

OsEra Enterprise Service Bus

- OsEra Enterprise Service Bus 1
 - Principles 2
 - Characteristics of the OsEra Enterprise Service Bus 4
 - Overall Architecture 5
 - Mapping of OsEra Architectures to the ESB..... 7
 - Legacy wrapping - Enterprise Application “façade” in a Role 10
- SERVICES OF THE SUPPORTING INFRASTRUCTURE..... 12
 - Containers 13
 - Distributed Services..... 13
 - Process Monitoring..... 18

This section describes the technical architecture needed to achieve loosely coupled business collaboration as describes previously. This architecture is centered around business focused enterprise components integrated using open middleware such as web services. This portion of the architecture is referred to as the enterprise collaboration technical architecture.

Principles

A number of basic principles have been defined for implementing the Enterprise Collaboration Technical Architecture on an Enterprise Service Bus.

LOOSELY COUPLED INDEPENDENT ROLES

The architecture will be based on the concept of business roles that need to collaborate, and the implementation of the roles and collaborations between those roles using technology. The roles in the collaboration architecture must have their contract of interaction well defined and complete. There should be no coupling between roles other than that defined in these contracts.

(Note - in some circumstances, roles will be coupled together when implemented. Where integrated vendor suites are used, a number of roles may well be effectively grouped together into a “super-role”. Some existing “legacy” systems may present several roles with tight back-end coupling. For practical and economic reasons these situations may continue, but these should be de-coupled where practical).

COUPLING OF BUSINESS PROCESS

While the roles are independent, the business processes in which they are participating must be well defined across all participating roles, like any contract among independent parties.

SERVICE AND EVENT BASED ARCHITECTURE

The architecture will be based around the concept of re-usable services and service specifications, both at the business (computational) level and at the engineering (infrastructure) level. One special point to note here is the need for a rationalized set of business service definitions, which are easily accessible and searchable. The concept of services applies to one-to-one interactions where the initiating roles need are aware of the identity of the other role. In cases where one to many or anonymous interactions apply, event publication and subscription mechanisms will be supported.

CONTRACT OF INTERACTION

Due to the requirements of loose technology coupling and high business cohesion, the contract of interaction becomes architecturally significant. The semantics of this contract must be sufficient for the business and technology requirements. The maintenance of such contracts becomes critical and highlights the importance of repositories for management and access to them.

Thus the technical infrastructure must include repository capabilities such as may be found in MOF, ebXML or UDDI.

TECHNOLOGY INDEPENDENCE

The interaction media and execution platform technology and product set are an important decision, but it would be naive to assume that any such technology choice will remain constant or may be imposed on all parties of a business process. There is substantial and long-term investment in defining, implementing and integrating business processes and the support technology. The architecture must enable technology-independent specification of collaborations that are bound to the technology as late as possible in the development process.

OPEN STANDARDS

Related to the previous point, the architecture and related technology choices need to be based on open standards as much as possible and practical.

DISTRIBUTION TRANSPARENCY

Business components should not know whether they are distributed or not. Calls out from the components should all be “local” from the components perspective (they may be to a proxy, which would then send the message or execute a remote call as appropriate)

LOCATION TRANSPARENCY

Individual components within the architecture should not need to be aware of the physical location of other components.

For these reasons the specification and implementation of business logic components should be as independent as possible from specific media, data formats, middleware or platforms.

Providing for technology independence will allow more flexibility over time, a longer lived systems asset, greater freedom in integrating new business partners and a capability to withstand both business and technology change over time.

Characteristics of the OsEra Enterprise Service Bus

This section contains a brief summary of some of the main characteristics of the ESB. Each item is covered in more detail in subsequent sections.

1. Integration of business units (E.G. Analysts, Information Providers (Internal and External) , Information Consumers (Internal and External) and applications will be provided based on the modeling and implementation of collaborations as specified in the Component Collaboration Architecture (CCA), a subset of the OMG Enterprise Collaboration Architecture (ECA), which is part of the UML Profile for Enterprise Distributed Object Computing (EDOC). This specification is intended to be technology neutral and thus should support any infrastructure selected.
2. Collaborations will be based on large-grain document exchange between actors playing roles in a collaborative business process. These collaborations will include synchronous, asynchronous and event (pub/sub) interactions but asynchronous interactions are the primary and preferred mechanism. Document information will be encoded in XML, also making use of WSDL/SOAP packaging where appropriate. Transport will be over HTTP(S) or using a message service (e.g. IBM MQ or MSMQ) as appropriate. *Note: XML is a technology of choice today but the architecture would allow the automated substitution of any middleware technology.*
3. The collaboration architecture will be expected to be able to support long-running business processes, but each interaction will be an atomic transaction.
4. Implementation of services will be secure and robust. Application and role-based security will be supported. Application servers will support reliable messaging, load balancing, logging and fail-over.
5. The overall architecture is based on industry standards as far as possible, however these are developing and fairly immature with respect to collaborations. The basic set of standards being followed is based around the concept of “web services” (XML, SOAP).

In general, discussion will avoid mention of specific products. However, there are a small number of assumptions that are made with respect to technologies that the architecture must be able to support. Examples are that custom components will be able to be built using Java (J2EE Servlets or EJBs) or using the .NET framework; Message Queuing will be able to be supported by JMS, IBM MQ or Microsoft MSMQ.

One additional concept that may also be referred to throughout this documentation is that of a “Business System Domain”. This is defined as “a managed set of services and resources.” It is characterized by the following:

shared resources (which are in a consistent state with respect to each other)

has defined boundaries

provides a defined QOS, reliability, performance

consistent, defined set of contracts between the elements

The purpose of identifying this concept is that it provides a scope for an individual set of work or discussions. This does not necessarily imply a fixed hierarchy, domains can legitimately be defined in different ways for different purposes, e.g. naming, security or business processes. A business systems domain is also a coherent set of technologies working together.

USE OF COMMUNICATION STYLES AND TRANSPORT PROTOCOLS

Depending upon the level of granularity to which role modeling is carried out, the methods of communication used between components becomes less of an architectural concern as the drill down goes further. Ideally, a role represents a specific business function, but in many cases, the granularity of the systems components is larger.

Bearing this in mind, the default method for collaborations **between** roles will be to use document-based message exchange with standard message headers that will be defined. This will apply to both asynchronous and synchronous exchanges. XML may be transmitted in a WSDL/SOAP-RPC (as opposed to document) form in some circumstances. This provides a means for interacting via finer-grained, object method level calls. However, SOAP-RPC offers no real performance advantage over document style exchange, and in general, larger grained communications are encouraged. In general, RMI or similar methods are not endorsed.

In some cases, HTTP or HTTPS may be used for transport, i.e:

- Single point-to-point request reply interactions
- Inquiries
- Interactions with UI and external (B2B) clients

In the enterprise architecture, the means of communication used **within** a role is less significant (e.g. from business logic to adapter in the generic role presented below). Ideally, and this is reflected in the pattern descriptions, XML format at least should still be used where a choice is available.

Overall Architecture

DESCRIPTION

It is important to describe a key concept used in the architecture, i.e. that of replaceable roles. The architecture is “role-based”, i.e. is based on a collaboration model defined around the interaction of business roles, and that these business roles may be played by many individual user and/or systems components, which themselves may fulfill many roles.

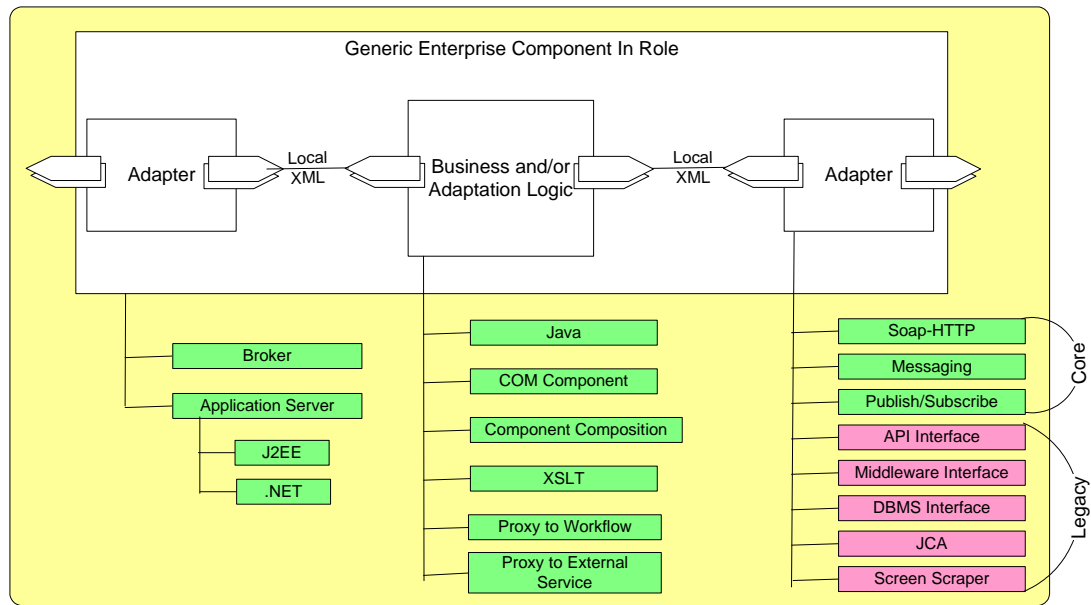


Figure 1 – Generic Role in Collaboration Architecture

Within each role, there is a basic generic pattern that is applied, which is reproduced in Figure 1. It does have different variations in certain circumstances, but applies in this form to many roles. As discussed earlier, communication between roles will be achieved using a standard set of protocols based around XML, however many different technologies and products will be used to provide the systems implementations of roles, so no assumptions can be made about the internals of a role – E.G. it may be java, cobol or a COTS application.

Because any single implementation of a role may communicate with other enterprise roles, a protocol adapter needs to be used. In the case where the component used the same protocols internally, this would in theory be unnecessary, but even then there needs to be a means to handle distribution concerns.

There are three basic components defined within the role. Each of these is described below.

Adapter (in)

The first is the input adapter shown on the left. This will transform the “enterprise” protocol and format into the protocol and format used internally within the role (which is ideally still XML-based). Note that this is technology-level transformation as opposed to business logic transformation. (This is expanded in 3.1.5 below)

Business and/or Adaptation Logic

The central component will carry out the business functionality of the role. This itself will often be a composition of other finer grained components and their in-

teractions, and may also include a proxy to another component in a different role (see the encapsulated intermediary pattern later in this document). Adaptation here refers to any business level adaptations or transformations, as opposed to technology protocol transformations. The two must be kept separate.

Where a custom component is written (i.e. Java or .NET), the XML would be parsed by the adapter. There are various techniques that may be used here, e.g. JAXP (DOM, SAX), JAXB, .NET XmlReader etc. In the case of DOM, a reference to the DOM tree would be passed to the Business Logic component. Various automated tools and facilities have been or are being built for XML to component mappings and conversions, and most Application Servers either include them out of the box, or plan to in the future.

It is important for future-proofing components in the architecture that business logic components not be written to technology container interfaces, such as EJB or .NET. Rather, business logic components are written to the middleware neutral interfaces generated by the component architecture tooling and infrastructure. It is then the job of the infrastructure to supply the adapters to technology and/or container specific interfaces and capabilities. Where possible, business logic is expressed completely in terms of models (or compositions of generic components), thus eliminating all technology specifics and hand coding.

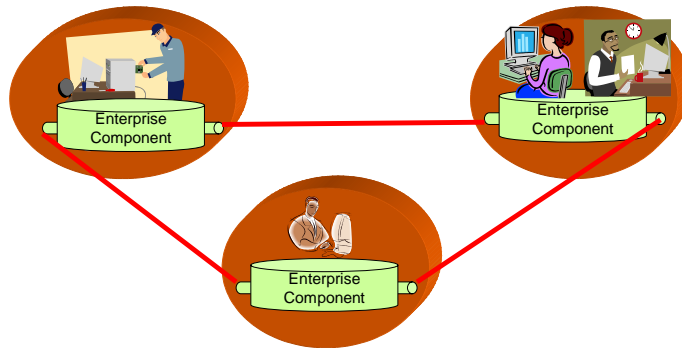
Adapter (out)

The third component is the output adapter, which provides protocol transformation back to the enterprise protocol or in some cases to a specific legacy or proprietary integration protocol. Adapters are bound to the components by the collaboration architecture MDA machinery.

Mapping of OsEra Architectures to the ESB

OsEra Architectures provide the business model that will drive the technical architecture. Enterprise components will be defined such that they implement business driven roles at multiple levels. Each enterprise components will be implemented using the pattern described above on the selected technical platform., the default being J2EE. Each protocol of an enterprise component will be mapped to one or 2 WSDL (Web Services) interface specifications such that they support the business architecture. The details of the WSDL mapping will be developed separately but are based in principle on standards in progress at the OMG. With each business service mapped to a technical component (with its own supporting user interface as appropriate) the architecture becomes realized as the technical components interact in accordance with the business architecture, effectively implementing the enterprise level workflow supporting the GSA value chain.

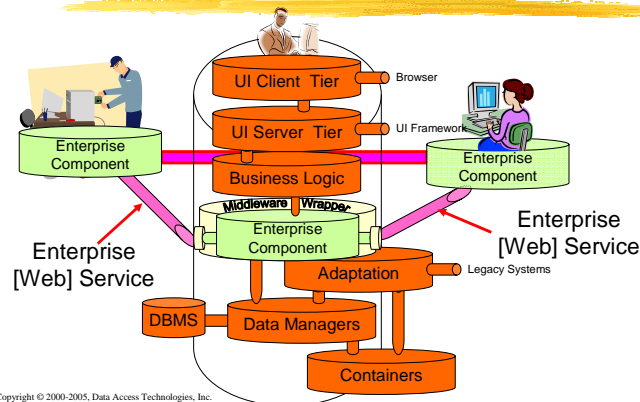
People, Components & Organizations Collaborating



Copyright © 2000-2005, Data Access Technologies, Inc.

The above diagram shows how a combination of people, organizations and enterprise components collaborate to implement the business model. Enterprise components will have user interfaces relevant to the works playing the various roles (if appropriate) and will then encompass both a systems interaction and human interaction as a coordinated and choreographed design.

“Lower” PIM View - Enterprise Component Internals



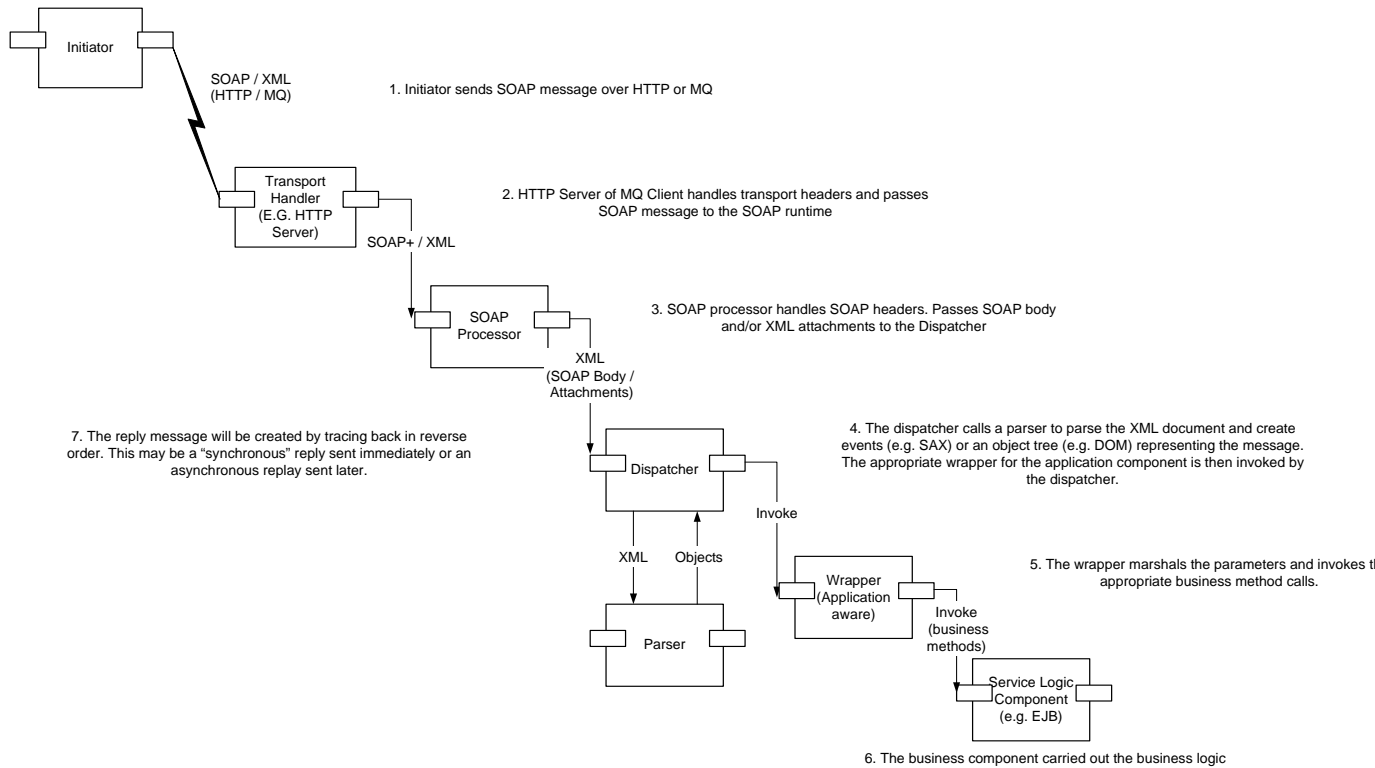
Copyright © 2000-2005, Data Access Technologies, Inc.

The above diagram shows how components of the the technology architecture (UI Client Tier, UI Server Tier, Business Logic, Adaptation, Data Managers and DBMS) will then be (conceptually) “inside” the implementation of each role (even if some of that technology is, in fact, shared). This technology architecture will join the users, business logic, data and legacy systems to implement the enterprise component as part of the business process.

SOAP / XML PROCESSING

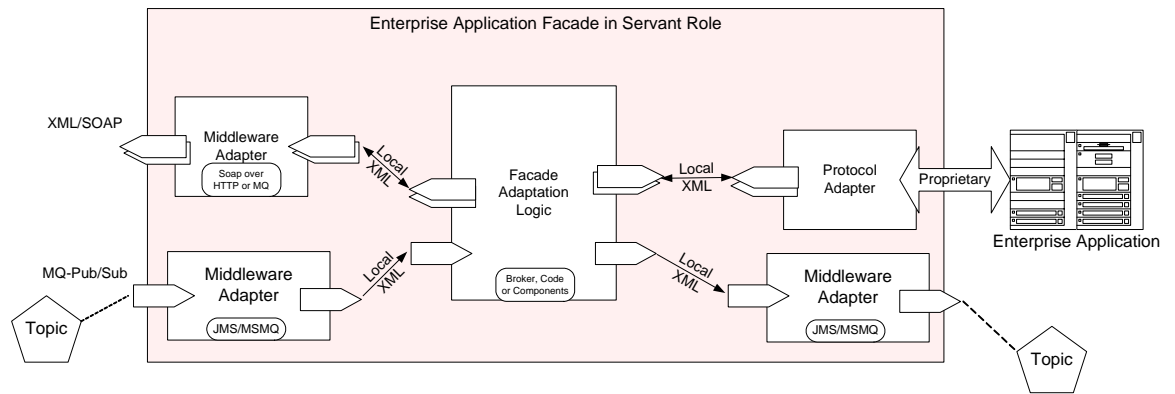
Communication between roles within the ECA is based around the use of SOAP and XML. However, the implementation of most roles will be by vendor applications or J2EE/.NET custom components. Many vendors are currently in the process of “XML-enabling” or “web service enabling” their products, so as time progresses, more and more of them will be able to process the SOAP/XML “natively” (at least from the point of view of a Client). Where this is the case, the means they choose to do so is a matter for them. In the case of J2EE and .NET, later sections in this document give an outline of the capabilities currently offered. In the mean time, a generic outline pattern is given below for processing a document-style SOAP/XML message using a role implemented by a component. This is a drill down inside one possible view of the “adapter (in)” component in the generic pattern above. This is also really a “worst-case scenario”. Each of the logical steps will need to be followed, although some could be combined into a smaller number of technology components and development toolkits should enable most of this to be handled transparently.

Since the architecture is middleware independent, other forms of interaction may subsume SOAP in the future with no change to the architecture, no change to business logic components and only small changes to the supporting infrastructure.



Legacy wrapping - Enterprise Application “façade” in a Role

This pattern will be used to create a common business level services layer in front of existing “legacy” applications. This both enables better reuse of existing capability and also enables the back-end systems to be subsequently replaced with little effect on the service clients.



➤ Figure 2 - Enterprise Application Façade in Role

LOGICAL COMPONENTS

Component	Responsibilities	Characteristics	Other Comments
Enterprise Application	Business Logic and Data Access		Any back-end system (existing or new)
Protocol Adapter	Transform Technology Protocol		Protocol conversion to enterprise application. Converts to/from XML messages or events. (Although “proprietary” is shown in the diagram, some new systems may conform to our standard technology protocols)

Component	Responsibilities	Characteristics	Other Comments
			<p>Protocol adaptation should ideally not be written in a legacy technology. It should be automated where possible, and carried out in a standard way, e.g. a standard adapter for CICS COBOL.</p> <p>The identification of specific preferred methods of adapters to specific technologies will not be dealt with in this document, although this need to be addressed in the near future. However, open standards (e.g. JCA) should be followed where available.</p>
<p>Facade Adaptation Logic</p>	<p>Business Document Transformations</p> <p>Control flow (micro process)</p> <p>May access local databases for transform look-ups.</p> <p>Data type conversions</p> <p>Coupling/de-coupling of messages</p>		<p>Common Business Services will be defined and implemented here.</p> <p>This may include business object logic.</p>
<p>Middleware Adapter (in)</p>	<p>stripping transport protocol and pars-</p>		

Component	Responsibilities	Characteristics	Other Comments
	ing XML .		
Middleware Adapter (out)	Publishing events to topic (pub-sub system)		JMS for Java cases. Must include ability to generate events for business process status.

SERVICES OF THE SUPPORTING INFRASTRUCTURE

Up to this point, the discussion has related to how different roles collaborate with each other at a logical (or computational) level. In order to provide a means of implementing these collaborations, a set of infrastructure must be provided that provides a set of common engineering level services. This section presents a description of these infrastructure services, and the assumptions made by components using the infrastructure.

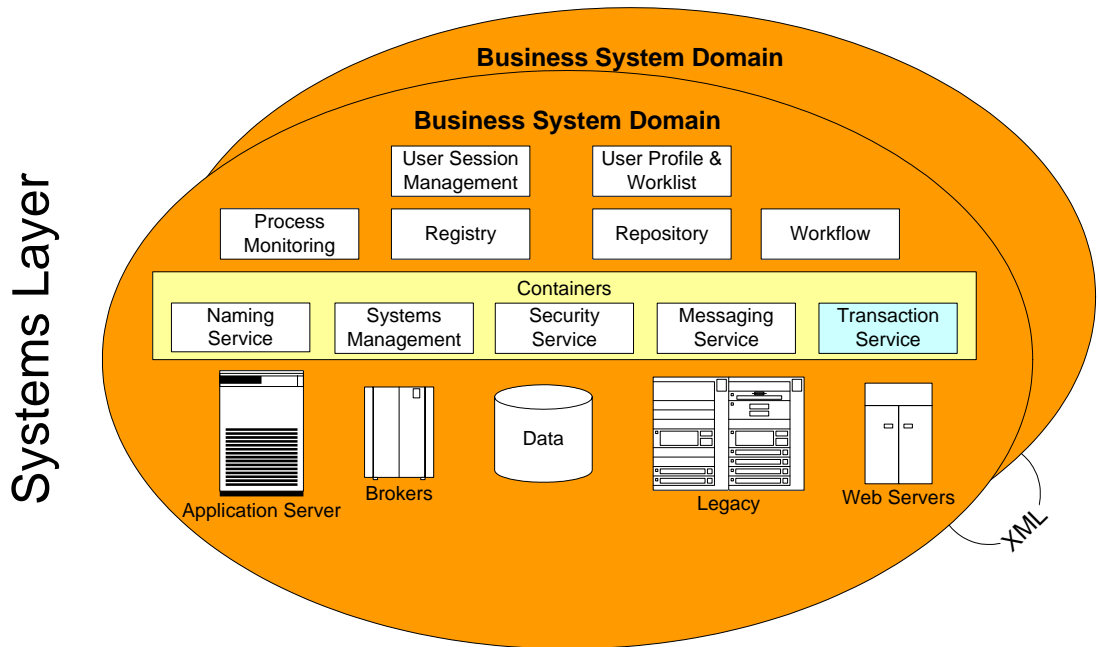


Figure 3 - Engineering Services Layer

Containers

The first assumption made with respect to the implementation of “logic” components, (i.e. any of the middle components presented in the role patterns) is the presence of a “container”, which is defined as “a deployment environment for application components, which provides the following services to the deployed components: naming, systems management, security, messaging, transaction”.

Each component living within a container will assume that its container provides the following capabilities, i.e. that individual component designs do not need to be concerned with them:

- Fail over, Load balancing
- Thread management / resource pooling
- Activation / deactivation of components
- Local transaction management
- Component Persistence

At this stage, no specific constraints have been identified with respect to the ECA for the above capabilities.

In addition however, the container is assumed to provide the capability to interact with certain enterprise distribution services. The containers must provide these interfaces using open standard interfaces where they exist. These services are:

- Naming Service (for Java containers – JNDI, .NET uses DNS)
- Systems Management (for Java containers – JMAPI/JMX)
- Security (for Java containers - JAAS / GSS-API)
- Messaging (for Java containers – JAXM / JMS)

Each of these services is described in the following sections, along with other services where no assumptions are made with respect to the container. Note that the interfaces to other services may still be provided or assisted by the container, but no assumptions are made with respect to these.

Distributed Services

NAMING/DIRECTORY SERVICE

This provides the ability for any component in the architecture to locate any system resource (i.e. other components, services, data sources). This provides location transparency, i.e. translates a logical name to a physical location.

The infrastructure will provide a naming service that is useable at a minimum by Java and Microsoft components / containers. For Java, the standard is JNDI. .NET address look ups are handled by Windows and Active Directory using DNS (Domain Name System).

A topic related to the naming service, but with more general applicability is that of namespaces. The definition and identification of standard namespaces will be considered as part of ongoing enterprise architecture work.

SERVICE METADATA REPOSITORY

This provides a network accessible repository of service specifications (contracts) for business and engineering services. The ability to quickly and easily locate appropriate existing service definitions and specifications is a critical enabler for reuse of services. From a management perspective, it is vital that all service specifications are entered into the repository.

For interoperability, the repository should provide interfaces conforming to the MOF specification (Meta-Object Facility) defined by the OMG.

REGISTRY

This provides a registry of service instances described using the OMG Enterprise Collaboration Architecture metamodel and WSDL. This is used by systems administration for configuration purposes, who enter or migrate the information to the naming service. The naming service provides the run-time access to components for this information.

The current standard for the registries is UDDI. At a later date, further investigations may be carried out into the need for an ebXML compliant registry.

From a technology perspective, the registry and repository capabilities may be provided by the same product.

MESSAGING SERVICE

The message service exists to provide reliable (i.e. guaranteed delivery or notification of failure), secure delivery of point-to-point messages and publication and subscription of events.

The messaging service will provide JMS and MSMQ interfaces. The future direction includes a likelihood that the message service will need to support the ebXML message service specification to guarantee true interoperability, but it is recognized that this is too limiting a restriction at this time. (The product vendor should include this as a stated direction). A JAXM provider will also be needed within the messaging infrastructure for use by Java components (subject to further clarification by the Java Community Process – there is some disagreement

amongst software companies over the relationship between the JAX-RPC, JAXM and JMS standards and the future direction).

The message service will also provide automated logging of messages, and must be capable of supporting the extremely high volumes and throughput required by collaborations within the ECA.

EVENT SERVICE

The event service provides a publish/subscribe environment whereby event producers publish information of interest to the “event cloud” and consumers subscribe interest to the same event cloud. Delivery of events is asynchronous and may be guaranteed by the event infrastructure. The producers and consumers do not have to be active at the same time. In these situations, the event service will buffer requests between them.

The event service will provide the following capabilities:

- Guaranteed delivery

- Deferred delivery

- High Performance / Throughput

- Granularity of subscription and capability to subscribe based on message data

- Scalability and distribution capabilities

- Integration with Java Messaging Service (JMS) and MSMQ

Messaging infrastructures are generally agnostic as to the structure of data carried by the events. Recently XML has become more widely used as the data structure carried by event systems. Standardizing on XML over events has the added advantage content-based (as opposed to header information) subscription may be supported, which further de-couples the endpoints.

SECURITY SERVICES

The infrastructure will provide a set of services related to security. Those with specific relevance to collaborations are identified below, in addition to a statement of their characteristics or assumptions made with respect to components using the services. It is not within the scope of this document to fully describe security architecture.

Secure connection

This will be achieved by the transport layer, i.e. HTTPS (SSL) or the message service (MQ / MSMQ) as appropriate. In sensitive applications the collaboration infrastructure will utilize the security infrastructure provided or specified by the customer.

Authentication and Authorization

A common means of authenticating and authorizing roles will be provided. Within the ECA, individual components will not be permitted to implement these facilities as externally supplied capabilities; the following are an example of commercial grade security capabilities – sensitive applications may require more.

Authentication

Both PKI and Username/password authentication will be supported. At a minimum, PKI will be used for authentication of internal users (employees / contractors) in new applications. The Username / Password combination may be used in low-risk situations or temporarily for practical reasons. Both must be managed by the same session manager.

Role based authorization

The security service will support role-based authorization. The roles used must be the same roles as the business roles defined for the collaborations, or must encompass them.

Application access to principal identity and role based security

Applications involved in collaborations must be able to determine the principal, and whether the principal is authorized to be in the collaborating role.

Principal identity will be carried in the message header right through chains of collaborations. The principal identify will include the full chain of trust (the user, the program and all intermediaries) Each intermediary adds its certificate as the message passes through. There is always an original initiator which must be explicitly identifiable. With this in mind, each server process playing a role in the collaboration architecture must have its own security credentials.

Role policies

Policies are used to authorize roles to use or perform services or assume other roles. These must be explicitly defined and user-maintainable.

Credentials

Issuing of credentials, particularly digital signatures, requires a Certificate Authority, either internal or external to GSA, and this must be integrated with other GSA systems, particularly HR.

Non-Repudiation

The ebXML specification for security is included by reference and defines non-repudiation for both sides of a collaboration (initiator and responder) as follows:

non-repudiation (initiator) – sender must sign and receiver must save an audit trail of

messages sent

non-repudiation of receipt – must digitally sign receipt ACKs and sender must save audit trail

The ebXML BPSS and Messaging Service specifications have defined a solution for non-repudiation, which will be used as a basis for supplying non-repudiation capabilities where required.

Encryption

Where appropriate, header and/or content information in messages will be encrypted. The infrastructure will provide standard means for carrying this out. This will need to include facilities for “encrypted partial content” (different parts of the message which may be visible to some endpoints but not intermediaries, e.g. credit card and health information).

There is much activity in this area continuing in standards bodies. One particular example is WS-Security, dealing with digital signatures (XML DSIG) and XML encryption and PKI. This is a probable direction for the GSA architecture, but this will be confirmed as and when the standard becomes adopted by w3c.

TRANSACTION MANAGEMENT

Within and across collaborations, compensating transactions should be used. Within a role, the underlying database transaction mechanisms will be relied upon to guarantee integrity, and within a single container environment, the container itself will provide transaction management capability. Distributed Transaction Management is basically unsupported by current products. If an unavoidable need is encountered in an individual case, a technology solution will be sought at that time.

SYSTEMS MANAGEMENT

For the architecture to be workable, the components must be deployed in a fully automated, managed environment. The following capabilities will be provided in a common way by the infrastructure:

- Remote deployment
- Clustered deployment
- Remote and clustered startup and shutdown
- Remote and clustered monitoring
- Remote and clustered performance tuning
- Remote and clustered logging (include error logging)
- Distributed namespaces
- Dynamic changing of the distributed namespace (i.e. re-assign location of a service)
- Context sensitive namespace (i.e. service location depends on client location or other factors)

- Monitoring and management of active (persistent and executing) business processes
- Hot upgrade – (Dynamic changing of implementation revisions in server)
- Management of clients
- Upgrade of clients
- Security policy
- Backup of active processes
- Service based interface to systems management

Component and container responsibilities are to be able to report out appropriate events and status information. For Java components/containers, the JMAPI (JMX) should be used.

SESSION MANAGEMENT

The infrastructure must provide the capability to manage user sessions, including B2B automated clients as well internal and external UI users. Session management will be further developed in the next revision of the technical architecture.

USER PROFILE MANAGEMENT

Allied to session management, the infrastructure must also provide the capability manage user profiles and user work lists. Again, the Client in Role section describes this above.

WORKFLOW & BUSINESS PROCESS EXECUTION

The business architecture specifies activity and messaging constraints that can be implemented in multiple ways, including;

- Business objects provisioned directly from the architecture – E.G. EJB session beans that enforce the choreography based on the architecture.
- Workflow Engines that are provisioned based on the architecture
- Business process engines, such as “BPEL”, that manage service interactions at runtime and are likewise provisioned from the architecture.

The baseline implementation of the OsEra ESB utilizes a BPEL engine interacting using web services and messaging.

Process Monitoring

Roles in a collaborative process are independent and should monitor their interactions to make sure that processes are proceeding as required. For example, time-

out parameters on an interaction may notify an application of a business partner that does not respond. Full knowledge of the state of that process from the perspective of any one role is really only the business of that role. So from a strict perspective, process monitoring and being able to monitor a specific role are the same thing.

However, an enterprise may be fulfilling many external roles and hundreds or thousands of internal roles. Within a managed environment it is necessary to be able to browse the state of multiple processes across multiple roles. This is the requirement for process monitoring.

The mechanism to be used for process monitoring will be based on instrumentation with events. Each component framework has an intrinsic capability to pose process interactions to the event infrastructure of the domain – when processes are created, completed or have significant state changes. Since these are published to the event system, any authorized process may subscribe to these events.

A process monitor application then subscribes to these process life-cycle events and keeps a database of their progress. This database may be queried to view and manage processes on an enterprise-wide scale with no special binding to those processes, other than the generic event service. The monitoring application may also be able to use the systems management capabilities to pause or stop errant processes. The process monitor solution will provide a UI component for management purposes.

Thus process monitoring is accomplished in a non-intrusive manner within the paradigm of loosely coupled roles and events.

Process monitoring is the driving capability behind business intelligence and realizing line of site – the connection of business metrics with live data observations. These observations will be made using the process monitoring capability.