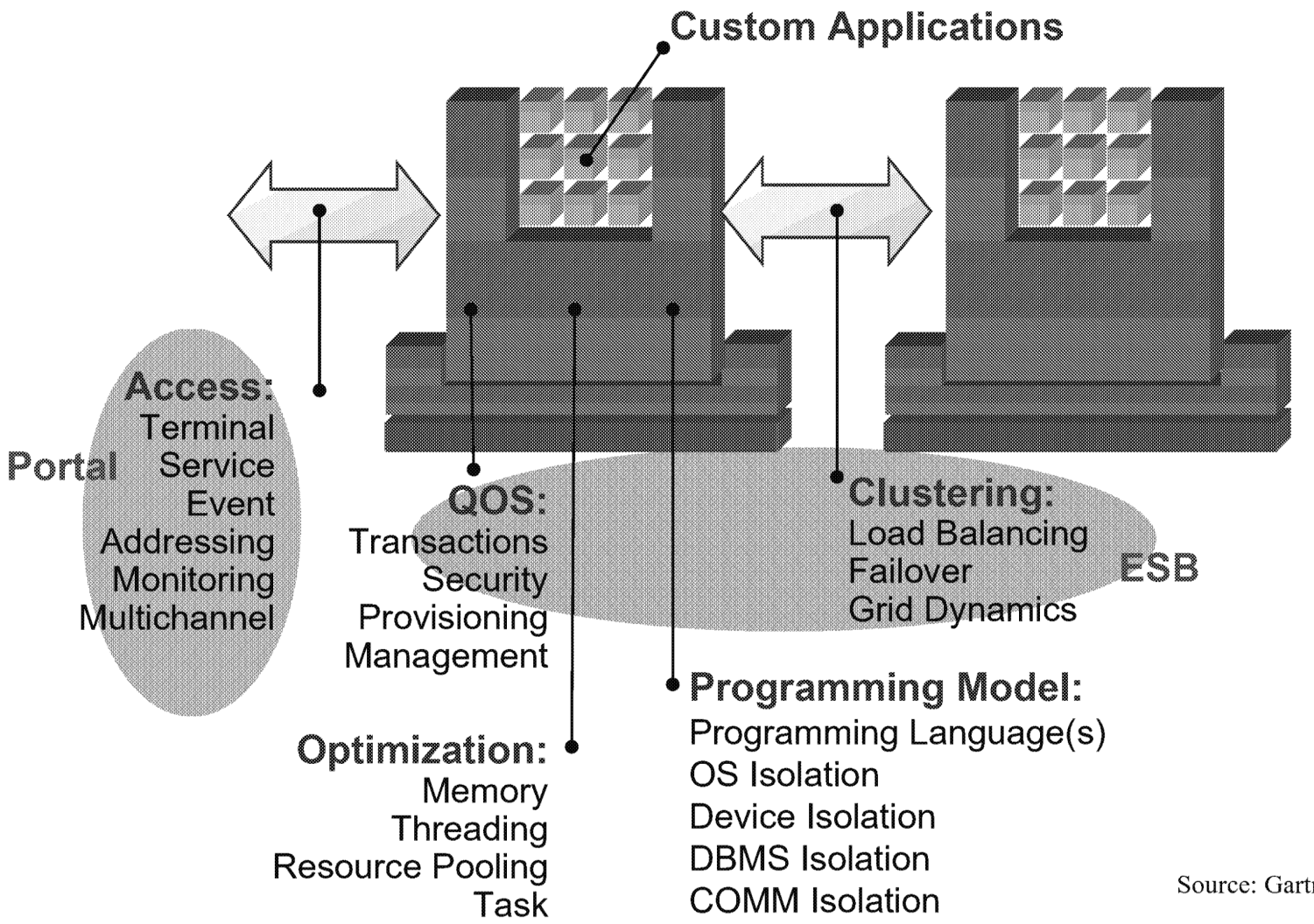
Middleware is not inherently complicated or mysterious, but it can seem that way if the concepts and terminology are not explained.

Service-Oriented Applications are not practical without one or more forms of middleware. Each of the middleware types discussed here may be needed, depending on this size and complexity of the SOA initiative(s).

Also see Appendix A: *The Enterprise Service Bus: Communication Backbone for SOA*

POL00137239

Application Server: Container and Programming Model

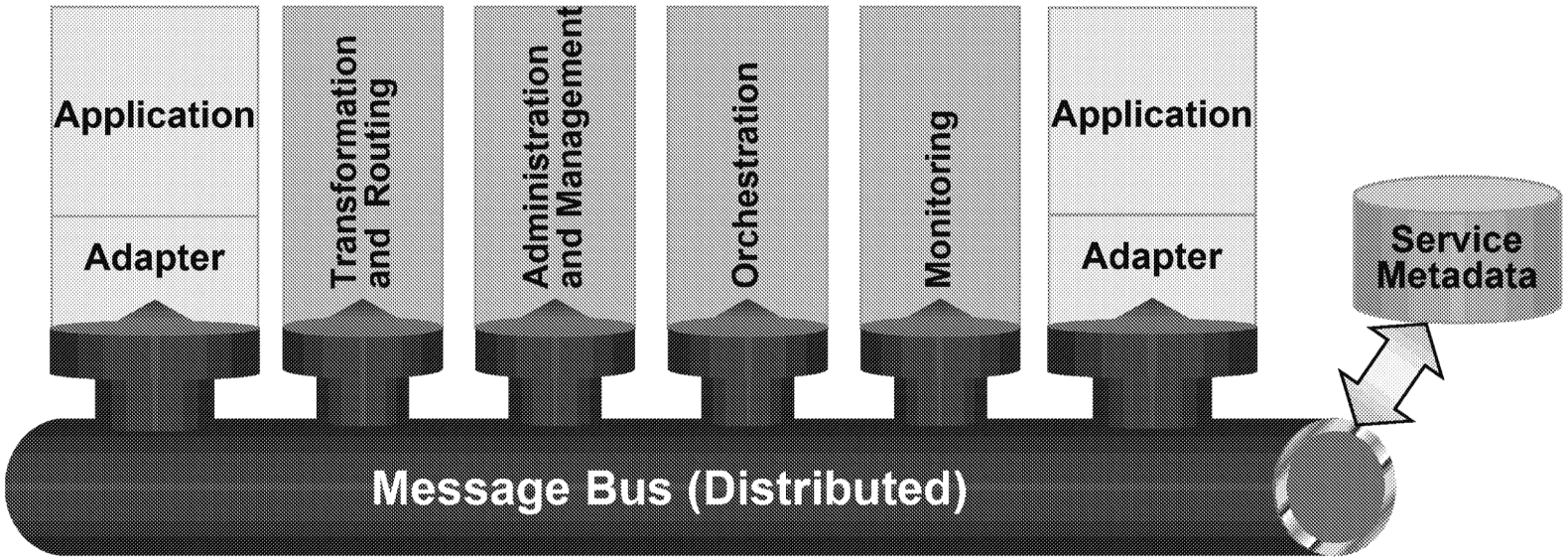


Source: Gartner Research

POL00137239

ESB Technology (SOA Backplane)

Royal Mail Group Ltd.



Core Service Bus Technology

- Program-to-program communication
- Support for Web services standards
- Service bindings
- Mediation
- Message Logging
- Protocol Bridging

Typical ESB Product Features

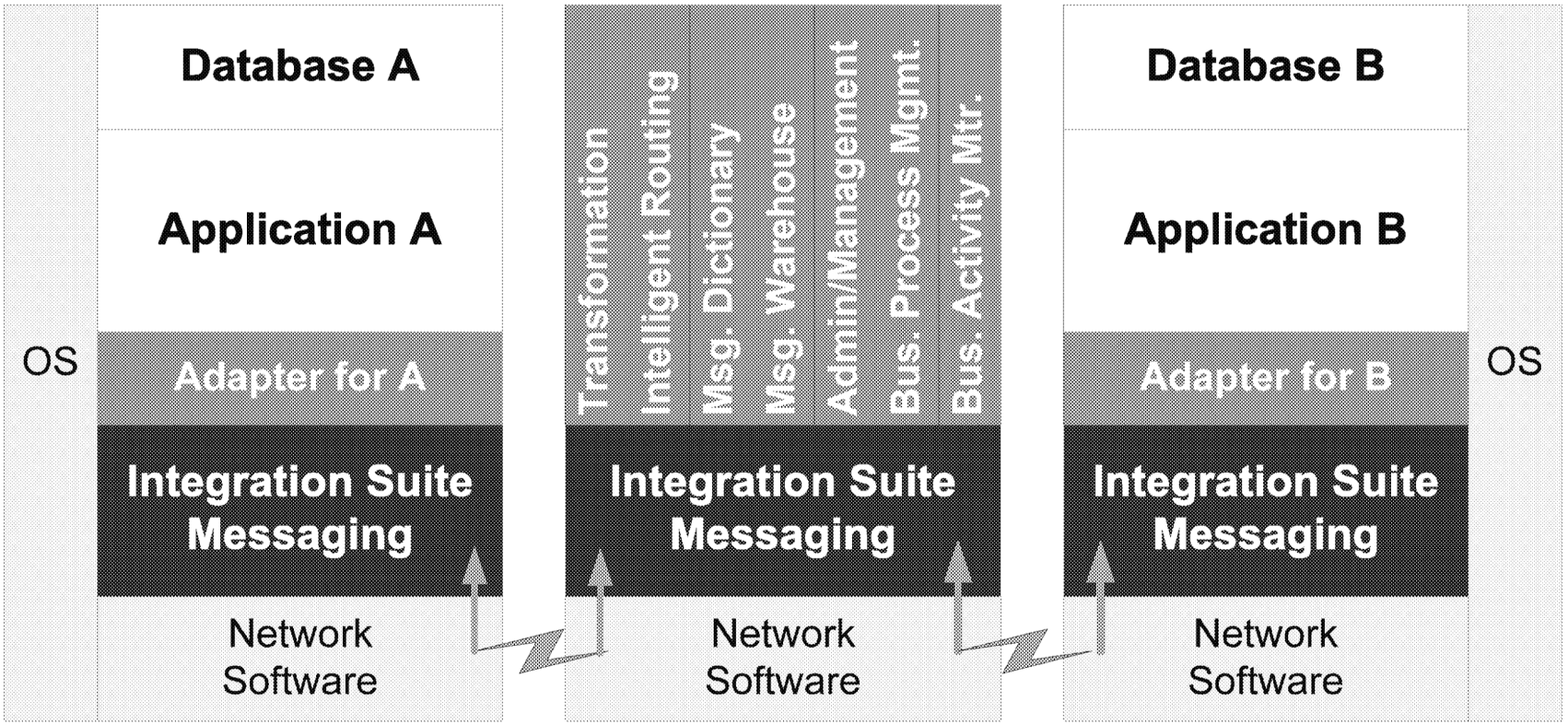
- Transformation
- Service registry
- Content-based routing
- Service orchestration
- Security
- Adapters

Source: Gartner Research

POL00137239

Royal Mail Group Ltd.

Integration Suites: Wide Ranging Capabilities



Capabilities include ESB features, plus more. The two are becoming more and more alike.

Source: Gartner Research

POL00137239

Integration Middleware Sources

	Source	Example Vendors / Comment
1	**Software Vendors (ESB and IS)**	**IBM, Fujitsu, TIBCO, Software AG, etc.** • Upfront costs are significant • Extremely competitive market (good) • Validated against a wide base of users • Support and regular enhancements
2	**Open Source Software (ESB)**	**Apache, ChainForge, MuleSource, Sun, Red Hat** • Not normally as feature-rich; ESBs not ISs • Can be very cost-competitive, initially • Vendor support may be available • Internal support needed as well
3	**Custom Built**	**Rare, but Happens** • Very costly, both development and enhancements • If upfront cost is a barrier, consider OSS • Last resort!

POL00137239

"HNG-X" Architecture Review

The Project Team's Architectural Approach

(As Relevant to the Design Review Objectives)

POL00137239

"HNG-X" Architecture Review

1. **Reviewed "Solution Architecture Outline" + 12 Detailed Architecture Documents**
2. **Concentrated on:**
   - High-Level Architecture
   - Interface and Integration Points (per the questions posed by POL)
   - Middleware Usage
   - Protocols
   - Relevant Standards
3. **Annotated Key Architectural Diagrams**
4. **Reviewed Understanding with Fujitsu Services Team**

POL00137239

Fujitsu Services Architectural Approach

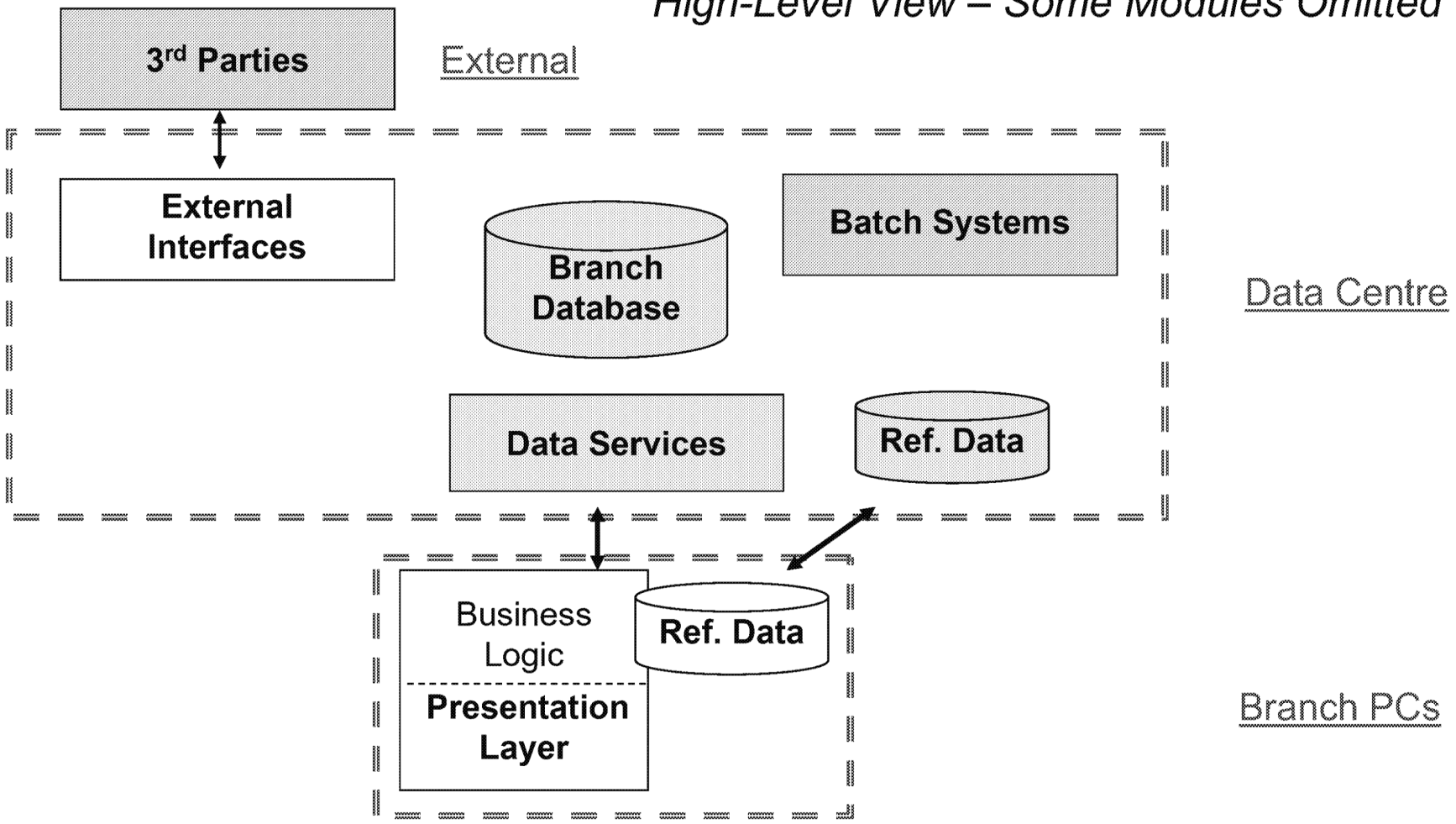
1. **Architectural principles are stated and seem generally sound and realistic**
2. **Documents are well written, thorough for the most part**
   - Include Assumptions
   - Include Risks / Risk Mitigation Approach
   - One key area – "PDL scripts" – was not covered in the architecture documents. (Fujitsu Services has since provided supplementary information.)
3. **XML usage is appropriate**
4. **Service-Oriented Architecture (SOA) with "Stateless" Request/Reply Services**
   - There are few if any HNG-X "business services"
   - HNG-X "data services" are a major improvement over current system
   - A number of encapsulated utility services exist
   - Architecture does not provide for Event Services
5. **Branch data is stored in a relational database, for the first time**
6. **Fujitsu "Interstage" Application Server is the Primary Middleware**
7. **Business logic and user interface are "data driven" by scripts (providing compatibility with the current Horizon business logic)**

POL00137239

Royal Mail Group Ltd.

HNG-X Major Components and Where

High-Level View – Some Modules Omitted



For a more complete picture, see the diagrams in Appendix A.

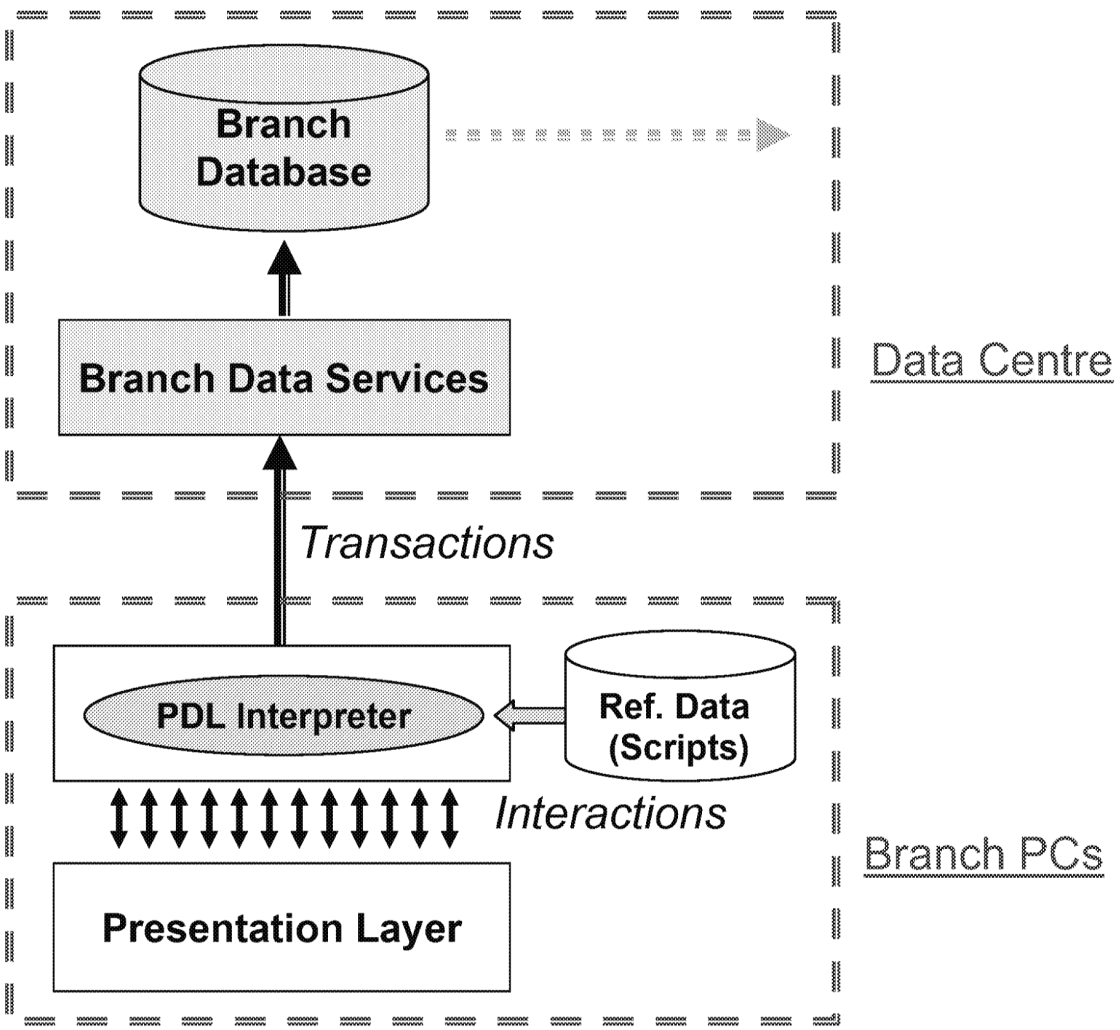
POL00137239

HNG-X Core Application Layers

Data services are relatively simple. They transform XML transactions into relational records and store them on the database.

Network traffic is minimized by sending completed transactions to the data center.

Business logic is determined by "scripts" which are interpreted by the Process Definition Language interpreter on the Branch PCs.



POL00137239

Branch PC Architecture

User Interface / Workflow & Scripting Approach

The XML-based Process Definition Language (PDL), run-time PDL interpreter and accompanying scripts are a unique characteristic of the HNG-X architecture. Rather than hand-coding screens, presentation logic and business logic for each transaction (there are hundreds), each transaction is embodied in a PDL script.

A generic user interface was written in Java and Sun's "Swing" GUI toolkit. The PDL interpreter, also in Java, is based on the open source "JEXL" interpreter from the Apache foundation. It has been lightly modified, mostly to provide stronger constraints.

Scripts (and associated context files) control such things as:

1. Variable portions of the screen display, including titles and field prompts
2. Edits
3. Usage of transaction-specific reference data, such as prices
4. Messages (error, confirmation, etc.)

Scripts also trigger the invocation of authentication, the recording of business events and various utility services that execute on servers at the data center (as they should).

Also see the diagrams in Appendix A.

POL00137239

Branch PC Architecture

Advantages:

1. Compact, flexible and efficient. Lets minimally-configured PCs execute hundreds of different transactions.
2. Post Office business analysts can create new transactions (in spreadsheets!) themselves. New transactions can be deployed without changes to the code base.
3. Backward compatibility with current Horizon (similar scripts).
4. Use of open source "JEXL" engine reduced development cost and risk.

Challenges:

1. May make "plug and play" more difficult.
2. Presentation and business logic are intermixed. (But it appears that there is very little complex business logic required.)
3. This is not a traditional approach. New developers will face a learning curve.

Conclusion:

1. All solution architects have to deal with "trade-offs". This is a reasonable approach given the requirements and constraints.

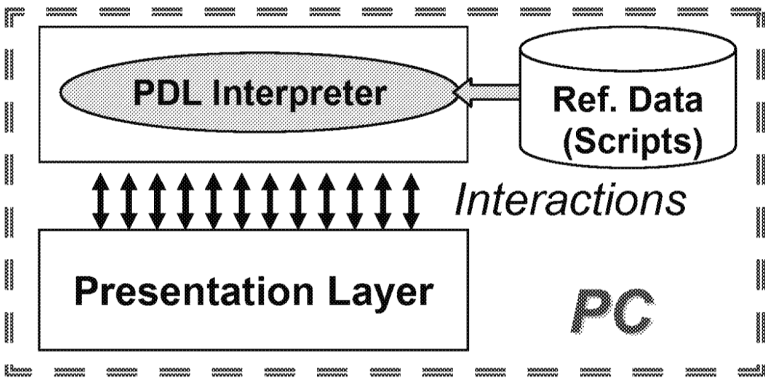
POL00137239

Does HNG-X Follow SOA Principles?

The HNG-X architecture is partially SOA.

- The data services are indeed SOA. They are properly encapsulated and invoked via XML over HTTP.
- Business logic is not encapsulated in services, and it resides on client PCs – not at the data centre.
- Business logic and (variable portions of) presentation logic are mixed together in the same scripts.
- External Web services can be invoked, from the Branch PCs, via proxy services at the data centre.

The placement of business logic on the Branch PCs and the use of a compact interpreter maximize the use of modestly-configured PCs and minimize the load on the network. It is not SOA, but it is a reasonable approach given the constraints.



POL00137239

13.1 Assumptions (from Branch Access Layer Architecture)

13.1.1 **Services are assumed to be stateless**

Services will be stateless and self-contained within one request/response exchange. …This assumption is consistent with the Post Offices requirement that the architecture is to be according to SOA principles.

13.1.2 **No further third party software licences will be acquired**

We have worked under the assumption that no further software licences for third party software will be acquired, and that what is provided by Interstage is our limitation. This assumption is important to make explicit, as it has guided some of our architectural decisions, such as:

-Creation of a custom NIO Integration Framework: An alternative to this framework, or at least the potential complexity of it would have been to get a licence for a highly scalable messaging system, such as IBM Websphere MQ and build on top of that. However, with third-party licences being deemed unsuitable due to commercial and cost reasons ...

13.1.3 **The network is limited to synchronous request/response interactions from the counter**

We have assumed, given the network topology and architecture that the most feasible and reliable mode of interaction between the counter and server is a traditional request/response synchronous mode of operation over http.

13.1.4 **An absolute minimum of changes will be done to existing backend infrastructure and applications**

There might have been a case for revamping the estate by creating a more unified integration backbone architecture using messaging and moving away from the current approach based on point-to-point integration with batch jobs and direct network calls to interrelated systems, which makes for a somewhat tightly coupled architecture. ***The architecture does open the way for the possible future implementation of an Enterprise Service Bus, something that should be considered as part of long term planning for the POL estate.***

POL00137239

13.2 Risks (from Branch Access Layer Architecture)

13.2.1 **Implementation risk of custom NIO integration framework**

There is an inherent risk in implementing the proposed NIO integration framework (section 2.4.3.2) proposed in this document. **The NIO API and writing concurrent code is error-prone and complex to test. It is possible that if the framework is not written by experienced and highly skilled developers that it will not be viable.**

13.2.1.1 Risk of occurrence

It is dependent on development experience.

If developers are inexperienced in developing I/O and concurrent applications, the risk is very high. If developers are skilled and experienced in writing I/O and concurrent applications the risk will be low to medium.

13.2.1.2 Risk impact

Very high – any integration with third party applications such as DVLA, Banking etc will fail.

13.2.1.3 Risk mitigation

Only allow the most experienced and skilled developers with experience in concurrent programming write any code on this particular component. Furthermore, it is important to do testing rigorously, including unit testing based both on deterministic and probabilistic approaches.

13.2.2 **Implementation risk of custom NIO HTTP Multiplexer**

(text similar to 13.2.1)

POL00137239

Middleware and Horizon Online

Middleware Type	HNG-X Architecture
Application Server	Fujitsu Interstage Application Server (limited use) and Application Client Software (JVM only)
Message-Oriented Middleware	Fujitsu Java Messaging Service (JMS) – limited use due to performance issues (Riposte is being retired with no direct replacement.)
Enterprise Service Bus (ESB) or Integration Suite	• Online Services Router (OSR) – Custom Built • NIO HTTP Multiplexer – Custom • NIO Connectivity Framework – Custom • Custom Transformers and Protocol Adapters
Other	• Branch Database provides the connection between branch transactions and the back-office (batch) apps • Sterling's "Connect:Direct" for managed file transfer

POL00137239

Findings

POL00137239

Question 1: New Customer-Facing Devices?

Does the design allow additional devices in the branches (and elsewhere) to be integrated with Horizon On-Line efficiently and effectively?

DIFFICULTY WILL VARY depending on the device and it's scope / ownership.

1. For a PC-based Kiosk running in a branch the workflow engine (PDL interpreter) could be ported and a new presentation layer constructed. Not too bad.
2. For a dedicated call centre PC, an approach similar to the existing one seems reasonable. The user interface would probably need to be modified / augmented to be more suitable for a call centre operator.
3. For hand-held devices that run Java a port might be possible. Data storage constraints might mean that only one or a few scripts could reside on the device at a time. Some logic might have to be moved to a server.
4. For other situations the user interface + business logic (script + interpreter) presents a substantial barrier.
   - For a Web browser, an "Ajax" approach with XHTML & JavaScript might replace the Swing & PDL script architecture?
   - In situations where only a few of the transactions are required, it would be easiest to simply re-code the transaction in a traditional fashion, and use only business-oriented reference data such as prices.

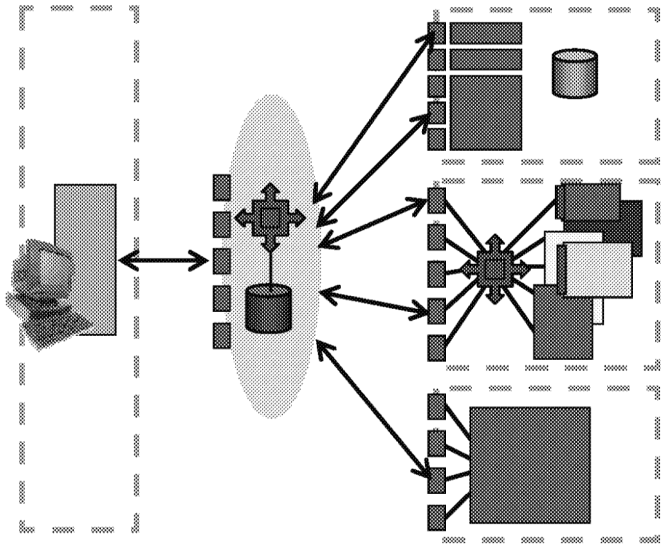
POL00137239

Question 2: Ability to Use 3rd Party Services?

Will it be possible to integrate software components from other suppliers into the Horizon On-Line system?

Answer:

- YES – If the new software components are well-encapsulated
  1. If they are SOA or Web Services (common)
  2. Or if they can be "wrapped"
- YES – because the HGN-X architecture has already provided proven methods for invoking external services through "proxies" at the data centre.



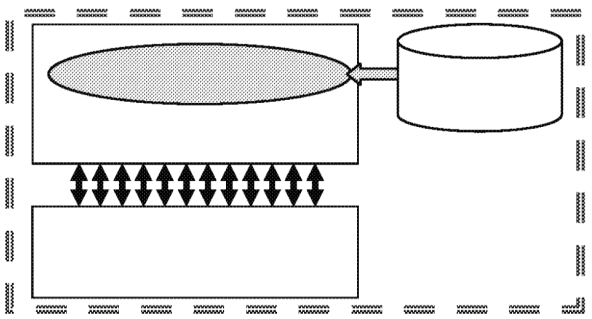
POL00137239

Question 3: Use of POL Services by Partners?

Will new HNG components (services) be independent and well-encapsulated, so that they may be "consumed" from new channels?

Answer:

- **YES** – For Data Services
- **NO** – For transaction capture and business logic
  - The PC-based business logic (script + interpreter) presents a substantial barrier.



POL00137239

Architectural Suitability

Summarizing The Answers To The Three Questions:

1. New Customer-Facing Devices?
   - **Yes**, but difficulty will vary depending on the device and its ownership
2. Ability to Use 3rd Party Services?
   - **Yes**
3. Use of POL Services by Partners?
   - **Yes** for data services
   - **No** for transaction capture and business logic (for good reason)

Soundness Of The Overall Architecture:

Overall, the HNG-X architecture is first rate. Gartner does not recommend changing it in any substantial way.

Placement of business logic on the Branch PCs and the use of a compact interpreter maximize the use of modestly-configured PCs and minimize the load on the network. All solution architects have to deal with "trade-offs"; the HNG-X architects crafted a strong solution that would be hard to improve upon.

POL00137239

Recommendations

(Current Project)

POL00137239

Primary Recommendations

1. **Reduce the amount of custom middleware and other sorts of systems software (replace with commercial products or possibly open source).**
   - The risk is high (appropriately recognized by the Fujitsu Services team)
   - Exceptionally skilled developers are needed – middleware is an arcane specialty
   - The cost of development and testing is high (and not yet complete)
   - Future maintenance will be a severe challenge and very costly
   - Future external integration projects will be easier with specialized tools
2. **Encapsulate Business Logic (if possible)**
   - The business layer and presentation layer are currently too-tightly coupled, making it hard to re-use business logic in new situations
   - Look at alternative designs that can lower the degree of coupling so as to further reuse of the more complex pieces of business logic in new situations -- without losing the considerable benefit of the scripting approach

POL00137239

Primary Recommendation #1: Commercial Middleware

1. Adopt Commercial or Open Source Middleware
   - "SOA Backplane" – Commercial ESB or Integration Suite
   - An open source software (OSS) ESB may also be considered
   - Use commercial or OSS software for load-balancing, Web server, etc.
2. Re-factor the HNG-X Application / Eliminate Custom Middleware Code
   - Branch Access Layer (BAL)
   - Remote Services Interface (RSI)
   - Online Services Router (OSR)
   - Other, as appropriate
3. Custom Transformers and Protocol Adapters
   - Plug into the ESB or Integration Suite
   - Future transformations may be done in the ESB or Integration Suite
   - Off-the shelf adapters are available

Also see Appendix A: *The Enterprise Service Bus: Communication Backbone for SOA; Where to Use an Enterprise Service Bus and Why* and *Open Source in ESB Suites, 2008*

POL00137239

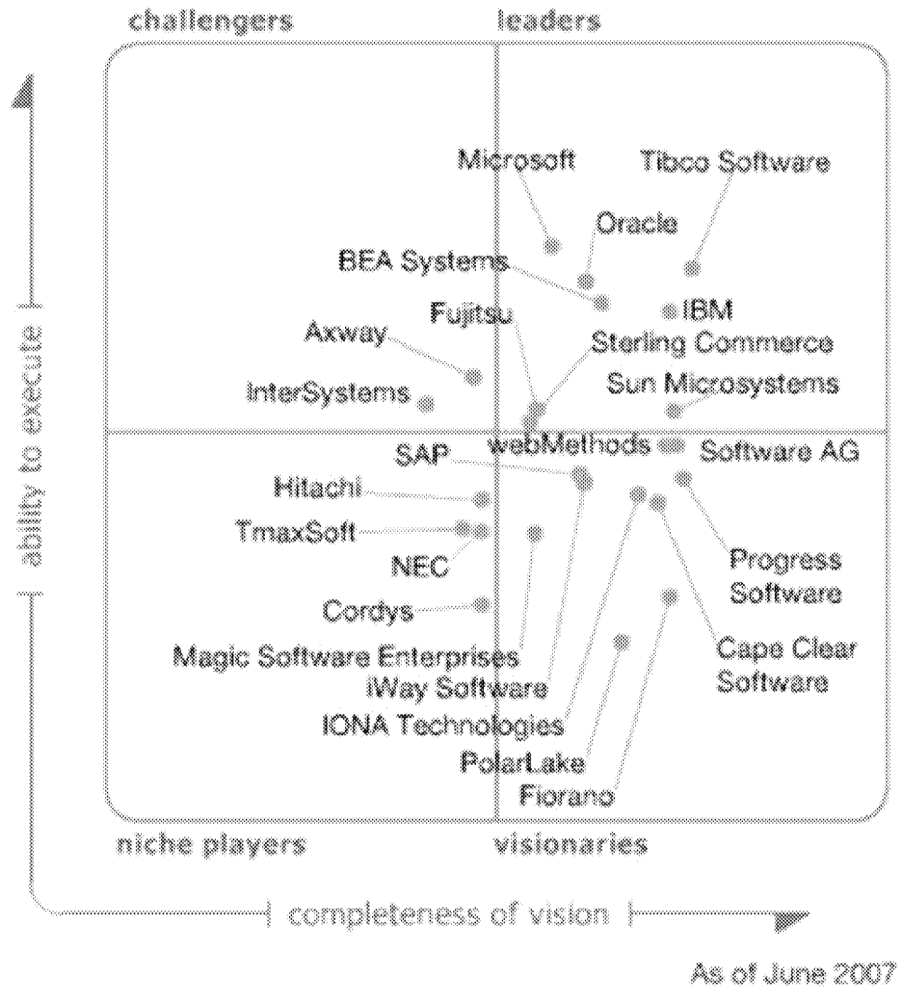
Likely Impact of Recommendation #1

Topic	Comment
Commercial Middleware	• Extra cost and time at outset • Offset by decreased development cost and time • Decreased testing costs • Considerable savings in future costs
Project Schedule	• Re-factoring may lengthen coding phase • Testing time will decrease
Deployment	• Lower probability of middleware defects surfacing in deployment • Lower probability of scalability and reliability problems • Shorter “Hydra” period (Horizon and Horizon Online in parallel)
Risk	• Reduced risk due to proven, industrial-strength middleware
Enhancements & Maintenance	• Lower costs over the lifespan of the application • Reduced middleware maintenance • Better middleware documentation, training, enhancements, etc.

POL00137239

Royal Mail Group Ltd.

Magic Quadrant for Integration Projects



Gartner's "Magic Quadrant for Application Infrastructure for Back-End Application Integration Projects" rates vendors that offer various products for this purpose, including ESBs and Integration Suites.

The HNG-X project does not currently use this type of product.

Since publication, Oracle has acquired BEA and Software AG has acquired webMethods.

Source: Gartner Research

POL00137239

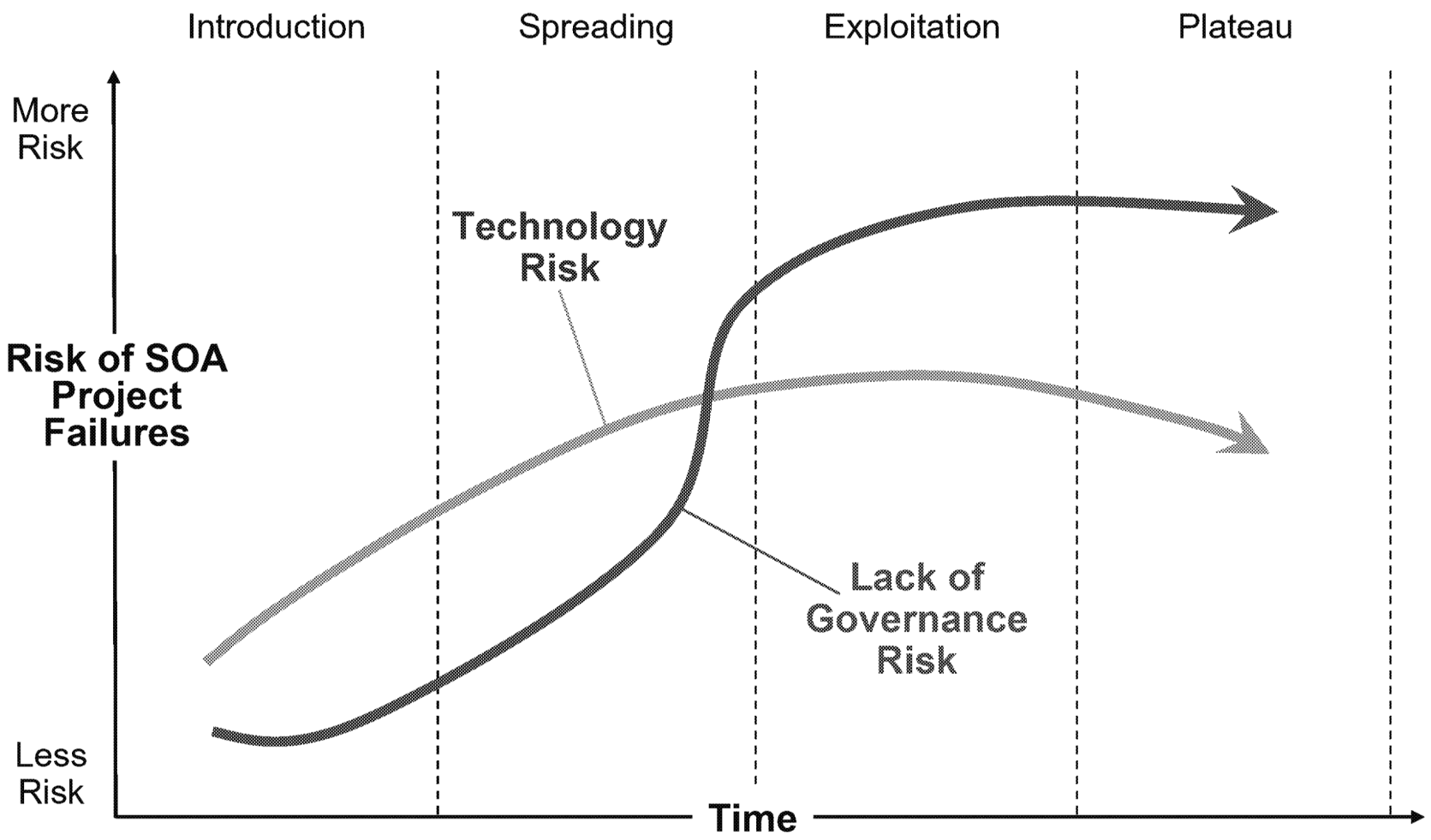
Recommendations

(Future)

POL00137239

Royal Mail Group Ltd.

Why SOA Initiatives Fail: Technology or Governance?



Source: Gartner Research

POL00137239

SOA Into the Future (Technical)

Get Good at "Technical SOA"

1. Master the SOA technology
   - Today it is new
   - Tomorrow it should be common, "standard" at POL
2. Move further towards "advanced SOA", including event services in addition to request/reply services.
   - Consider the use of event services (message-based) to integrate with the back-office (batch) applications on more of a real-time basis.
3. Promote the design of additional SOA applications
4. Be prepared to educate and train new project teams

POL00137239

SOA Into the Future (Governance)

Success Factors for SOA Governance

1. A funding model to maintain services as shared assets
2. Development process and architectural standards that incorporate SOA
3. Guidelines to address what to build for reuse and what is specific to an application
4. An Integration Competency Center (ICC) or "SOA Center of Excellence"
5. A process by which existing software is cataloged, understood, and harvested for services

You need just enough governance:

- *Too little governance will kill your SOA initiative*
- *Too much governance will kill your SOA initiative*

POL00137239

Realizing the Value of SOA

Exploit Shared Services

1. Largest single future payoff
2. Start with the Post Office Web site?
3. Be ready for the impact (succeed on the first try):
   - Some candidate services may need to be enhanced
   - Testing has to be extra rigorous
   - Support the new “consuming” applications intensively

POL00137239

Royal Mail Group Ltd.

How Do You Know If It's Working?

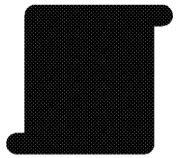
Goal/Focus	Examples of SOA Technical Metrics	Optimal Trend
Reuse	*# of Services Deployed* *# of Consumer Applications Deployed* *# of Services/ # of Consumers* *# of Services Shared by at Least Two Applications* *Average Sharing Ratio*	
QOS	*Volume of Service Requests* *Amount of Requests per Service* *Service Request Response Time*	
Cost Reduction	*Number of New Services Developed per Each New Consumer Application* *Time to Deployment for New Consumer Applications* *Cost of Application Maintenance*	

Source: Gartner Research

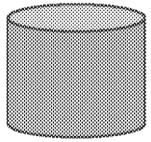
POL00137239

How Do You Enforce Sharing of Services?

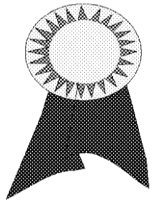
Service Definition Process



Standardized Business Objects



Service Registry and Life Cycle Management Tools



Reward Policies

Source: Gartner Research

POL00137239

Q & A

POL00137239

Q&A

Questions and Comments?

-
-
-
-
-
-
-
-

POL00137239

APPENDIX A

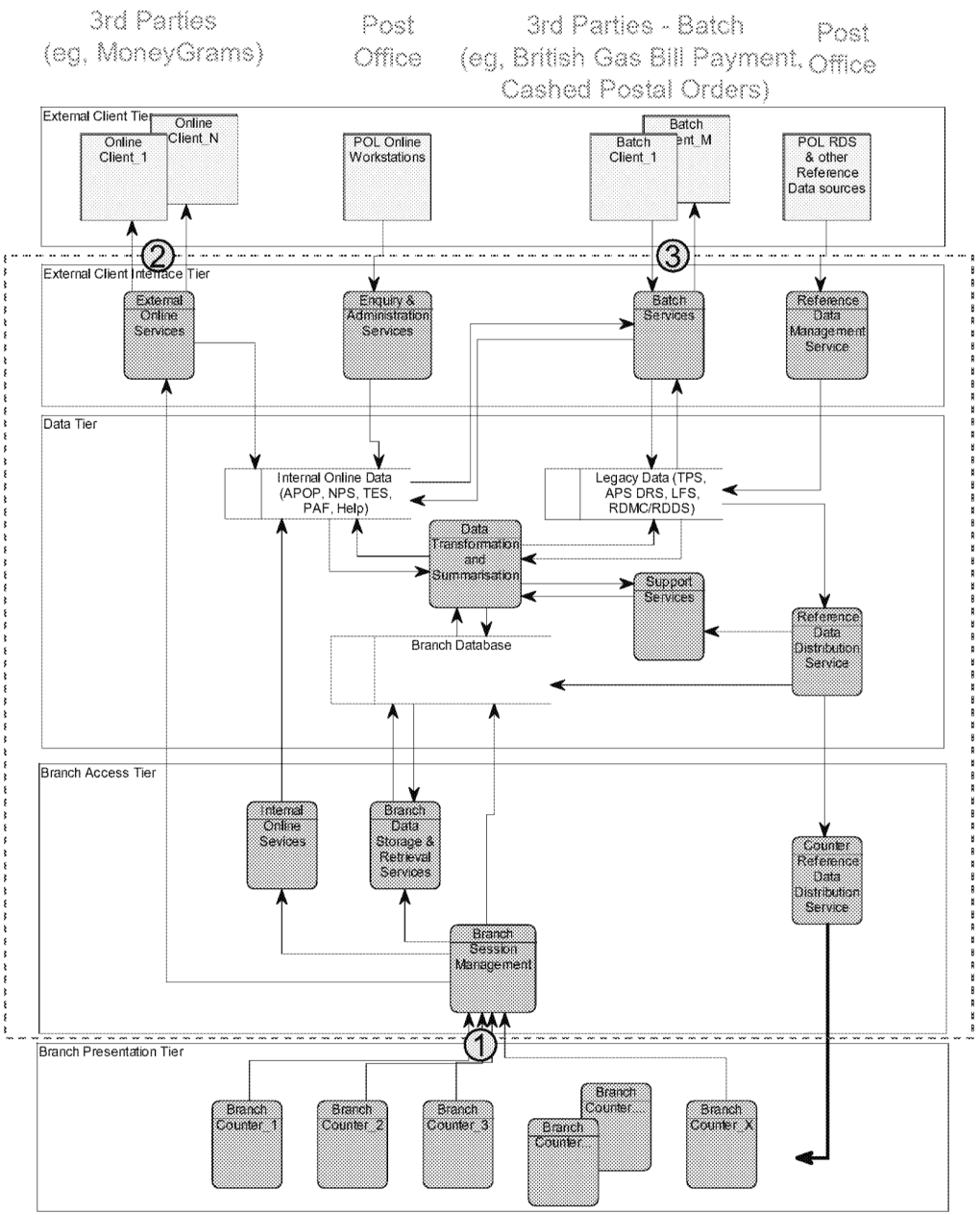
Diagrams from HNG-X Project

(Annotated)

POL00137239

HNG-X Application Architecture

(Overview)



① Via Remote Services Interface (RSI) - not shown

From Solution Architecture Outline

POL00137239

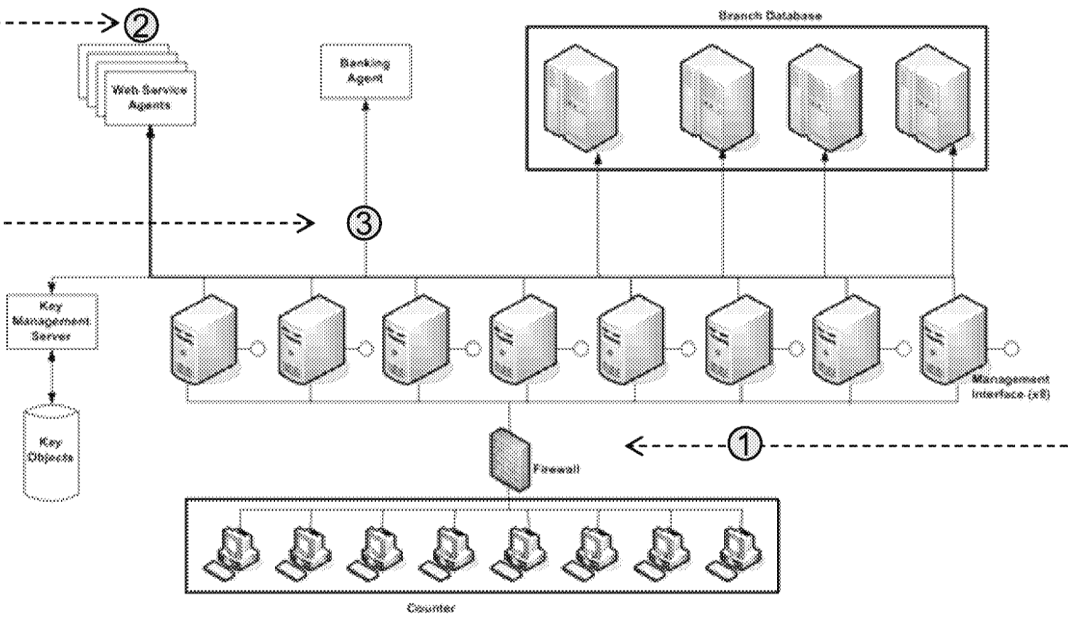
BRANCH ACCESS LAYER

The BAL serves as the interface between the counter and other data centre systems. It communicates with the following servers:

- Banking Agents – this is to perform banking transactions on behalf of a counter
- Web Service Agents – to allow a counter to communicate with other external systems.
- BRDB – the counter accesses the BRDB through the BAL servers
- Key Management Server – to retrieve any cryptographic keys or seeds that may be required for the BAL or the counter.
- Management Interface – A JMX based management server is used to control and monitor BAL servers.

Web Services

Service Router + Message Transformers



XML over HTTP (not SOAP)

Via Remote Services Interface (RSI) - not shown

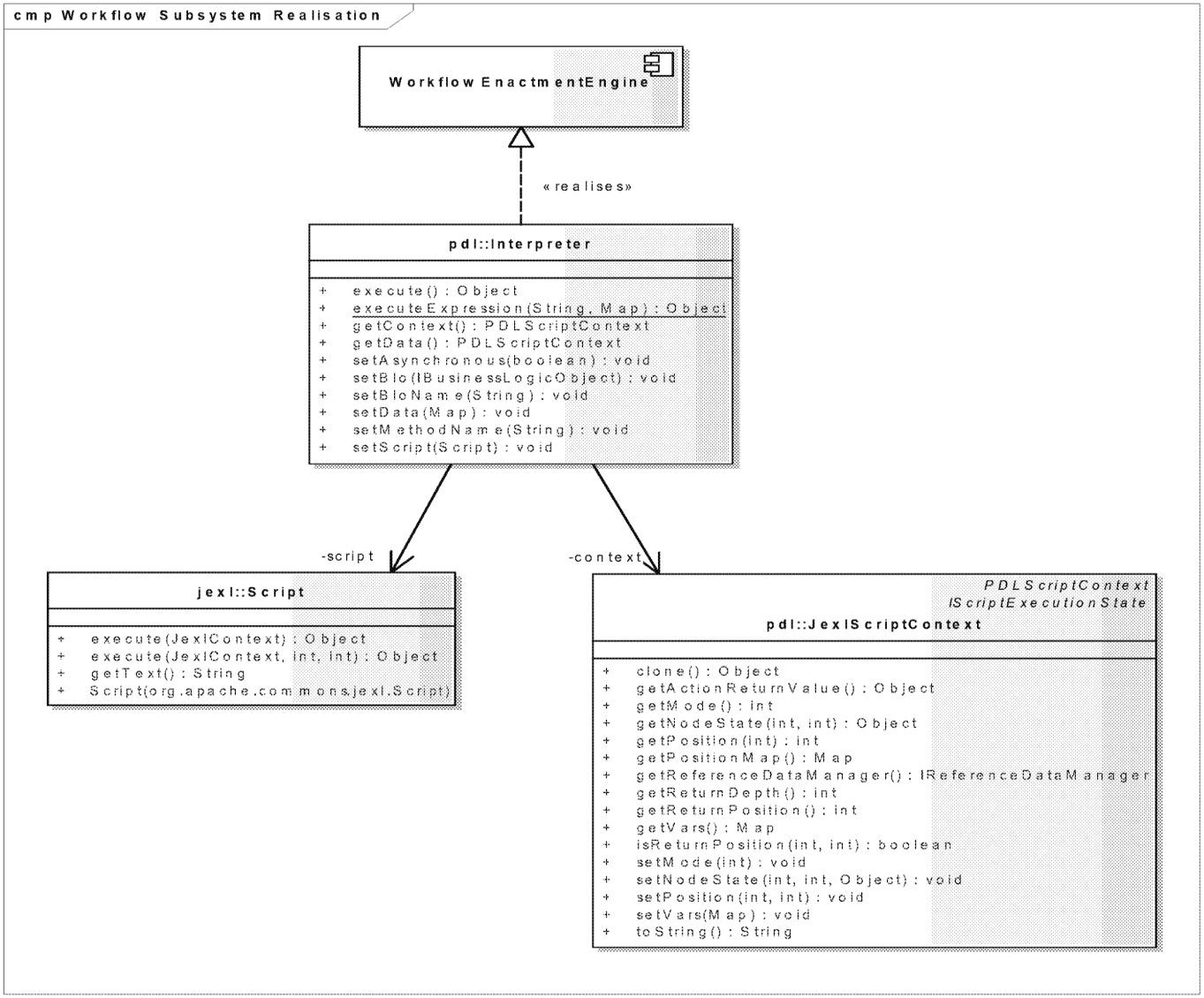
Figure 1 – BAL Environment

BAL servers are stateless, the number deployed can be easily altered so the eight servers shown in the diagram above should be used as a guide only, less or more servers could be deployed as is required.

From Branch Access Layer

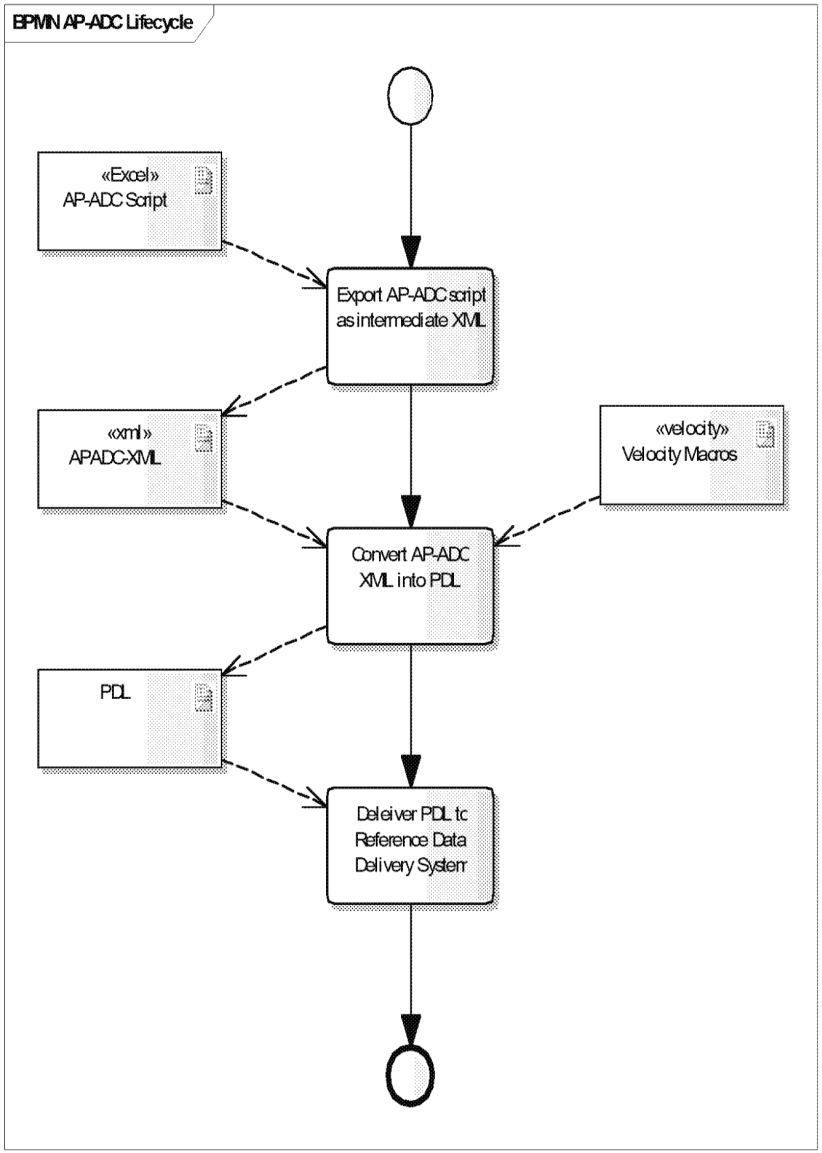
POL00137239

Workflow Subsystem



POL00137239

Delivering AP-ADC to the Counter

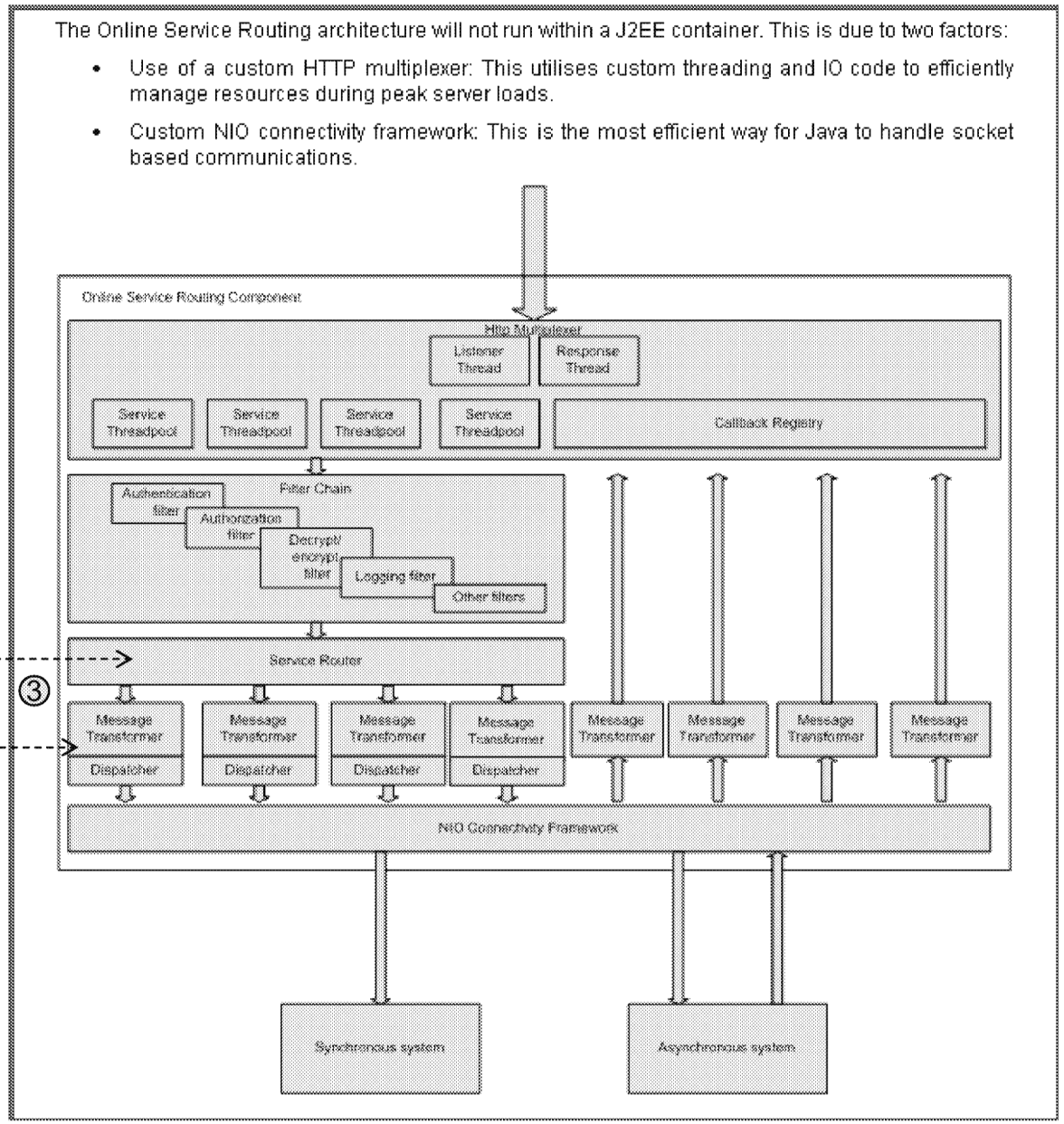


POL00137239

Online Service Router

The Online Service Routing architecture will not run within a J2EE container. This is due to two factors:

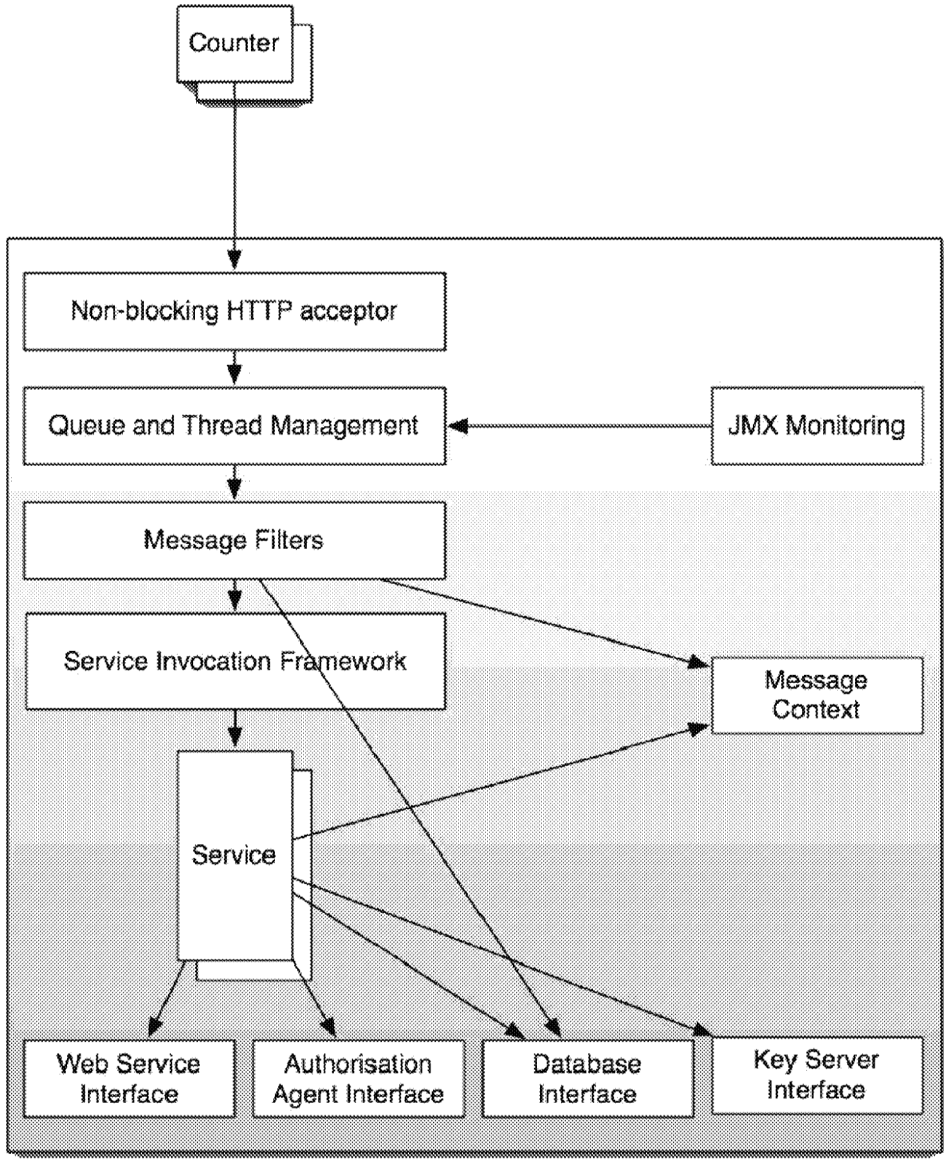
- Use of a custom HTTP multiplexer: This utilises custom threading and IO code to efficiently manage resources during peak server loads.
- Custom NIO connectivity framework: This is the most efficient way for Java to handle socket based communications.



From Branch Access Layer

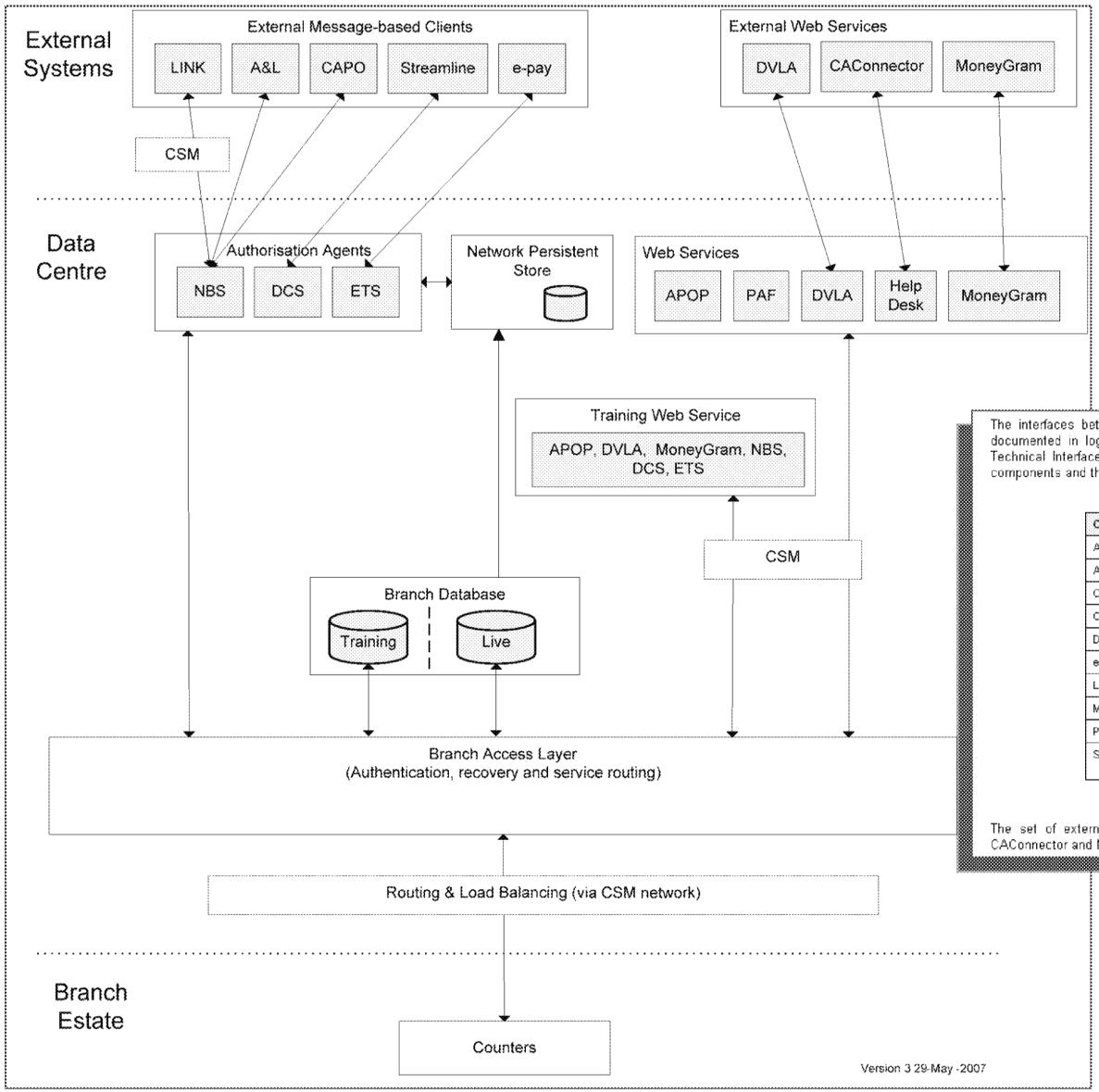
POL00137239

OSR Internals



POL00137239

Interfaces to External Systems



The interfaces between the online service components in the Data Centre and the external clients are documented in logical terms by an Application Interface Specification (AIS) and in physical terms by a Technical Interface Specification (TIS) – see Table 2 below. The interfaces between the online service components and the internal clients are defined by an AIS only.

Client	Service	Interface Type	AIS	TIS
A&L	NBS	ISO 8583	NB/IFS/026	NB/IFS/029
APOP	APOP	PL/SQL	AP/IFS/064	n/a
CAConnector	Help Desk	SOAP	TBA[2]	TBA[2]
CAPO	NBS	ISO 8583	NB/IFS/025	NB/IFS/027
DVLA	DVLA	HTTP	DV/IFS/001	DV/IFS/002
e-pay	ETS	APACS 30	ET/IFS/001	ET/IFS/003
LINK	NBS	ISO 8583	NB/IFS/024	NB/IFS/028
MoneyGram	MoneyGram	SOAP	AP/AIS/068	TI/IFS/009
PAF	PAF	API	QuickAddress	n/a
Streamline	DCS	APACS 30 and SMS	EF/IFS/002	EF/IFS/001

Table 2 - External Interface Specifications

The set of external clients falls into two groups: those that use synchronous[3] web protocols (DVLA, CAConnector and MoneyGram) and those that use asynchronous RAC message-based protocols.

Gartner Notes:

1. Does not show new "service hub" used to access external Web Services.
2. "CSM" is Cisco Network (now "ACI").

POL00137239

APPENDIX B

Gartner Research Papers

1. **The Enterprise Service Bus: Communication Backbone for SOA**
2. **SOA Applications Should Mix Client/Server, EDA and Conversational Patterns**
3. **Where to Use an Enterprise Service Bus and Why**
4. **Best Practice for Software Architecture: Intermediation**
5. **Open Source in ESB Suites, 2008**

POL00137239

Gartner Research

Publication Date: 3 May 2007 ID Number: G00143223

The Enterprise Service Bus: Communication Backbone for SOA

Roy W. Schulte

This research provides a definition and overview of enterprise service buses (ESBs). IT managers, architects and developers who are building, buying or contracting for SOA applications and services need to understand the role of ESBs for complementing basic communication software stacks and development tools.

Key Findings

- ESBs are a type of middleware that combines support for service-oriented architecture (SOA) and Web services with features from several older types of middleware. All ESBs implement SOA service binding, message-at-a-time communication and related features.
- ESBs support SOA applications better than traditional middleware because ESBs separate communication and integration logic from the business application logic.
- The industry has had some confusion about ESBs because the term is used to cover the core bus technology, ESB products and ESB design patterns.
- ESB products contain more than just plain SOA-oriented communication buses.

Recommendations

- Companies implementing SOA on a large scale should add ESB technology to their IT strategic plans and their technical architectures.
- See related Gartner research to understand ESB product features, packaging and management issues:
  - "Where to Use an Enterprise Service Bus and Why"
  - "Enterprise Service Bus Usage Scenarios and Product Categories"
  - "Succeeding With Multiple SOA Service Domains and Disparate ESBs"

POL00137239

ANALYSIS

The industry has been confused about ESBs, because the term covers the core service bus, ESB products and ESB design patterns.

- The core bus is a communication backbone, a set of middleware capabilities that is built into a variety of commercial products, including, but not limited to, those called "ESBs."
- Commercial products called "ESBs" contain features that include, but go beyond, the core communication bus.
- ESB design patterns are SOA application topographies that take advantage of the characteristics of ESB technology.

These are explained further in the next sections of this research.

The Core Service Bus Technology

The core service bus is a Web-services-capable communication subsystem that has the ability to support optional mediation functions, particularly for SOA applications, but not limited to SOA applications. To qualify as an ESB, middleware must:

- Implement synchronous and asynchronous program-to-program communication, moving messages between SOA service consumer modules and service provider modules at runtime. An ESB may also move files, database rows and other data.
- Support the fundamental Web and Web services standards, including Uniform Resource Identifiers, Extensible Markup Language (XML), SOAP and Web Services Description Language (WSDL). Almost all ESBs also move non-XML messages and data and offer additional proprietary communication protocols.
- Implement service binding to create associations between SOA consumer and provider modules.
- Have an architecture that enables it to apply optional intermediary functions to messages in flight. Mediation functions can be added to the core bus to, for example, inspect, validate, reroute, transform, enrich, log and track messages as they pass through.
- Support typed messages, that is, messages for which contents are explicitly defined and documented. This is necessary to implement many kinds of mediation.

ESBs support SOA applications better than message-oriented middleware (MOM), plain SOAP stacks and other traditional middleware, because ESBs provide a way to plug in optional value-added mediation and integration functions without having to implement a separate custom-built proxy server or wrapper. ESBs make it easier to offload communication and integration functions from the application developer, so developers can focus on the business logic. Some addressing and policy concerns, such as security, protocol choice and quality-of-service options, may be postponed to deployment time or runtime. This approach has major implications for development tools, not just the ESB.

Small or simple SOA applications can run fine without an ESB. For example, they can use point-to-point Web service connections supported by SOAP message handlers (SOAP/XML/HTTP stacks), plain old XML (POX) on HTTP or various forms of middleware. However, large, long-

POL00137239

living or frequently changing SOA service domains benefit from the features provided by an ESB. Further details on the technical characteristics of ESBs and where they are helpful are included in "Where to Use an Enterprise Service Bus and Why."

The core bus is rarely bought as a separate product. In almost all cases, companies acquire the bus as part of a larger product, such as embedded in an SOA middleware infrastructure product that contains many other features (see the next section) or in the operating system (for example, Microsoft's Windows Communication Foundation [WCF] is the service bus in Vista and the forthcoming Windows "Longhorn" server, and it can also run on Windows XP Service Pack 2 and Windows Server 2003).

ESB Products

ESB products are just one of many types of SOA infrastructure product. SOA infrastructure products are diverse in their packaging and labeling, but all contain the core SOA bus technology (described above) and other features. SOA infrastructure products can be sorted into three general categories, described in order of increasing levels of feature bundling:

- ESB products
- SOA platforms with development and presentation features
- Full SOA software stacks

1. ESB Products

ESB products are the most unbundled. Vendors tend to call their SOA infrastructure products "ESBs" if their capabilities are limited to communication and integration tasks. ESB products have the core bus (described above), and they typically include:

- Transformation
- A basic registry or name space that supports binding and some type of service virtualization (for example, using a service name as an alias to bind to an alternative service implementation)
- Content-based routing and basic service orchestration
- Security, including authentication and authorization, generally working in conjunction with external identity management, encryption and decryption services
- Optional adapters to files, database management systems (DBMSs), legacy platforms and packaged applications

They often also have:

- Message validation
- Some transaction management capabilities
- Message logging and auditing
- Protocol bridging
- Load balancing
- Failover

POL00137239

2. SOA Platforms With Development and Presentation Features

A product that is missing some of these features is still an ESB as long as it includes the core bus and some subset of the "typical" or "often also have" functions.

An SOA infrastructure product that provides additional development, presentation and monitoring features may be called an “SOA suite,” "service grid," “integration suite,” “Web services framework,” “composite application platform” or “service deployment platform.” Such products contain the core bus, some or all the "ESB product" features listed above, plus some or all the following:

- Process modeling, long-running business process management, process simulation and workflow services for human activities
- Repository or other metadata management tools
- Portal, Ajax, mobile and other presentation-related services
- Service monitoring and management capabilities for tracking availability, response times and other service-level issues
- Federated (“virtual”) database and data service support
- Business activity monitoring

3. Full SOA Software Stacks

A full software stack for SOA applications bundles even more. If an SOA infrastructure product contains many of the features described in the previous two categories, includes its own general-purpose application server and is offered with a comprehensive application development environment, it is more likely to be called an application platform suite (APS), business services fabric, enterprise services infrastructure, integrated service environment (ISE), or portal platform suite. The label business process management suite (BPMS) is used if process modeling, simulation, management and workflow are emphasized.

In the absence of consistent vendor packaging and naming decisions, all product labels in the SOA infrastructure market are somewhat arbitrary. We have outlined three general levels of packaging: ESB product, SOA platform with development and presentation features, and full SOA software stack with application server. However, commercial products are not consistently assigned to those categories. The same product may be called an ESB, service grid, enterprise service infrastructure, business service fabric, BPMS, APS or ISE, depending on who is talking and what they think the listener wants to hear.

In almost all cases, companies do not buy one SOA infrastructure product from one vendor; rather, they use products from several vendors (see "Enterprise Service Bus Usage Scenarios and Product Categories").

The ESB ("SOA Backplane") Pattern

Using an ESB (or an SOA infrastructure product that is a superset of an ESB) has important implications for SOA application architecture. An SOA application without ESB capabilities puts consumer modules in direct contact with service provider modules. The intelligence for finding the right service provider, orchestrating the flow, transforming messages and other functions is coded into the consumer or provider modules (if such features are needed). By contrast, SOA systems that use an ESB and related services to implement the concept of a "SOA backplane" offload many of these addressing and mediation functions to the ESB and service engines that are

POL00137239

plugged into the ESB. The programming model, techniques of service assembly and methods of implementing various policy and quality-of-service levels are different when using an SOA backplane. In this sense, an ESB implies a certain design pattern (the backplane) for the SOA application elements. The pattern relies on the participation of ESB software, a custom proxy server or something similar at runtime outside of the endpoint application modules themselves. ESBs also have inherent design-time, development-time and deployment-time implications, because developers use the development and administration tools associated with the ESB to create and configure SOA elements.

Microsoft's WCF is a core SOA bus, but not exactly an "ESB product," because it does not have embedded mediation functions. However, developers can use WCF to implement the ESB pattern by adding BizTalk Server or a third-party integration hub to mediate the communication (this revises earlier Gartner reports that did not fully explain the distinction between the core service bus, ESB products and the ESB pattern). Microsoft provides architectural guidance, patterns and practices for implementing ESBs and a set of reusable BizTalk Server and .NET components.

Background

Origins of Commercial ESB Products

Progress Software was the first to use the term "ESB" and the first vendor to ship (in 2002) a commercial ESB product with Web services support (Sonic XQ, renamed a year later to Sonic ESB). A few ESB-like, commercial middleware products existed prior to that time, including Fiorano's Tifosi, originally announced in 1998. It has evolved into Fiorano SOA Platform 2007, a full-blown SOA infrastructure with Web services, orchestration and other modern features. It is still actively marketed and supported. Candle's Roma, also announced in 1998, was another pioneering implementation of this type of product. Roma was later renamed Pathwai and then acquired by IBM when it bought Candle. Roma implemented the essential concepts of an ESB, including SOA communication over a messaging backbone, but it was designed prior to the introduction of Web service standards, and it had limited adoption.

Custom-Built ESBs

A number of large companies with far-sighted architects and sufficient technical resources built their own custom middleware backplanes during the 1990s. These generally were implemented as a set of libraries that acted as a super-application programming interface over a MOM product, usually IBM's WebSphere MQ (formerly MQSeries) or occasionally over an object request broker, such as Iona's Orbix, or a TP monitor, such as BEA's Tuxedo. Like a modern ESB, these backplanes insulated an SOA application from some aspects of communication, policy implementation, addressing, security, logging and correlating replies to requests. However, most did not have a formal service registry, integration features or metadata facilities for documenting message schemas.

After the introduction of XML in 1998 and SOAP v.1.1 in 2000, many of these custom backplanes were extended to support the Web services standards, essentially becoming home-grown ESBs. Some Gartner clients continue to develop custom or semi-custom ESBs and ESB-like backplanes for themselves. Developing a custom ESB from scratch is rarely practical, for the same reasons that developing a custom MOM or DBMS is rarely practical. Most companies do not have the expertise, the willingness to support it for the long term or the extreme business requirements that would make a custom ESB necessary. However, a sizable minority of companies are good candidates for semi-custom ESBs that can be assembled from a mix of open-source components (for example, open-source MOM, SOAP stacks and transformation engines), proprietary

POL00137239

components and some custom code. Interest in open-source SOA infrastructure is growing to serve the needs of these companies.

RECOMMENDED READING

"Five Principles of SOA in Business and IT"

"Enterprise Service Bus Usage Scenarios and Product Categories"

"Where to Use an Enterprise Service Bus and Why"

Acronym Key and Glossary Terms

APS application platform suite

BPMS business process management suite

DBMS database management system

ESB enterprise service bus

HTTP Hypertext Transfer Protocol

ISE integrated service environment

MOM message-oriented middleware

POX plain old XML

SOA service-oriented architecture

SOAP Simple Object Access Protocol

WCF Windows Communication Foundation

WSDL Web Services Description Language

XML Extensible Markup Language

POL00137239

REGIONAL HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, C[GRO]D
U.S.A.
GRO

European Headquarters
Tamesis
The Glanty
Egham
Surrey, TW20 9AW
UNITED KINGDOM
+GRO

Asia/Pacific Headquarters
Gartner Australasia Pty. Ltd.
Level 9, 141 Walker Street
North Sydney
New South Wales 2060
AUSTRALIA
GRO

Japan Headquarters
Gartner Japan Ltd.
Aobadai Hills, 6F
7-7, Aobadai, 4-chome
Meguro-ku, Tokyo 153-0042
JAPAN
GRO

Latin America Headquarters
Gartner do Brazil
Av. das Nações Unidas, 12551
9° andar—World Trade Center
GRO-São Paulo SP
BRAZIL
GRO

Gartner

POL00137239

Gartner Research

Publication Date: 13 February 2008 ID Number: G00155166

SOA Applications Should Mix Client/Server, EDA and Conversational Patterns

Roy W. Schulte

This research clarifies the difference between client/server and event-driven architecture (EDA) design patterns. Architects, developers and business analysts must understand when to use each pattern to make their applications effective.

Key Findings

- A service-oriented architecture (SOA) interface may implement a client/server interaction (typically a request/reply message pair), an EDA notification (one message) or a conversation (a sequence of messages).
- Large SOA systems and systems of systems are multifaceted — some interfaces are best implemented with client/server, others with EDA and a few should be conversational.
- EDA is appropriate for time-sensitive asynchronous processing — when time is less critical, data-centric solutions using files or databases are sufficient for asynchronous processing.
- Within the client/server and EDA patterns, there are other important variations, including Representational State Transfer (REST).

Recommendations

- Use SOA in all large, new composite applications and business processes to clarify the application structure, facilitate data and code sharing, and enable incremental maintenance and enhancements.
- Use client/server when components must collaborate to fulfill one business activity and the flow of control is determined within the client component.
- Use EDA when components can be run asynchronously and can be minimally coupled.

POL00137239

ANALYSIS

In 1996, when Gartner published its first reports on SOA, we described it as an implementation of client/server and didn't mention EDA. Some architects still think only of request/reply client/server patterns when they design new SOA applications; however, most large systems and complex business processes have asynchronous aspects that are not well-addressed by client/server. These aspects may be addressed through EDA. This research describes the trade-offs between these patterns.

Client/Server

In a client/server relationship, the client component sends a request message to a server component, which responds by performing a function. The communication model is usually request/reply, although in rare situations, a reply isn't needed.

Client and server are roles. A component ("A") is a client because it sends a request, and another component ("B") is a server because it responds to the request. Client/server relationships have been common for decades, particularly for interactions between an application program and a system utility. For example, an application invokes a print server to put a file out on a printer. However, in SOA applications, the server performs a business application function, rather than a system function.

One example is a client/server application that captures a customer address change from a Web page, validates the ZIP code, then updates its local database. The mainline portion of the application that accepts the end-user Web input is client A. It uses a client/server relationship to invoke an SOA service supplied by component B to validate the street address against the ZIP code. Component B could run on another computer in another department or even in another company miles away. After receiving the reply from B, client A regains control and writes the new address into the database.

Characteristics of client/server relationships:

- The client directs the flow of control by specifying which server to invoke and when.
- The client delegates some of its work to the server and depends on the server. A server can, in turn, act as a client to further delegate some work to another component, and so on.
- The client connects to the server using a find-bind-invoke sequence. The coupling is "loose" if it works indirectly — for example, in Web services, bind and invoke are combined into one operation.

Client/server has many variations:

- The client usually suspends work until the reply is received; however, in some cases, the client continues working after sending a request, and the reply is returned at a later time.
- Client/server usually relates exactly one client and one server at runtime (just as a call statement invokes one procedure, never zero or two). Client A can have a relationship with multiple servers in succession, and server B can serve multiple clients in succession, but each relationship is one-to-one.

POL00137239

- However, a client request may be sent to an intermediary, which then relays it to multiple potential servers (this is "publish/reply"). The intermediary can select one reply (for example, the first one) to return to the client or the client may receive multiple replies.
- Some conversational interactions can be considered types of client/server patterns, because a component performs a function under the direction of another. However, state is maintained in the server between messages, so it's more tightly bound than in request/reply client/server relationships, where the server is stateless after sending the reply.
- REST is a unique form of client/server with characteristics that are different from those of traditional procedure calls. REST can also be used to implement EDA. (A full analysis of REST and conversational relationships is beyond the scope of this research.)

Most developers are comfortable with client/server, because it resembles the way subroutines are invoked in a program. However, most business situations have asynchronous aspects that are not addressed well through client/server.

Client/Server Limitations

Consider again an address change application. A company may have a dozen or more application systems (C through N) that maintain customer address data. (Whether it's smart to have multiple, partially redundant databases is irrelevant; it's a common situation.) When application A captures the address change, it must notify systems C through N. The address change is a business event — a meaningful change in the state of something relevant to the business.

It is possible, although intolerably clumsy, to build a client/server solution for this scenario. The original client A would invoke an address change function in each of the other systems. Client A calls application C, sending the new address in the request message. C returns an acknowledgement. A then calls D with a similar request, and so forth. After a dozen client/server interactions, systems C through N would have posted the change to their respective databases. However, if there's a communication problem, or if any system isn't running, A must perform an error recovery procedure. This arrangement makes A complicated and unnecessarily coupled to systems C through N. Adding or changing a consumer application requires changing, recompiling, retesting and redeploying A. Although it's sensible to have A depend on function B, the ZIP code validation service, it makes no sense for A to depend on the otherwise-autonomous applications C through N.

Batch and Polled Event Processing

The traditional way of handling this situation is to have A write the new address data to a file or database to be picked up later by the other systems. This avoids the complexity and runtime dependencies of a dozen client/server interactions. C through N typically update their address records in scheduled batch jobs. However, they could be set up to poll a file or database every few minutes or hours to pick up new addresses. The address change is a business event, so either arrangement (batched or polling) can be described as event processing (but not EDA). If the business can tolerate delays in transferring the updates, these data-centric, non-SOA patterns are satisfactory.

POL00137239

EDA

If the business requires up-to-the-minute data consistency, a better solution is to disseminate the new addresses in messages. When event data is transmitted in a message, the combination is called an "event notification." EDA is defined as an architectural style in which one or more components of a system execute in response to receiving one or more notifications (see "Tutorial for EDA and How It Relates to SOA"). In our example, A through N are considered to be components of a system of systems. System A could send one message containing a new address to an intermediary — for example, a message-oriented middleware (MOM) product — that delivers it to 12 consumers, systems C through N. The relationship between A and B is client/server, and the relationships between A and C through N are EDA.

Like client/server, and unlike file and database solutions, EDA is built on program-to-program communication. EDA event consumers are somewhat similar to client/server clients, and EDA event sources are somewhat similar to client/server servers, but with some key differences:

- EDA notifications are pushed by the event source, not pulled by the event consumer. The event source determines when the message is sent. An event consumer cannot predict when it will receive a notification, so it must be implemented with an event-capable, asynchronous communication mechanism. By contrast, client/server clients typically use synchronous procedure calls.
- An event consumer doesn't pass parameters to an event source. The consumer does not know what the event source is doing; it only knows that the source will emit a notification when an event occurs. By contrast, client/server clients send a document or other parameter set to the service provider to convey instructions related to the current instance of work (for example, client A passes each new customer address to server B).
- Event sources do not depend on event consumers. If all consumers stop running, the source still runs. The events that it emits can be dropped, or MOM can save them in a queue for delivery at a later time, if the business requires it. By contrast, a client/server server will not run unless it has been invoked by a client.

EDA minimally couples the source and consumer, and makes it easy to modify them independently. As long as the notification message stays the same and the change is compatible with the business requirements, a developer can change or add event consumers without changing the source component. Similarly, the event source can be changed without changing the consumer(s), as long as the source emits the same notifications, and the logic of the business process is not impaired. In some cases, the person developing an event source may assume that a downstream event consumer will perform certain activities in a business process, but there's nothing in the notification message or software interface that makes this explicit or "hard wired."

EDA has numerous variations:

- Each notification is typically available to multiple event consumers (a one-to-many relationship). The source may emit ("publish") a notification, and a subscription manager in a middleware intermediary may deliver a copy to all consumers who have registered an interest in (have "subscribed to") that type of notification.
- However, EDA can also be implemented with direct, one-to-one communication. EDA doesn't need to use publish-and-subscribe, although it's valuable when there are multiple sources and consumers or they change frequently.
- A sophisticated event consumer can analyze multiple incoming notifications using rules to detect patterns that indicate situations of interest to the business. This type of

POL00137239

computing, called complex-event processing (CEP), is the basis for business activity monitoring and similar applications. (The topic of CEP is beyond the scope of this research.)

Using Middleware

EDA and client/server can be implemented with or without a middleware intermediary. If the EDA or client/server application is simple, plain protocols — such as HTTP or SOAP/HTTP (Web services) — without middleware may be sufficient. However, if the relationship or communication patterns are complex, it's generally better to use off-the-shelf middleware, rather than coding equivalent features by hand in the application components. For example, MOM is a natural fit for EDA, because most MOM products support publish/subscribe, point-to-point communication and persistent queuing. However, the use of MOM does not define EDA. With enough effort, these functions could be built into the event source and consumer.

MOM can also be used to implement request/reply client/server relationships, which is a relatively common practice (although most client/server uses alternative communication mechanisms, rather than MOM). Other intermediaries, such as enterprise service buses, integration brokers and business process management tools, may also be useful for client/server, EDA and mixed applications that require address redirection, transformation, content-based routing or process management.

When to Use Client/Server or EDA

Architects, analysts and software engineers make many decisions when designing SOA applications. For example, they must decide whether to put a function, such as ZIP code validation, in a separate component (B) or embed it in the main line (A). They must also decide whether an application system (A) should be combined with one or more of the other application systems (C through N).

In our example, component B (ZIP code validation) is a sensible SOA service, because:

- It has a reasonable granularity, can be isolated from other parts of procedure A and an interface can be clearly defined
- There may be a desire to host B on a different computer, perhaps owned and operated by a business unit other than the one that hosts A
- Function B may be shared by multiple, disparate consumer applications

Similarly, the purpose of application A in our example is sufficiently distinct from that of C through N, so that each should be deployed as its own system. For each relationship and interface in this system of systems, developers had to select between client/server and EDA.

The client/server pattern is appropriate for relationships like that between A and B when:

- The server (B) performs a subset of a larger activity controlled by the client (A)
- The client depends on the server to perform a function and cannot finish its work without the server's reply
- The server requires input instructions (in this case, address data) from the client to know what to do

The EDA pattern is appropriate in relationships where the components (in this example, whole application systems A and C through N) are largely autonomous:

POL00137239

- The event source reacts to external stimuli (in this case, end-user input regarding new addresses), rather than instructions from downstream consumers
- The only commonality among the components is an interest in data about the same event (the address change)

Although our example is simplistic, it represents real-world scenarios in critical ways. Client/server and EDA are complementary, and they should be used to address different aspects of work in a system or in a system of systems. In most projects, client/server relationships will outnumber EDA relationships. Client/server tends to apply to one application system, although it is also used among separate systems for interactive, composite applications (using wrappers for legacy systems where necessary). EDA relationships often apply to coarser-grained relationships, such as those between separate application systems and separate companies, although EDA is also used in applications for fine-grained, asynchronous, minimally coupled processing.

RECOMMENDED READING

"Tutorial for EDA and How It Relates to SOA"

"Advanced SOA for Advanced Enterprise Projects"

"Understanding and Applying the Design Differences Between WS-* Based Architecture and Web-Oriented Architecture"

"Applying WS-* Based Web Services and WOA Standards to Enterprise Application-to-Application Interoperability Challenges"

Acronym Key and Glossary Terms

CEP complex-event processing

EDA event-driven architecture

MOM message-oriented middleware

REST Representational State Transfer

SOA service-oriented architecture

POL00137239

REGIONAL HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT GRO
U.S.A.
GRO

European Headquarters
Tamesis
The Glanty
Egham
Surrey, TW20 9AW
UNITED KINGDOM
GRO

Asia/Pacific Headquarters
Gartner Australasia Pty. Ltd.
Level 9, 141 Walker Street
North Sydney
New South Wales 2060
AUSTRALIA
GRO

Japan Headquarters
Gartner Japan Ltd.
Aobadai Hills, 6F
7-7, Aobadai, 4-chome
Meguro-ku, Tokyo 153-0042
JAPAN
GRO

Latin America Headquarters
Gartner do Brazil
Av. das Nações Unidas, 12551
9° andar—World Trade Center
04578-903—São Paulo SP
BRAZIL
GRO

Gartner

POL00137239

Gartner Research

Publication Date: 3 May 2007 ID Number: G00143292

Where to Use an Enterprise Service Bus and Why

Roy W. Schulte

This research describes the features and functions of an enterprise service bus (ESB) and compares and contrasts them with other forms of middleware and basic communication protocols. Architects and developers who are building service-oriented architecture (SOA) applications should understand ESBs so that they know where to use them and where they are unnecessary.

Key Findings

- For most new SOA projects, the choice of communication infrastructure now comes down to using simple Web service stacks, plain old XML (POX) on HTTP or an ESB.
- ESBs improve the quality of program-to-program communication, make it easier to share SOA services and make service versioning and changes to SOA interfaces easier to implement.
- ESBs are often used to present portions of legacy and purchased non-SOA applications as SOA services. However, ESBs are also relevant in large-scale or long-living SOA service domains where there are no legacy non-SOA applications.
- Since their inception, ESBs helped enable the separation of communication and integration logic from the application business logic. This approach is now being expanded to include more policy-related issues regarding security, choice of protocols and quality of service.

Recommendations

- Avoid hard-coding the identity (for example, the uniform resource indicator [URI]) of the service providers into the consumers in any large, evolving SOA system.
- Use an ESB to offload communication and integration functions from the consumer and provider elements in all large domains (more than 20 services) so that composite SOA applications and business processes can be modified more quickly and easily.
- There is no need for an ESB in simple, small (fewer than 20 services) SOA applications, particularly where all the interactions are request/reply, interfaces are slow to change, there is no integration with packaged or legacy applications, and the whole application is built by one disciplined development team.

POL00137239

ANALYSIS

An ESB is a communication and mediation layer that connects service consumers and service providers in SOA scenarios and situations that mix SOA and other architecture styles. However, not all SOA scenarios require an ESB. Good SOA applications have been built since the 1990s without ESBs, and many, especially prior to 2002, did not even use the Web or Web services. SOA can be implemented with many different technologies — a choice with a long history (see "Middleware for Service-Oriented Architectures").

A few SOA applications are still being implemented on object request brokers (ORBs), transaction processing (TP) monitors and message-oriented middleware (MOM), but for most developers, the choice now comes down to using simple Web services (SOAP on HTTP), POX or an ESB. The first two of these, SOAP and POX, are widely available and free, bundled into Web servers, application servers, portal products and operating systems. ESB technology is available not only in products called ESBs, but also embedded in other types of SOA infrastructure products (see "The Enterprise Service Bus: Communication Backbone for SOA").

To understand where ESBs should be used, architects should consider four issues:

- Multiple communication patterns
- Intelligent addressing, routing and orchestration
- Mediation
- Complementing application platforms

Multiple Communication Patterns

ESBs are useful where the applications will use a mix of communication patterns:

- All ESBs support one-way messages and two-way request/reply exchanges.
- Almost all ESBs also support message queuing (store-and-forward) and publish-and-subscribe (pub/sub).

If the applications will *only* use request/reply, then plain HTTP or a simple SOAP stack may be sufficient (unless other ESB features are needed). However, HTTP does not supply reliable messaging, queuing or pub/sub, although a limited form of pub/sub is available in protocols such as RSS and Atom. SOAP stacks (outside of ESBs) are beginning to support WS-Reliable Messaging (delivery confirmation for one-way messages), but SOAP standards do not cover queuing, and SOAP-based pub/sub specifications (such as WS-EventNotification) are still being debated. MOM can support all the communication patterns offered by ESBs (indeed, virtually all ESBs embed MOM), but plain MOM lacks other features helpful for SOA and is less aligned with industry standards.

An ESB may also move files, database records and other types of data used in non-SOA communication. Many ESBs are adding explicit support for Representational State Transfer (REST) and conversational communication. Microsoft's Windows Communication Foundation (WCF) and the Service Component Architecture (SCA), a new design approach promoted by the Open SOA Collaboration (www.osoa.org), include REST and conversations in their specifications. Many ESB vendors have pledged to support SCA (see "Service Component Architecture Is a Winner in the Quest to Establish a Common Notation for SOA").

POL00137239

Intelligent Addressing, Routing and Orchestration

ESBs offload some addressing, routing and orchestration tasks from the application, enabling the consumer and provider elements to be simpler. Communication paths are "soft wired" into the ESB, where they can be changed at deployment or runtime, rather than being "hard wired" into the application code at development time. There are three aspects to this:

- **Service virtualization —** ESBs bind each service consumer with a suitable service provider at runtime. ESBs use a name space or registry, whether embedded or external, to resolve the service reference to a specific implementation (element). The registry may be based on UDDI, or it may be entirely proprietary. All ESBs make it simple to substitute an alternate provider at deployment time, and most also enable runtime substitution. The developer of the consumer does not have to know the URI of the provider, because the ESB redirects the request, for example, by using a service identifier as an alias.
- **Rule-based routing —** Most ESBs support content-based routing. Routing rules may be written in JavaScript, XPath or a third-generation language, or they may be specified with a graphical development tool (which may generate XPath or another language). Many ESBs also use topic names, message properties or queue names to direct asynchronous, one-way messages (often leveraging MOM features built into the ESB).
- **Orchestration —** Most ESB products have a facility that orchestrates the flow of a composite SOA application or a multistep business process. The sequence of the services to be executed and conditional routing rules are usually specified through a graphical tool. The flow can be reconfigured with few or no changes in the consumers. Some early ESBs did not have orchestration (it is not definitional to an ESB), but most commercial ESBs now do, as a standard feature or an extra-cost option. Orchestration may be implemented in an ESB hub (for example, in a BPEL server), in distributed adapters (for example, as itinerary-based routing) or both.

HTTP, plain SOAP, TP monitors, ORBs and MOM do not have intelligent routing or orchestration capabilities. Aside from ESBs, only integration tools, such as programmatic integration servers, integration brokers and similar products, have this. Any large, continuously evolving SOA system should avoid hard-wiring the identity (for example, URI) of the service providers in the consumers. A number of companies have coded their own simple mechanism for virtualizing service identifiers so that they do not use an ESB for this purpose, but few companies build their own rule-based routing or orchestration tools.

Mediation

ESBs move messages between SOA elements so ESBs are in an ideal position to modify the messages or otherwise add value as the messages pass through. By definition, an ESB must have a mechanism that enables mediation to be performed, although an ESB is not required to supply any particular mediation functions. Examples of common mediation functions include:

- Message validation
- Transformation
- Protocol bridging (for example where one element uses SOAP v 1.1 and another uses SOAP v.1.2 or a MOM)
- Message logging and auditing

POL00137239

- Security, including authentication and authorization
- Service virtualization, rule-based routing and orchestration

Many of these functions require reading the message contents, which is why ESBs must support typed messages (the message attributes are explicitly documented). Most ESBs have development-time tools that read WSDL files and XML Schema Definitions and then generate metadata used by the ESB at runtime. Many ESBs can also import metadata from database catalogs and other sources. A full repository is not part of an ESB and is not necessary for the ESB at runtime, but companies engaged in large-scale SOA programs should have one (see "When to Use Metadata Repositories, Registries or Both").

Transformation is particularly important because it makes it easier to change services and interfaces. A new version of a service with additional functions can be installed and used by new consumers without disrupting consumers of the previous interface, because the ESB can transform the new messages into the older formats that have the message attributes used by earlier consumers.

Basic protocols, such as HTTP and SOAP, and traditional middleware, including ORBs, TP monitors and MOM, have no mediation capabilities — they pass messages through unchanged. A developer can write a custom wrapper or manually insert a proxy server in the middle of a message flow to intercept and mediate a message, but it would not be a native part of the communication infrastructure as it is with an ESB, integration suite and certain other integration tools.

Complementing Application Platforms

Many ESBs support certain functions that are also performed by high-end application servers and TP monitors, particularly:

- Load balancing
- Failover
- Transaction management

The ESB implements load balancing and failover by rerouting messages to an alternative server. This could potentially involve dissimilar application servers, although, in practice, load balancing and failover almost always use the same server technology, because the alternative service provider element must be interchangeable with the original. Transaction management includes synchronizing with other resource managers, such as database management systems (DBMSs).

Projects that have made a prior decision to use a high-end (for example, JEE) application server can get these functions from their application server or optionally from an ESB. However, an ESB may affect the choice of platform because it also supplies these functions to a simple Java Standard Edition platform or a plain operating system process, potentially making the high-end application server unnecessary in situations where the only motivation for the high-end platform was scalability or reliability.

Some types of SOA infrastructure products include ESBs and application servers. Even some commercial products originally called "ESBs" now have their own limited or full-blown application server, going beyond the original understanding of an ESB to become a larger form of SOA infrastructure. In this type of product, the embedded application server can be the container (host environment) for the service provider elements, while the ESB aspects are used to connect into external application servers that host other services(see "Enterprise Service Bus Usage Scenarios and Product Categories").

POL00137239

Role of an ESB

The role of an ESB is roughly analogous to the role of a DBMS, although ESBs do program-to-program communication, whereas DBMSs support program-to-database interactions. DBMSs offload data management tasks from the application by performing data navigation (such as joins), transactional integrity, concurrency control, backup, recovery, load balancing, caching and other functions, thereby improving the overall quality of data management. DBMSs also make it practical to share data among multiple applications. Occasional changes in data models are easier to implement. Similarly, ESBs improve the quality of program-to-program communication, make it easier to share SOA services among multiple consumer applications and make occasional changes to SOA interfaces easier to implement. To the extent that these capabilities were previously coded into the application, an ESB (such as a DBMS) simplifies application development. However, most applications simply existed without these features in the past, so the ESB's value in flexibility and quality is more notable than the savings in application code.

Since their inception, ESBs have encouraged the separation of communication and integration logic from business logic. This principle is now being applied more extensively as ESBs begin to implement the new SCA and WCF models for SOA architecture. The goal is to abstract policy-related issues by expressing them declaratively at development time and postponing the implementation of the choices to deployment or runtime. This applies specifically to policies regarding security, choice of protocols and quality of service. This evolution (and SCA and WCF in general) involves not only the ESB, but SOA application design practices and SOA development tools.

Where to Use an ESB

Large (more than 20 services), demanding or frequently modified SOA applications benefit most from ESBs. The factors that tend to promote the use of an ESB include:

- Business requirements that call for a mix of protocols and communication patterns, including request/reply, one-way messages, message queuing and pub/sub
- Applications composed of elements that run on a mix of heterogeneous application servers and operating systems, requiring compatibility across platforms or protocol conversion
- Processes that change fairly frequently, or those with complex routing rules where business analysts want to change the flow without changing the consumers or service providers
- SOA scenarios where consumer or service provider elements are added, modified or moved fairly often
- Applications with more than 20 SOA services, because they are likely to change often
- Integration scenarios involving packaged and legacy applications

ESBs are often used to present portions of legacy and purchased non-SOA applications as SOA services. Developers do not have to write a custom wrapper or design a proxy server because those functions are implemented within the ESB, its adapters or servers that are plugged into the ESB. However ESBs are not just applicable to these classical integration scenarios. Any large-scale or long-living set of SOA services is an appropriate target for an ESB because of the “organic” nature of SOA. New business processes, consumers and providers are frequently added, and older ones are periodically modified or retired. The ability of an ESB to simplify

POL00137239

ongoing changes in interfaces and processes is relevant, even where there are no legacy non-SOA applications.

Versioning is a major challenge in the operation of an SOA service domain, so the ability of an ESB to transform messages can be a key benefit. In the future, there will be fewer legacy non-SOA applications, but more legacy SOA services, so there will be less need for technical gateways, but an ongoing need for the other mediations that ESBs offer.

An ESB is overkill for small, simple, static SOA applications, particularly where:

- All the communication is request/reply or "best efforts" one-way messages (delivery need not be ensured).
- Navigation and routing logic is simple and can be embedded in the consumers.
- There are relatively few services (fewer than 20).
- Consumers and services do not change often.
- All the applications are designed by one team or closely cooperating teams that exchange metadata and agree on consistent interface definitions.

SOA proof-of-concept projects and other SOA applications with a limited scope and life span generally do not need an ESB.

RECOMMENDED READING

"Middleware for Service-Oriented Architectures"

"Service Component Architecture Is a Winner in the Quest to Establish a Common Notation for SOA"

"When to Use Metadata Repositories, Registries or Both"

"The Enterprise Service Bus: Communication Backbone for SOA"

"Enterprise Service Bus Usage Scenarios and Product Categories"

Acronym Key and Glossary Terms

BPEL Business Process Execution Language

DBMS database management system

ESB enterprise service bus

HTTP Hypertext Transfer Protocol

MOM message-oriented middleware

ORB object request broker

POX plain old XML

REST Representational State Transfer

RSS Really Simple Syndication

SCA Service Component Architecture

POL00137239

SOA	service-oriented architecture
SOAP	Simple Object Access Protocol
TP	transaction processing
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
WCF	Windows Communication Foundation
WSDL	Web Services Description Language
XML	Extensible Markup Language

REGIONAL HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT GRO
U.S.A.
GRO

European Headquarters
Tamesis
The Glanty
Egham
Surrey, TW20 9AW
UNITED KINGDOM
GRO

Asia/Pacific Headquarters
Gartner Australasia Pty. Ltd.
Level 9, 141 Walker Street
North Sydney
New South Wales 2060
AUSTRALIA
GRO

Japan Headquarters
Gartner Japan Ltd.
Aobadai Hills, 6F
7-7, Aobadai, 4-chome
Meguro-ku, Tokyo 153-0042
JAPAN
GRO

Latin America Headquarters
Gartner do Brazil
Av. das Nações Unidas, 12551
9° andar—World Trade Center
GRO –São Paulo SP
BRAZIL
GRO

Gartner

POL00137239

Gartner Research

Publication Date: 17 July 2008 ID Number: G00149891

Best Practice for Software Architecture: Intermediation

Yefim V. Natis

Intermediation in the design of business applications means that software elements (clients, service implementations and event handlers) interact indirectly, through intermediaries. The resulting improvement in manageability and extensibility can make business applications more durable, scalable and agile. However, achieving this means learning new skills in architecture and programming, and sometimes using new tools — a potentially costly first step.

Key Findings

- Business systems designed with intermediation are more agile, thus will last longer and enable greater extensibility, scalability and innovation.
- Software scenarios that require tight coupling between software elements for atomistic or other reasons generally shouldn't use intermediation.
- Most interactions, especially under the service-oriented architecture (SOA) model, can be intermediated, and the IT environment likely will benefit.
- Most IT organizations have the technical means for intermediated communications (queuing and publish-and-subscribe [pub-sub], distributed caching, enterprise service bus [ESB] and other middleware). Therefore, intermediation is available to most projects at the early planning stages or at the programming stage.

Recommendations

- Software designers should use intermediation for most business systems.
- IT managers should ensure proper training in intermediated communications for IT architects, project leaders and engineering staff.
- Technology evaluators should include intermediation support in selection criteria for platforms, tools, packaged applications and software-as-a-service (SaaS) contracts.
- IT planners and architects should invest in understanding the advantages and limitations of intermediation to avoid underpowering their systems or overstretching middleware by making wrong choices.
- It organizations with little or no experience in intermediation should plan the experimental use of intermediation in a less-critical, real-world project, such as a pilot program.

POL00137239

STRATEGIC PLANNING ASSUMPTION(S)

Through 2013, intermediation as a core design principle will result in applications of the best scalability, extensibility and longevity in the software industry.

ANALYSIS

A fundamental effect of SOA is the partitioning of business application software into smaller building blocks: services, event handlers or similar software elements. In most cases, interactions among services and between clients and services are in the form of direct request/reply. However, industry experience shows that intermediated (indirect) communication among software elements can deliver substantially more benefits than the traditional direct communication. Only advanced software architecture teams can deploy intermediate communication as the core architecture of their applications. As the demand for the high-end characteristics of SOA applications reaches the mainstream, intermediated communication likely will become a mainstream best practice of advanced SOA.

The difference between direct and indirect communication is:

- Direct communication is communication where the requestor (A) makes a direct call to the service (B) by its name or alias.
- Indirect (intermediated) communication is when the originator (A) does not name or attempt to contact the target (B) but communicates with an intermediate third party (X) — a queue, a middleware product, a caching mechanism, a database or another software intermediary. It becomes the job of the intermediary to identify and contact the target (B), or (in simpler scenarios) the target may be actively polling the intermediary for work requests.

All software is multilayered, and modularity and interconnectivity exist (largely independently) at each level. This research is exclusively about the design of business applications. The communicating software elements in this example are SOA services and clients, event-driven architecture (EDA) event handlers and the like. Thus, the intermediaries are directly visible and addressable by the business software (a queue, a middleware product's application programming interface [API], such as an ESB, a caching mechanism, a database or another software intermediary). It is possible that what is, for example, direct communication at the application level is implemented at the level below through the intermediated model (messaging or similar), but this fact is unknown at the application level and, therefore, is of no concern. Mixing the levels when considering these matters will create confusion (see Note 1).

The choice to use intermediation in a software system can be made at the early stage of application design and planning or at the later time of programming.

- Early decision, in this respect, ensures greater consistency at programming time and greater cohesiveness among the different stages of application planning and the development processes.
- Late decision has limited scope and can result in the same project ending up with multiple models and technologies of intermediation chosen by different technical contributors.

Regardless of the stage that intermediation is considered, a specific intermediary must be established for all software or design elements. This choice is best-informed if multiple scenarios

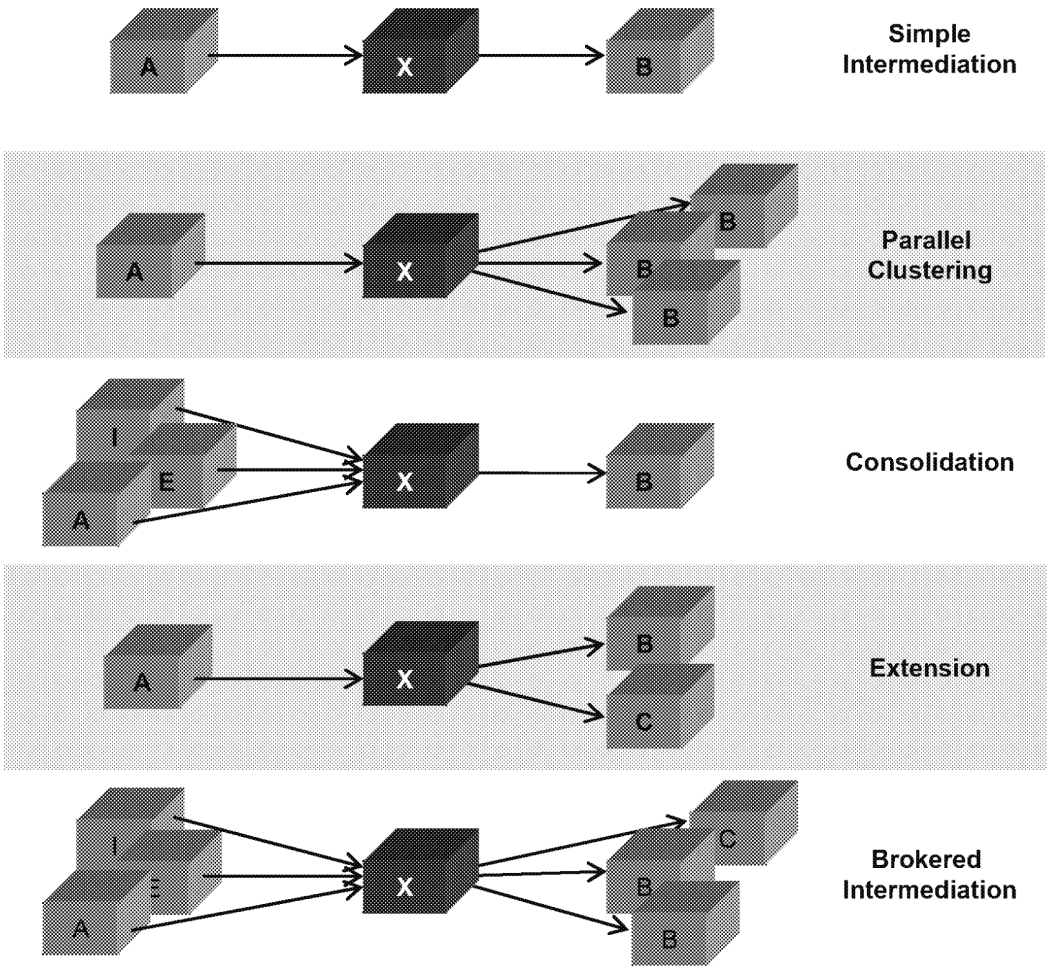
POL00137239

are considered. When a programmer chooses an intermediary, the program typically is aware only of the requirements of his or her part of the larger project. For all these reasons, we recommend that the decision to use intermediation and its specific rendition be made by software architects based on the specifics of the business design of the application. Thus, application software architects should be the key intermediation experts in the IT organization.

The Scenarios of Intermediation

Simple intermediation begins with putting a simple intermediary (a queue or another) between the two communicating software elements. However, advanced scenarios include clustering, consolidation, extension and the all-inclusive brokered intermediation, and can engage a large number of participating software elements (see Figure 1).

Figure 1. Use Scenarios for Application-Level Intermediation



Source: Gartner (July 2008)

POL00137239

Advanced scenarios require an "intelligent" intermediary (a middleware product, such as an ESB or another form of integration broker) to manage the passing messages and to apply processing rules. Some more-advanced intermediaries also may be context-aware. This variety of add-on intelligence, and the potential power of intermediation, is a direct result of decoupling the software elements in intermediated communications. These are the typical use scenarios for intermediate communication:

- Simple intermediation: A gives the input data to the intermediary, and B gets it from the intermediary. The same result can be accomplished by a one-way request-reply with a greater degree of coupling between A and B.
- Parallel clustering: Multiple instances of B are begun to take work from the intermediary and process it in several parallel streams to maintain the service-level agreement's (SLA's) response time by increasing the system's throughput.
- Consolidation: B processes requests from multiple related sources, consolidating processing and improving resource use for relatively infrequent requests.
- Extension: A new service (C) is added to process the request, enabling an unintrusive extension of the application's functionality. Monitoring functionality, including business activity monitoring (BAM), is a great beneficiary of this scenario. This requires an intelligent intermediary; simple intermediary storage will not do, because the multiple targets and passing messages must be managed.
- Brokered intermediation: Multiple scenarios are combined in one intermediation environment. Multiple requests and services are interconnected and managed via an intermediation broker. Business rules can be added to the broker to add value to the incoming requests before forwarding them to the appropriate targets. Advanced brokers may also be context-aware. This, too, requires an intelligent intermediary, because management and brokering require the processing of rules and other logic.

The Benefits of Intermediation

The indirect decoupled nature of intermediated communication is the cause of its distinct benefits:

- The intermediary keeps track of messages that are passing through, enabling tracking, interception, management and post-transaction analysis.
- The originator is not blocked while the service executes, nor is it dependent on whether the service is available to execute at the time when the request is issued.
- The intermediary, by its nature, enables centralized control, including dynamic optimization of traffic by manipulating priorities and resource allocations for different channels, depending on configuration and context.
- Load balancing, clustering and fault tolerance are natural attributes of intermediated communication when an "intelligent" intermediary is present.
- Extensibility by simply adding new services (listeners) to the process in a new way; some recognized requests can be unintrusive for applications and transactions.
- Extensibility by adding more points of origination of already recognized requests can be entirely unintrusive for applications and transactions.
- Version control and multiversion coexistence can be accomplished by configuring the intermediary.

POL00137239

- Complexity is reduced from many-to-many connections and transformations to a pair of many-to-one and one-to-many. Opportunities arise to establish canonical formats to further reduce redundancy and complexity.
- Automatic fault resilience can be achieved by redirecting affected traffic in real time.
- Automatic support of heterogeneity can be added via intermediary switching protocols, access models and data types.
- Additional processing in the intermediary (security, business intelligence, monitoring, pattern matching and context injection) can add value.
- Redundant services supported by multiple providers (such as e-mail or fax) can be consolidated.
- An intermediary can establish configurable policies to select from among similar services offered from different providers, depending on the required SLA, price or other criteria — in real time.
- There is an opportunity for parallel processing, including support for multiprocessor computing grids. (This capability can be a foundation for massive improvements in performance and is built into the leading online transaction processing and extreme transaction processing engines and into EDA platforms [EDAPs]).

The Limitations of Intermediation

The indirect nature of intermediation also causes its limitations:

- Intermediation prevents tightly coupled transactional behavior. As a result, the traditional atomistic distributed transactions that occur via two-phase commit protocols are not possible. Error recovery in some distributed transactions may require compensating transactions — a weaker form of integrity protection.
- Request-reply interactions must be replaced with a "round trip" of decoupled intermediated interactions that are paired and coordinated by an intelligent intermediary.
- For users who are familiar only with the direct request/reply interaction model, additional programming using a less familiar programming model is required to interact with the intermediary.
- Insufficient standards typically lock in the application to the initially chosen intermediary.
- Maintenance (pruning) of a simple intermediary is required to prevent accumulation of "dead" requests.
- The intelligent intermediary is a separate middleware product and can be expensive to acquire, maintain and use, requiring staff training.
- Problems with the intermediary (for example, poor performance and technical failures) can affect the entire environment and a large number of services and applications.
- A shortage of productivity tools dedicated to the intermediated model of communication requires additional training.

POL00137239

Intermediation by Point-to-Point Message Queuing

The simplest form of intermediation is by program A writing a message to a named queue, while program B arranges to be notified when a new record is written to the queue or periodically retrieves the messages (if any) from the queue. This is simple intermediation, and any added logic would have to be in the software that posts and retrieves the message from the queue. Some queues are deployed in real memory to improve performance. Some queues offer automatic destructive read (the message is read only once and then is removed from the queue). Some queues can participate in standard, two-phase commit transactions to ensure message delivery and its integrity.

Intermediation by Pub-Sub

Pub-sub is an advanced form of intermediation. Program A posts a message of a stated type ("topic") to the pub-sub broker (intelligent intermediary). Any number of programs (B, C and so on) can indicate to the intermediary that they require notification when a new message of the named topic is posted (subscription). Thus, not only can the originally intended services be exposed to the topic, but services can be added, without intrusion on the running software. Moreover, new originators can be added and old ones removed while subscriber(s) continue to operate unchanged. BAM is most powerful in an environment where business interactions are driven by pub-sub architecture by adding listening services to all relevant topics. Many innovations and extensions are possible in the pub-sub environment.

Other Means of Intermediation

Queues and pub-sub brokers are the most popular but not the only means of intermediation. Shared memory, especially advanced distributed caching platforms, can be used for this purpose, as can databases and file systems. Choosing the most appropriate means of intermediation depends on the costs (in budget, effort and skills), the desired levels of simplicity and the required levels of integrity and performance.

SOA and Intermediation

Basic SOA typically is seen as the directly interactive architecture (and is reflected as such in the architecture of Web Services Description Language [WSDL], Web services, Java API for XML Web services and other core SOA specifications). Most SOA applications are implemented without intermediation: A invokes B by a direct-interface call. Advanced SOA includes other interaction models, notably the event-driven SOA style (EDA), which, in turn, typically is intermediated.

EDA and Intermediation

EDA is concerned with the processing of events. Generally, event objects can be passed to software elements using any method of communication. However, the nature of event origination favors the decoupled architecture of asynchronous and intermediated communication. The vast majority of EDA is implemented using intermediation (via queuing or pub-sub intermediaries).

Intermediation often is deployed in event processing, but it is not equivalent to it. In many examples of intermediation, the messages that pass through the intermediary are not event objects, and some event objects are communicated directly. Thus, not all event processing is intermediated, and not all intermediated communication conveys events. You don't have to master event processing to use intermediation.

POL00137239

Vendors and Products for Intermediated Communication Architecture

As mentioned, many middleware and platform product categories supply enabling technologies for intermediated communication. These include:

- Specialist queuing and pub-sub products include Apache ActiveMQ (open source), FioranoMQ, IBM WebSphere MQ, Microsoft Message Queuing and Tibco Rendezvous.
- Java Platform, Enterprise Edition (Java EE) implementations include Java Messaging Service, which has queuing and pub-sub capabilities. These include the IBM WebSphere Application Server, Oracle WebLogic Server, Red Hat JBoss Enterprise Application Platform, SAP NetWeaver and Sun Microsystems GlassFish Enterprise Server.
- Distributed caching platforms include Alachisoft's NCache, GemStone Systems' GemFire, GigaSpaces Technologies' XAP Enterprise Data Grid, IBM's WebSphere eXtreme Scale, Oracle Coherence, Red Hat JBoss' JCache and Terracotta's Terracotta Server. Several other notable offerings are under development and cannot yet be listed.

Most users have access to some (often multiple) of these technologies and are technically equipped for intermediated communication. We recommend that, after examining the advantages and challenges of this approach, users include intermediation on their shortlists of best practices for modern software design.

RECOMMENDED READING

"How to Get Started With Event Processing"

"Tutorial for EDA and How It Relates to SOA"

"Key Issues for SOA, EDA and WOA, 2008"

"SOA Applications Should Mix Client/Server, EDA and Conversational Patterns"

Note 1
Implicit vs. Explicit Intermediation

This research covers only the explicit form of intermediation. This is when the intermediary is visible to the application software elements (services, clients, event sources, handlers and others). In explicit intermediation, the source communicates with the intermediary without knowing the ultimate destination of its data (message). Thus, the intermediary is a software element that acts as a value-added pass-through or a broker but never as the final destination. It fully decouples the source and the destination, because both are aware only of the intermediary. The intermediary is shared by many application elements (source or target); it does not have a purpose, other than to facilitate and enrich communication.

In implicit intermediation, the application element addresses another application element directly, but the underlying middleware that delivers the message at runtime uses an internal intermediary to facilitate the communication. At the application design level, this is a direct communication: Only the source and the target are visible (A calls B). At runtime, middleware implements the communication via a lower-level intermediary (Oracle Tuxedo is implemented this way, and most message-oriented middleware and ESB products support direct interaction at the application level using implicit intermediation "under the covers").

POL00137239

The same intermediary can be used in both patterns. An ESB can be addressed as an intermediary via its APIs to post an event or to put an item on a queue. An ESB also can be used as a covert underlying implementation to what was designed at the application level as direct communication. A queuing system also can be addressed directly by the application logic (as an explicit intermediary), or the same queuing system can be used internally by runtime middleware to facilitate what was designed to be a direct communication at the application level.

Implicit intermediation delivers some of the same benefits as explicit intermediation (greater scalability, fault tolerance and manageability), without some of the costs (learning a new programming model). Implicit intermediation is a progressive step, as compared with fully coupled direct connections. However, explicit intermediation is the most powerful form of this architecture, offering not only performance improvements but also added flexibility and extensibility in design and a long-term evolution of the applications.

REGIONAL HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT GRO
U.S.A.
GRO

European Headquarters
Tamesis
The Glanty
Egham
Surrey, TW20 9AW
UNITED KINGDOM
GRO

Asia/Pacific Headquarters
Gartner Australasia Pty. Ltd.
Level 9, 141 Walker Street
North Sydney
New South Wales 2060
AUSTRALIA
GRO

Japan Headquarters
Gartner Japan Ltd.
Aobadai Hills, 6F
7-7, Aobadai, 4-chome
Meguro-ku, Tokyo 153-0042
JAPAN
GRO

Latin America Headquarters
Gartner do Brazil
Av. das Nações Unidas, 12551
9° andar—World Trade Center
GRO -São Paulo SP
BRAZIL
GRO

Gartner

POL00137239

Gartner

Research

Publication Date: 25 March 2008 **ID Number: G00155743**

Open Source in ESB Suites, 2008

Jess Thompson

The use of open-source enterprise service bus technology is widespread, and Gartner expects this to grow. However, its use must be supplemented with other technologies to provide the manageability, reliability and security demanded by production environments.

POL00137239

ANALYSIS

Situation Now

Open-source software (OSS) enterprise service bus (ESB) technology is available from multiple open-source communities, including Apache, ChainForge ESB, MuleSource and WSO2. In addition, Sun Microsystems' Java Business Integration (JBI) reference implementation (Open ESB) is available as an open-source offering.

The creation of most OSS products is driven by a combination of community notion, the existence of standards and the availability of OSS technology components. This potent combination led to the creation of multiple OSS infrastructure technologies, such as portals and Java Platform, Enterprise Edition (Java EE) application servers. In the same way, OSS ESBs are being driven by a collection of standards — two of the most notable being the Java Messaging Service (JMS) and JBI (aka JSR 208).

Standards most-frequently used to create OSS ESBs include:

- **Connectivity:** Support for JMS, Java Database Connectivity (JDBC), TCP, multicast, HTTP, SMTP and Post Office Protocol Version 3 (POP3).
- **JMS:** An application programming interface (API) for implementing reliable enterprise messaging. The core of an ESB is a Web-services-capable communication subsystem that can support optional mediation functions, particularly for (but not limited to) service-oriented architecture (SOA) applications (see "Open Source in MOM, 2008").
- **JBI:** A runtime architecture that enables plug-ins to interoperate via a mediated message exchange model. This enables the seamless addition of JBI-compliant services that perform validation, routing and transformation, as described above.
- **BPEL:** Several OSS ESB technology downloads extend the core ESB feature set. A feature frequently offered in conjunction with the ESB is a process orchestration engine that supports aspects of business process management (BPM).

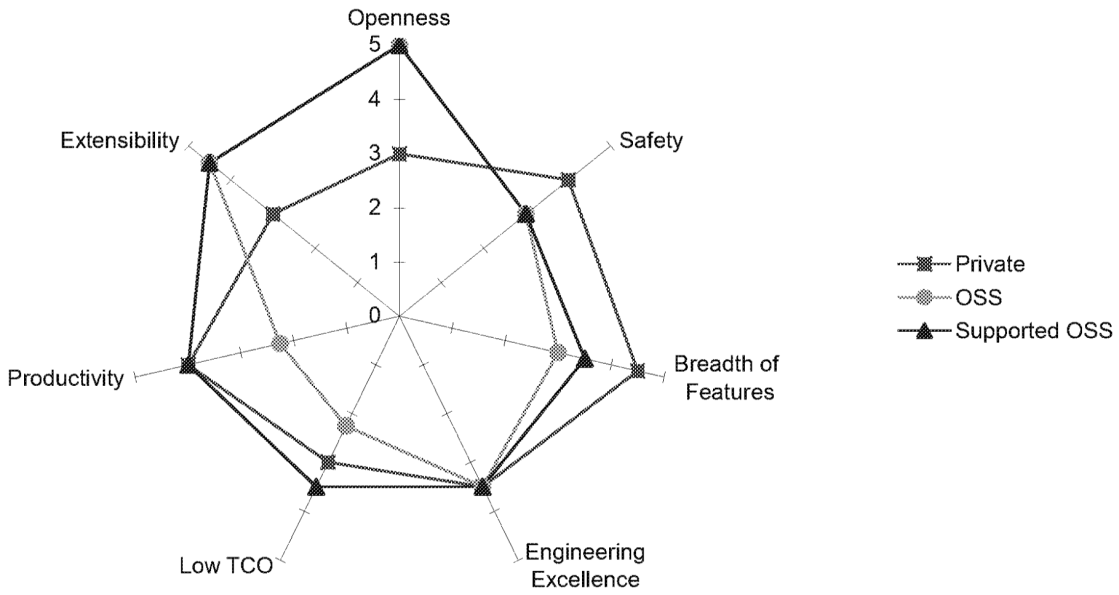
Note that none of these standards are definitional, and that their use varies among open-source communities.

OSS ESB technology also is available from vendors that offer it in conjunction with maintenance and services for that technology. Examples of such vendors include Iona (Fuse became available in July 2007), Red Hat (JBoss ESB became available in February 2008) and Sun (Open ESB became available in June 2005). Note that these vendors, although offering OSS ESB technology as stand-alone products, use it as a leader to broader, more-complex suites.

Combined, there have been approximately 1.5 million downloads of OSS ESB offerings, but it's impossible to identify how many of those downloads are in production. The primary reason for this is because there are no license fees for OSS technology. However, license fees are only one component of the total cost of ownership (TCO). The staff required for support and maintenance when OSS ESBs are used in a production environment adds significantly to the TCO. Figure 1 compares TCO with other salient characteristics for OSS, supported and commercial ESBs, using characteristics explained in Table 1. Note that it's unlikely the criteria used will be equally important to all organizations.

POL00137239

Figure 1. Comparison of OSS ESB Technology Sources for Production Environments



Source: Gartner (March 2008)

POL00137239

Table 1. ESB Characteristics

Openness	Integration, interoperability, portability, "pluggability," access and standards compliance (JBI, JMS and ActiveMQ)
Safety	Viability, commitment to market, continuity, manageability, cost of exit and customer experience
Breadth of Features	Breadth, completeness of function, and service from vendors and partners
Engineering Excellence	Internal architecture, code quality and delivery record
TCO	Total costs to include license, maintenance, support and staff skills
Productivity	Ease of use, training, learning curve and time to results
Extensibility	An architecture that enables users to plug in optional, intermediary functions to process messages in transit

Source: Gartner (March 2008)

Users are attracted to the use of an OSS ESB because it reduces capital expenditures (eliminating license costs) and vendor dependence. However, OSS technology is a double-edged sword. On the one hand, it offers significantly reduced costs for technology acquisition, documentation and updates. On the other hand, when deploying OSS technology, an end-user organization can end up devoting significant (and, in many cases, unplanned) head count to maintain IT infrastructure that uses OSS ESB technology. This can result in a backward situation in most organizations whose strategy is to *minimize* overall IT costs.

Future and Transition

By 2012, OSS technology used for SOA deployment will evolve to the point where the ESB is only one of many SOA backplane components (see "Predicts 2007: SOA Advances" and "Toolkit: Planning for Service-Oriented Architecture With the Gartner SOA Adoption Model") available from open-source communities. Examples of additional, available OSS components include registry, a BPEL process orchestration engine and SOA governance.

Relevance to Technology Users

Gartner metrics project that, during 2007, 22% of IT budgets were spent on SOA. With the appropriate support and services, OSS ESB technology offers the potential to *reduce* the portion of SOA budgets allocated to acquiring and deploying infrastructure.

Recommendations for Users

- Consider using OSS ESB technology as part of an incremental approach to building an SOA backplane.
- Identify the staff required to administer and maintain OSS ESB technology in a production environment. Note that OSS ESB maintenance will include the staff required to debug problem reports that are traced back to the ESB. Consider supplementing your staff with services and maintenance offered by vendors.
- Recognize that as your SOA maturity grows, your needs evolve from features provided by basic ESBs to those of a more-comprehensive SOA backplane.

POL00137239

- OSS ESB technology can be used in conjunction with other OSS offerings to provide the features of an SOA backplane. Identify the interoperability of OSS offerings. (JBI compliance is one approach to interoperability.)
- Recognize that an ESB isn't just about SOA. It also provides basic application integration features.
- Application infrastructure vendors should be wary of OSS technology. Adoption is growing. Vendors must offer ESB products with support and maintenance that provide a *cost-effective* alternative to an OSS ESB.

RECOMMENDED READING

"Predicts 2007: SOA Advances"

"Toolkit: Planning for Service-Oriented Architecture With the Gartner SOA Adoption Model"

"User Survey Analysis: SOA, Web Services and Web 2.0 User Adoption Trends and Recommendations for Software Vendors, North America and Europe, 2005-2006"

"Red Hat Seeks New Market With Open-Source SOA Platform"

"Open Source in MOM, 2008"

This research is part of a set of related research pieces. See "The State of Open Source, 2008" for an overview.

POL00137239

REGIONAL HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
U.S.A.
+ GRO

European Headquarters
Tamesis
The Glanty
Egham
Surrey, TW20 9AW
UNITED KINGDOM
GRO

Asia/Pacific Headquarters
Gartner Australasia Pty. Ltd.
Level 9, 141 Walker Street
North Sydney
New South Wales 2060
AUSTRALIA
GRO

Japan Headquarters
Gartner Japan Ltd.
Aobadai Hills, 6F
7-7, Aobadai, 4-chome
Meguro-ku, Tokyo 153-0042
JAPAN
+ GRO

Latin America Headquarters
Gartner do Brazil
Av. das Nações Unidas, 12551
9° andar—World Trade Center
04578-903—São Paulo SP
BRAZIL
GRO

Gartner