DataAccessTechnologies

Where Business Meets Technology

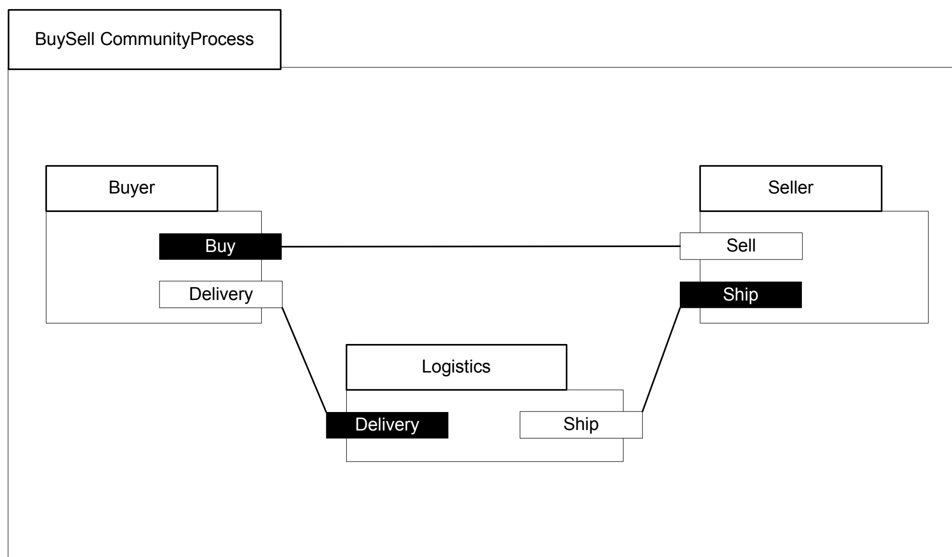
An Enterprise Systems Architecture for Collaborative Business Processes

Rev 0.1 - March 2, 2002

Overview

A Collaborative Business Process (CBP) is a loosely-coupled but well defined business process involving a number of actors across divisions, companies and geographic boundaries. The CBP is defined in terms of roles within that process interoperating across open middleware.

This architecture defines the systems environment required to support such collaborative business processes as well as the design, implementation and deployment considerations of enabling collaborative business processes.



Example of a collaborative business process

The concepts and models of the CBP are assumed to be aligned with the OMG-EDOC standards “Enterprise Collaboration Architecture” which is included by reference in this architecture.

Copyright © 2002, Data Access Technologies, Inc.

Table of Contents

- Overview 1
- High-level concepts 3
- The mix of business and technology components 5
- Legacy Integration 5
- Important Architectural points 6
  - Loosely coupled independent roles 6
  - Coupling of business process 6
  - Contract of interaction 6
  - Technology independence 6
  - Requirement for distributed transactions 6
  - Non-distributed components 7
  - Collaborative Vs. managed processes 7
  - Services Architecture 8
  - Event based computing 9
- Choice of interaction media 10
  - The services support media 10
  - Event support media 11
- Platforms 12
- Platform Support Facilities 13
  - Component Framework 14
  - Messaging 15
  - Services 15
  - Automation 16
  - Application servers and containers 18
- Support for standard and legacy protocols 18
- User interface 18
  - Three-tier structure 19
  - Web-based user interface 19
  - Application based user interface 20
- Process Monitoring 21
- Shared Data 21

Copyright © 2002, Data Access Technologies, Inc.

Business Object Entity Components 22
Security 22
Role based security 22
Connection Security 22
Identity and credentials 22
Authorization 22
Application security 23
Systems management 23
Validation and Quality Control 23
Problem 23
Ubiquitous Services 24

High-level concepts

The actors in a business process (people, companies, division, automated systems) are described in terms of their roles within the business process. These roles have ports that are their communication points to other roles. In the systems environment each role has a corresponding distributable business component (DBC), which plays that role in terms of the systems environment.

In the process model the roles interact in a well defined way to implement the business process. In the systems environment, each port on a DBC (corresponding to a communication channel of the role) corresponds to a technical interaction point for a two-way conversation between components. This interaction may be implemented across a variety of local and distributed communication media, provided that such media meets the business and technology requirements. The selection of such media is a primary systems architecture point.

DBC's may be hosted on a variety of platforms including Unix, Microsoft NT, Mainframe systems and other legacy systems. The technical environment to host the DBC on such platforms is another systems architecture point, this may include containers and other application server technology.

Communications between DBC's may be either point-to-point service based interactions or across an event system, such as IBM MQ-Series. The choice of services or events is reflected in the high-level business process and is a primary design point for those processes. The choice of using a service based, event based or mixed environment is another systems architecture choice, but one influenced by the process models.

Since roles may be assumed by users as well as other systems, user interface capabilities must be supplied. These may use either web-based and/or application based user

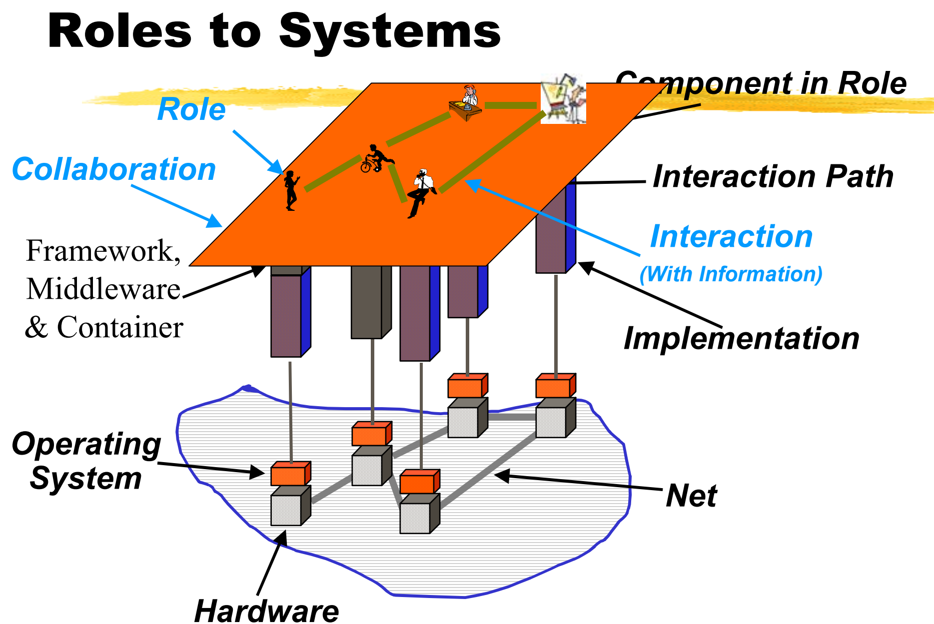
Copyright © 2002, Data Access Technologies, Inc.

interactions with various tradeoffs. The choice of web-based or application based user interfaces has systems architecture impact as well as impact on flexibility, ease of maintenance and fidelity of the user interface. User interface choices will require support software and tooling.

Other considerations are access to shared data, the requirement for local or distributed technical transactions and the capabilities of legacy systems to support various forms of integration.

The technical environment may require services available to multiple components, the most common being a repository. These “pervasive services” are defined in terms of their interactions like any other component in the environment.

The following picture illustrates how high-level roles are drilled down to supporting physical technologies.



To summarize critical systems architecture decisions;

- The choice of interaction media and middleware
- The choice of platforms and platform support software
- The choice of web based or application based user interface and the support infrastructure
- The choice of service and/or event based architectures
- Technical transaction boundaries

Copyright © 2002, Data Access Technologies, Inc.

- Shared data and pervasive services

The mix of business and technology components

While the high-level picture shows only business roles, the executing systems environment will include other components as well. These will be system services, repositories, transformations, adapters, transaction coordinators, business objects, messaging queues and other components to support the business roles.

In a systems sense the technology components and business components have the same "shape" in the system, but they are designed at different layers. The systems components are the responsibility of the implementation team.

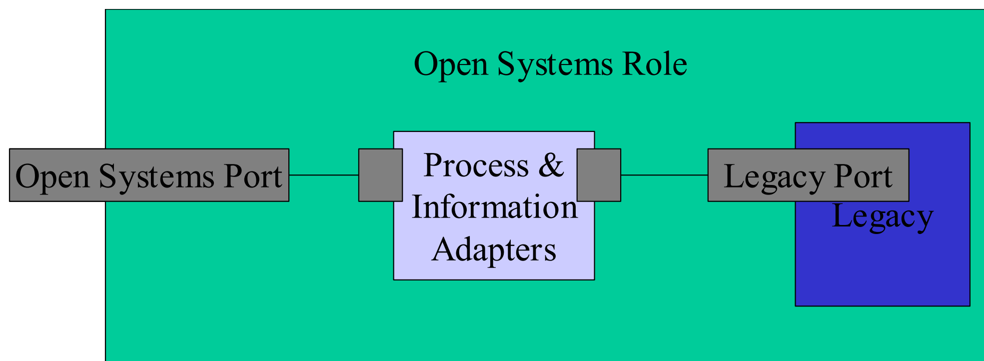
System components may have different interaction requirements than business components. For example, mist business interaction can use interactions that are atomic from a technical sense, while technology components may require a transactional interaction.

Legacy Integration

Since one of the primary requirements of this architecture is the integration of legacy systems, the capabilities of these legacy system must be considered. For example, what interaction media are they capable of using? Can they participate in distributed transactions? Is the best interface via an API or through the data?

Each legacy system may present unique problems and constraints, particularly in terms of the interaction media.

Since legacy system have such limits it is common to have a more restricted interaction with these systems and to adapt them to the systems architecture via wrappers. These wrappers are also components, ones that use the legacy interaction capabilities on one side and the enterprise-common media on the other. In between are both technical (protocol) transformations as well as transformations in process flow & information. These transformations are accomplishes with other components for this adaptation role.



The above diagram shown how a legacy system is wrapped by (one or more) roles in the business process. Separate components are used to adapt the information and process of the legacy to that required of the role in the system as a whole. The adapters may be executing on the same system as the legacy or on a separate "broker" system node.

Copyright © 2002, Data Access Technologies, Inc.

Important Architectural points

Loosely coupled independent roles

The roles in the collaborative business process must have their contract of interaction well defined and complete. There should be no coupling between roles other than that defined in these contracts.

Legacy systems may temporarily violate these constraints. Such legacy systems may present several roles with back-end coupling. This situation may be tolerated as an interim system but an initiative should be in place to de-couple components.

Coupling of business process

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

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 suggests the importance of repositories for management and access to them.

Thus the technical infrastructure must include repository capabilities such as may be found in 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.

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.

Architecting for technology independency 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.

Requirement for distributed transactions

Early attempts at widely distributed systems extended the technical transaction model to the distributed system. In this model multiple systems work within a single distributed transaction coordinated with two-phase commit and a transaction manager. Such transaction coordination is absolutely required between some components, but not all.

The trend in distributed systems is to limit, as much as possible, distributed transaction interaction. Such interactions require a technology and time-based coupling between

Copyright © 2002, Data Access Technologies, Inc.

components. Specific infrastructure is required by all parties participating in such transactions and system resources become locked.

The newer style of interaction uses larger-grain interactions between parties that are each atomic (in a technical transaction sense). Any “backout” of business processes is accomplished with counteracting transactions.

The large-grain transactional atomic style of interaction is the basis for all of the “internet” based paradigms, such as web services. Asynchronous systems, such as IBM MQ-Series have assumed such a restriction for some time. This a move to asynchronous interaction is supported by an atomic transaction approach.

Thus as an architectural point, it is necessary to assess when distributed transactions are truly required.

Non-distributed components

While business and technology components are defined in terms of their external contract such that they may be distributed, distribution is only one option of coupling. Components or compositions of components should be able to be co-located on the same systems or within the same process and should be able to take advantage of lite-weight interaction mechanisms when so co-located. The component architecture should enable but not require a distributed infrastructure at any point. In some systems environments the ability to participate in a shared technical transaction will require co-locating components.

For example, a legacy system interface may be co-located with the process ind information adapter components that expose it to the enterprise collaborative business process.

Collaborative Vs. managed processes

Collaborative processes are like those that exist in any community, actors working together to achieve some goal. A buyer-seller relationship is the classic example of a collaborative process – the exchange of assets for mutual benefit. A key aspect of collaborative processes is that while one actor may have initiated the process, there is not overall control of the process – it occurs as a natural result of the coordination between the parties. In a collaborative process each actor is responsible for carrying out their role, but how they do so and how other actors carry out their own roles is “private business”, thus in the collaborative process, the behavior of each role is encapsulated.

The other style of process is the managed process – more like a factory. In a managed process there is an oversight role that coordinates the activities of subservient actors. For example, a quality control manager may coordinate the tests and activities within the QC department to a high degree of precision. Workflow systems are another example of managed processes.

Managed processes work well within a controlled domain, where the oversight role is practical and acceptable. In a managed process the oversight role typically has detailed information on the state and capability of each actor so that these “resources” can be efficiently managed. Managed processes work up to a certain level of granularity, but for large-scale processes and “B2B” processes, the concept of a unique manager is not

Copyright © 2002, Data Access Technologies, Inc.

practical or desirable. The manager imposes constraints on the actors that may not be acceptable and does not have the right to manage the resources of these more independent entities.

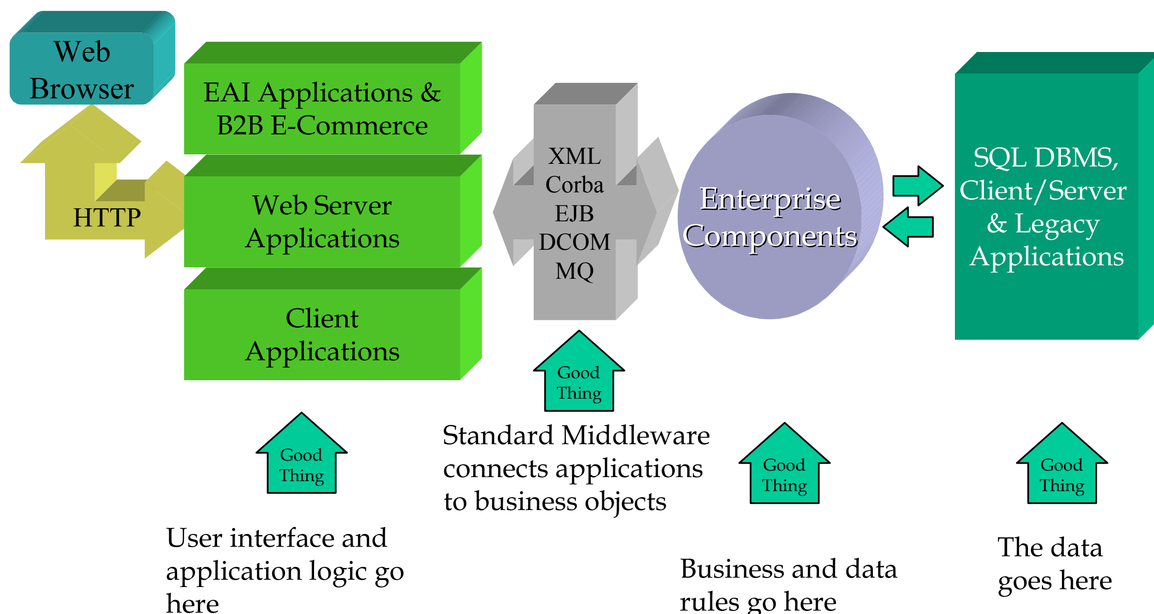
Since the roles of the collaborative business process are encapsulated from each other and any oversight role they are much less tightly coupled than managed processes, providing advantages to the system architecture. Interestingly, must managed processes can be described as collaborative processes – a practice which serves to keep the system and the enterprise more flexible. Relating this distinction to modeling – managed processes are typically depicted in activity diagrams while collaborative processes typically use collaboration or activity diagrams.

Where managed processes are required, they are typically inside of a larger collaborative process and treated as a single role within the larger system.

Services Architecture

Separation of information and business rules from the user interface, client application and technology details allows a much more flexible and integratable information system. Enterprise Components are used to capture the business and information rules, middleware (which includes Internet technologies such as XML) is then used to connect to applications, web servers or other applications. This service based architecture exposed through middleware is the gateway to the core of your business system.

The growth of XML over the internet has provided the capability for applications to integrate with clients and other applications across the globe, in one or multiple companies. We see XML and web services as an important technology facilitating global enterprise integration but it is only one possible media for such an architecture.



Copyright © 2002, Data Access Technologies, Inc.

The multi-tier approach is also important from a technology viewpoint. The work of an application can be served by one system or distributed over many servers, growing the application to “Internet scale”.

Event based computing

Event driven computing is becoming the preferred distributed computing paradigm in many enterprises and in many collaborations between enterprises.

Event driven computing combines two kinds of loosely coupled architectures:

- Event driven process architecture. This is a loosely coupled process architecture where the activities are not sequenced in traditional workflow fashion. Rather each participant in the process has autonomous responsibilities and performs those responsibilities on the basis of loosely coupled notifications, (in the supply chain world a.k.a. business signals).
- Publish and subscribe information distribution architecture. Publish and Subscribe is a loosely coupled mechanism for getting information from publishers to subscribers, while keeping the two independent of each other. Publish and subscribe is often implemented as loosely coupled, distributed components that communicate with each other through asynchronous messaging.

In event driven computing the most important aspect of the business process is the events that happen during its execution, and the most important part of the component-to-component communication is the notification of such events from the component that made them happen to all the components that need to react to them.

In the Enterprise collaboration architecture (ECA) we support both the definition of loosely coupled event-driven business processes, and the loosely coupled publish and subscribe communication between distributed components.

Neither the business world, nor the computing world, however, applies only one paradigm to their problem space. Businesses use a combination of loosely coupled and tightly coupled business processes and computing solutions deploy a combination of loosely coupled and tightly coupled styles of communication and interaction between distributed components.

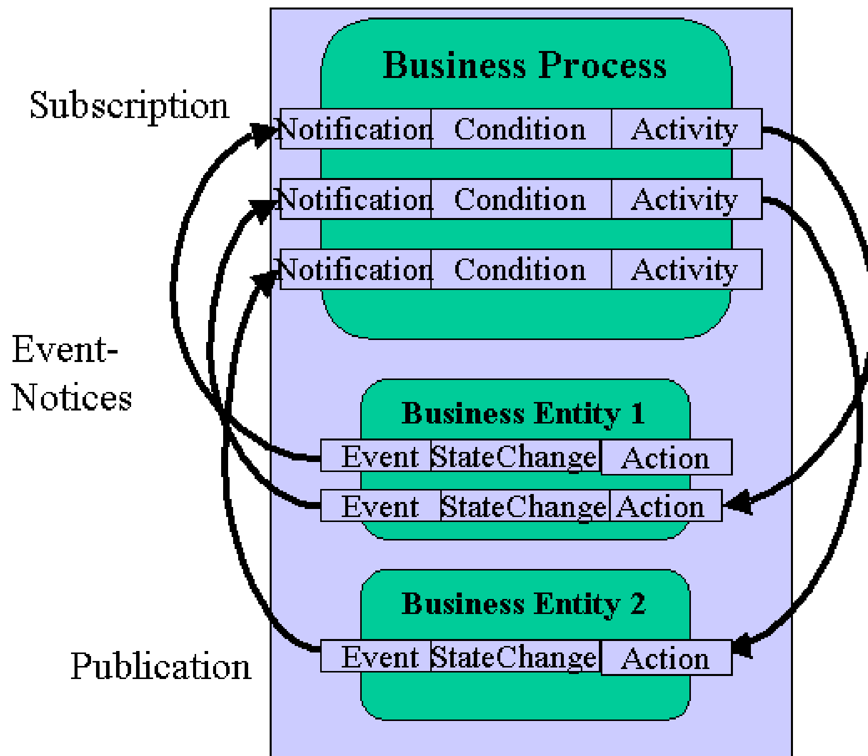
An ECA based business process can be defined as event driven for some of its steps and workflow or request/response driven for others. Likewise, distributed components in the ECA component profile can be configured to communicate with each other in a mixture of publish-and-subscribe, asynchronous Point-to-Point, and client-server remote invocation styles.

An event based business model is driven by business events. Whenever a business event happens anywhere in the enterprise, some person or thing, somewhere, reacts to it by taking some action. Business rules determine what event leads to what action. Usually the action is a business activity that changes the state of one or more business entities. Every state change to an Entity constitutes a new business event, to which, in turn, some other person or thing, somewhere else, reacts by taking some action.

The main concepts in event driven business models are the business entity, business event, business process, business activity and business rule.

This continuous, cyclical view of the interaction between these five business concepts can be depicted as follows:

Copyright © 2002, Data Access Technologies, Inc.



The lines between processes and entities flows through the event infrastructure (media) providing a level of indirection between producers and consumers of events.

Choice of interaction media

Interaction between systems components may be local or distributed, service based or event based. There are frequently legacy middleware systems in the corporate environment that will effect the choice of technologies.

While we will architect to be able to support a variety of such media, pragmatics suggest that a limited number be selected as the primary media within any one organization. Adapters may then be used to connect to other media types.

In some companies a single such media may be possible – a single "normal forma" for all interactions. In other companies multiples will be required for technical, historical and political reasons.

The services support media

There will be at least one distributed services interaction media, examples of such media are: Corba, EJB, ebXML, .NET and Web Services.

A primary selection criterion will be the requirement for distributed transactions. We will assume that at least one "non transaction" media will be used, as such media is required for B2B and other very loosely coupled interactions. There has been a strong

Copyright © 2002, Data Access Technologies, Inc.

move to utilize XML for such interactions. The weight of this industry movement and the resultant support should play heavily into choosing an XML based media as primary.

If distributed transactions are required, an appropriate support vehicle will be required, obvious choices being Corba – as a language independent solution to EJB for a portable language solution. It has been our experience that in many cases no distributed transaction media is required as all applications required for a specific action may be co-hosted on the same system.

Specific tradeoffs; Basic Web Services (Soap, WSDL & UDDI) has large-scale support and will undoubtedly be the bases for a large number of internal and external integrations with industry standard support. Web services will be directly supported by a large number of applications, frequently for free. On the down-side, basic web services lacks in the areas of security, integrity and robustness. Basic web services can be implemented very easily, even with little supporting infrastructure. If used as the primary services support media it will have to be "beefed up" and this will require putting together a few technologies.

EbXML is a more robust and secure infrastructure with a well defined business process, security profile, repository and messaging infrastructure. We consider ebXML to be second-generation web services. However ebXML does not yet have wide-scale buy-in or implementation. Due to this robustness, implementing an ebXML service will always require some "infrastructure" to do so which may have cost and portability considerations. If security is a major concern within the integrated environment ebXML should be considered. If ebXML is chosen, basic web services may also have to be supported for external integrations.

Besides the primary non-transactional media and a secondary transactional media (if required), additional media should be selected only as dictated by integration requirements. There will be a cost an overhead associated with each choice and additional media.

In summary;

- Choose a non-transactional XML media as the primary services interaction middleware such as basic Soap or ebXML.
- Asses the requirement for distributed transaction, if required, select an appropriate media such as EJB or Corba
- Add additional media only as required.

Event support media

The event "media", typically called the event service provides a publish/subscribe environment where by 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. In some cases the

Copyright © 2002, Data Access Technologies, Inc.

producers and consumers do not have to active at the same time, the event service will buffer requests between them.

Note that events are the preferred interaction mechanism for process monitoring, below.

Considerations of interest in selecting the event service are;

- Guaranteed deliver
- Deferred delivery
- Performance
- Granularity of subscription and capability to subscribe based on message data
- Scalability and distribution capabilities
- Availability of clients on the required platforms
- Integration with Java Messaging Service (JMS)
- Security

IBM MQ-Series has substantial support and penetration. The availability of the Java Messaging Specification (JMS) has allowed the introduction of newer systems based of this standard, such as Sonic-MQ.

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 that the message content may be informative as to the subscription, if allowed by the event service.

In summary, the systems architecture points relative to the event service are;

- Determine if an event system will be used
- Define or determine the structure of information to be carried by the events
- Select an event service
- Ensure that the component support environment will support both events and services

Platforms

The platform includes the hardware, operating system and core system software on which components execute. In many cases the platforms are pre-determined by legacy, politics and installed base. We will assume as a pre-requisite that there are multiple platforms involved in the same collaborative business process.

Typical server platforms are MVS, CICS, Unix and Windows-NT, but there may also be a variety of other platforms inherited through years of legacy. A distinction may be made between platforms that will just host a legacy application and wrapped and those that will directly host collaborative business components. A totally wrapped platform will be

Copyright © 2002, Data Access Technologies, Inc.

communicated with by some means (The worst case being screen scrapers) and the actual component will be hosted on amore open and accessible platform. The architect must determine which platforms will host components and which will simply be used "as is" by components. The support framework available on the legacy platform, the cost of development associated with that platform and performance considerations will determine this choice.

Side note: It is sometimes assumed that "offloading" an integration task is more efficient and scalable, but in some cases the overhead of distribution exceeds the overhead of the additional functionality. Testing or simulation may be the only way to quantify performance considerations.

While multiple platforms will be in use, a small number of such platforms should be identified as the primary integration platforms so that tools, expertise and infrastructure may be accumulated for them.

Consider the Java-JVM as an independent platform type that can hosted on multiple other platforms. This capability for a virtual platform can be quite useful and sometime solve performance problems. For example, the JVM platform can be hosted *inside* an Oracle DBMS or *inside* an AS-400 system making the connections to these back-end resources very efficient while retaining the advantages of the open and flexible Java environment.

In summary, the architectural tasks in relation to platforms are;

- Determine the required and preferred platforms
- Identify those that will host components
- Select a small number of platforms as the primary integration platforms, consider Java-JVM as a platform

Platform Support Facilities

Various media and middleware may dictate certain infrastructure on a platform. For example, use of Corba requires a Corba "ORB" be installed on the system and that component implementation adhere to a specific structure. Other media, such as web services, are defined completely in terms of their interface so do not require such infrastructure (But use of come common infrastructure may make development more efficient, as we will explore later).

Other than the basic messaging infrastructure, additional components and capabilities may be part of the framework into which solutions are provisioned. While it is theoretically possible to code an ebXML process "from the ground up", directly as part of a Cobol application – doing so is clearly inefficient.

So while it is architecturally sound to insist that no specific tools or infrastructure be required to support a particular media, it is pragmatically required to insist that tools, frameworks and services be available to make implementing business process components as simple as possible. The set of these is collectively referred to as the platform support facilities (since most such facilities are dependent, to some degree, on the platform). Some such facilities, such as Component-X may be portable across

Copyright © 2002, Data Access Technologies, Inc.

multiple platforms and shield the business logic from platform details while also making development and deployment more efficient and flexible.

It is important to note that there may be multiple layers of infrastructure and framework in any one-platform facility. For example the "stack" may consist of the operating system, DBMS, Language, Transaction Servers, Middleware, Application servers, Component framework and business logic components. A particular set of facilities will also have a corresponding set of tools that may or may not shield the developer from platform details. Component-X tooling does such shielding. Contrast this with the .NET environment that tightly couples tooling, facilities, media and the operating system.

As part of the systems architecture, the selection and integration of the support facilities is crucial to the functionality of the system as a whole.

Component Framework

The component framework is the technology framework into which the business logic for CBPs is instrumented. The component framework acts as a "container" for the component, controlling the component life-cycle and providing required system services. Component frameworks may be differentiated in their level of abstraction, services, portability and capability to support a variety of middleware.

Many component abstractions and their underlying framework are "flat". That is components may be built, deployed and combined – but they are not recursive, components can not directly be used to create other components. A key architectural point in the component framework is this recursive capability. Without a recursive component framework all components will have to be "hand coded" instead of assembled.

Many component frameworks are defined in terms of a particular middleware. These component infrastructures allow components to be built but they are limitied to the specific interaction protocols of the container. Both EJB and .NET components are tied to their infrastructure in this way, also, both are "flat" component architectures (.NET is somewhat less flat, but a different kind of component (COM Component) is used inside the .NET component.

The EDOC-Enterprise Component Architecture models the application as arbitrarily nested and interacting components which are not directly tied to a middleware. The Component-X XML-Component framework provides direct execution of this abstraction. This framework can operate on top of the .NET or J2EE frameworks providing greater flexibility and a higher level of abstraction. Another application capability expressed in EDOC-ECA is the concept of a multi-party business collaboration. Some component frameworks are limited to two-party interactions.

For environments where there is no existing framework or the framework is very technology specific, code generation can be used from the enterprise process model to provide a skeleton into which the business logic can be implemented, probably by calling a legacy application. For such wrapping purposes code may be directly generated to implement the desired protocol. There may be substantial engineering required to

Copyright © 2002, Data Access Technologies, Inc.

provide the wrapping support in a legacy environment, but once done it should provide functionality for most applications on that platform.

The component framework may also provide a library of built-in components to assist the development and adaptation process. Such capabilities may include document transformation, business logic routing, DBMS interactions, etc. After all, the leverage of a component environment comes from the reuse of existing components.

Summary of component framework architectural considerations;

- Is it flat or does it provide for recursive component assembly?
- Does it work with multiple platforms?
- Does it work with multiple middleware technologies?
- Does it isolate business logic from the interaction media and infrastructure?
- Is their code generation support for legacy wrapping?
- Does it support long-lived persistent processes?
- Does it support multi-party collaborations?
- Does it provide a useful component library?

Messaging

The messaging facilities are, of course, bound to the protocols to be supported. Messaging frameworks are frequently tightly bound to the protocols and middleware infrastructure. As such, it is best not to be bound directly to the messaging infrastructure for other than wrapped legacy systems (where there is little choice). These legacy systems will generally not be used to adapt multiple protocols.

The architectural considerations of the messaging framework include;

- Openness (As discussed)
- Performance
- Security
- Robustness
- Fall-over support
- Connection pooling
- Server farm support
- Logging support

Services

Business logic and technology components require various services from the environment, such as naming and directory, transactions and access to security

Copyright © 2002, Data Access Technologies, Inc.

information. The set of services required is, of course, application specific. These "container" services should be differentiated from pervasive services that may be provided by other components.

The architectural considerations of the container services include;

- Naming and "trading" to find external resources
- Ability to provide and configure resources
- Transaction support (from the components perspective)
- Security support (From the components perspective)

Automation

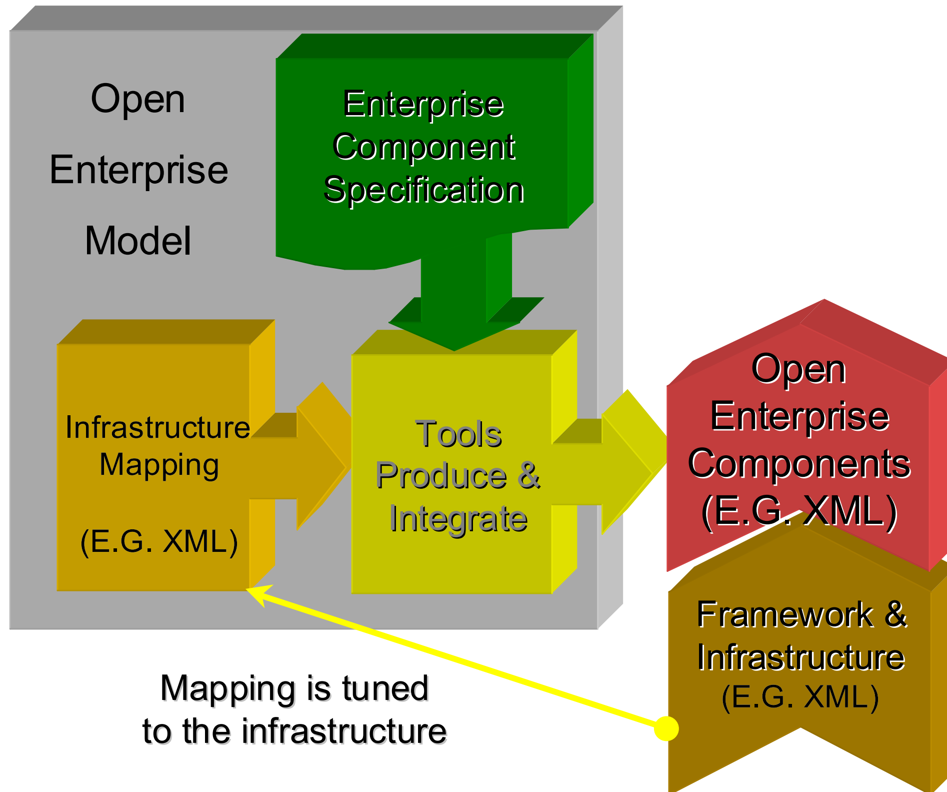
Automation is the ability of the component framework to be driven by and comply with the business process architecture. A substantial amount of information can be captured in the Enterprise Collaboration Architecture (ECA) Meta-Model. The information in this model, when applied to a specific component framework and platform, can either generate or validate the "boundary" of these components – the place where they interface to the outside. Where recursive component composition is used, automation can "glue together" multiple components to produce larger grain components.

Automation can also be used to generate the wrappers and adapters required for legacy systems integration and much of the artifacts required of middleware – such as IDL files, WSDL specification, ebXML business processes and CPPs.

Producing systems artifacts and using automation is less error prone and more consistent than attempting to hand-implement each artifact from a paper document. Automation is also able to tie together the various layers of the platform.

Copyright © 2002, Data Access Technologies, Inc.

Early attempts at automation were code generator options out of design tools. These generators were not sensitive to the target platform and lacked the capability to be parameterized so that engineering choices may be made in the automation process. Automation should be flexible such that it can be changed to suite the target environment



and provide parameterization capabilities.

Diagram of automation from specification to systems architecture

Automation of the systems architecture from the application architecture is the equivalent of automating the generation of assembler from high level languages. System automation is essentially a high-level compilation process that makes the application models part of the source code of the application implementation. Since substantial effort is made in designing the automation mappings, the result is often more efficient than custom coded pieces.

The architectural considerations for automation can be summarized as;

- A tight binding with the application Meta-model – in this case EDOC-ECA.
- Adaptability to various target platform packages
- Parameterization for capturing engineering choices
- Integration with design tools
- Integration with down-stream deployment and systems management
- Ability to "code into" generated template

Copyright © 2002, Data Access Technologies, Inc.

Application servers and containers

Application servers are "bundles" of infrastructure and framework for a specific platform, the most common examples today being J2EE and .NET, but include such veteran technologies as CICS.. Some application servers provide distributed object paradigm, others a web services paradigm and still others a messaging paradigm. Some are combining this together into one large package. The application server is essentially the combination of the above platform and framework considerations into one package. Others may provide a simpler infrastructure for the support of one or two interaction paradigms – such as those implementing web services servlets. Thus the application server may span the range from a servlet engine to a full application development and management environment.

The component framework may be tied to or provide a layer of independence over the application servers of choice.

Support for standard and legacy protocols

The concept of standard integration protocols is not new, web services is just a new cut at established standards like EDI and ASN.1. Within the systems architecture their must be nodes in the system of components that will be able to integrate with both legacy protocols and the enterprise standard media. Automated transformation between legacy formats and protocols allows such protocols to be integrated into the systems architecture with high-level tools and standard support infrastructure.

Architectural considerations;

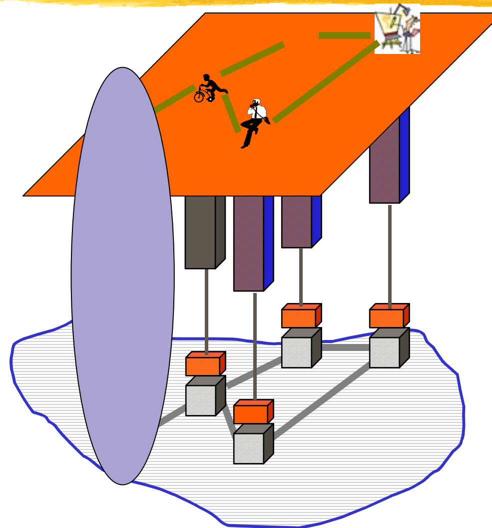
- Identify legacy protocols
- Establish metadata transformation with the specifications of these protocols
- Establish technology bridges for runtime integration
- Ensure that there exist platforms that are able to integrate with required intersections of protocols

User interface

The user interface is the human connection into the systems and application architecture. From the perspective of collaborative business processes, a user interface is simple a way for a human user to play one or more roles – a slice of the environment from the perspective of one actor in which case the system role is the gateway for that user into the business processes.

Copyright © 2002, Data Access Technologies, Inc.

Personal Information Environment



Environments fits
users mental model-
"My role in the enterprise"

or

My enterprises role
in the supply chain!

Copyright © 2001, Data Access Technologies, Inc.

Users view of the system from one actor

Since a single user may play multiple roles, their user interface becomes a composite of all the roles they are authorized to play.

Three-tier structure

The concept of users fulfilling one or more roles via the business process fits directly with the n-tiered information system structure; Back-end systems, business logic and user interface. The user interface layer relies on the middle tier – the services components, to provide the capabilities and business logic. The user interface is dedicated to making this role presentable, easy to use and efficient.

The two paradigms of user interface today are web-based and application based. Either approach required a similar architecture. A choice must be made when to use one or both.

Web-based user interface

The web based user interface has gained prevalence at a remarkable rate since the early '1990's. The web based (or thin client) interface assumes the existence of a browser on the client machine, perhaps augmented with standard plug-ins for special capabilities. All application logic, session management and business logic is performed by the web server applications. This has substantial advantages in accessibility, ease of maintenance and a graphical user experience.

From a systems architecture point of view the only thing that exists is the "web application", it is this web application – hosted on a server that contains the application

Copyright © 2002, Data Access Technologies, Inc.

logic. This web application must be able to play a role in the collaborative business process and integrate with at least one of the integration media – probably a web service.

While web-based interfaces have improved substantially, they are still not as productive for high-volume users as applications – but this distinction is narrowing. Also, web interfaces place more burden on the communications infrastructure.

An any modern system it is a given that some of the user interface will be web-based, the only question remaining is if there is also a requirement for application based user interfaces.

While there is substantial support for building web pages, web application building facilities that integrate with back-end web services are still new and a bit crude. Some level of web application framework and perhaps automation capability is likely to be required to reduce the workload of producing web application servers. Such packages are available from Microsoft, IBM and others. The choice of web application building environments should be considered in light of the enterprise standard protocols.

Architectural choices for web based user interface;

- Will web-based be used for some or all requirements?"
- What special requirements does the user interface place on HTML?
- What web-page construction kits are available that support web services?
- Does the user interface tool place special requirements on the web browser?
- Is the user interface subsystem unduly tied to the backend services?

Application based user interface

Application based user interfaces have the upper hand in providing a friendly and efficient user interface to the users roles. They have the disadvantage of requiring application code to be managed (and perhaps licensed) on the users computer. Applications with high-bandwidth UI requirements, such as CAD systems, must be application based.

While some user interfaces must be application based they may still separate application logic from back-end services. And, like the web applications, will require an application support framework tuned to being a web services client. Unfortunately web service client application frameworks are not mature and some engineering of the client environment should be expected.

The primary modern application user interface environments today are; Visual basic and Java beans. Visual basic is a more mature UI, while Java beans provides an open and portable environment. There is substantial tool and widget support for both.

Architectural choices for web based user interface;

- How many user interfaces will be applications?
- How well do the application UI building tools integrative with web services?

Copyright © 2002, Data Access Technologies, Inc.

- Does the tooling support and widget library cover the UI needs?
- Is it primarily a text-based or graphical application?
- Is portability important?
- How well do application integrate with web based user interfaces?
- Is the user interface subsystem unduly tied to the backend services?
- Are their facilities for remote management and upgrade?

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, timeout parameters on interaction may notify an application of a business partner does not responds. 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.

One mechanism for process monitoring is instrumented with events. Each component framework has an intrinsic capability to pose process interactions to the event infrastructure of the domain – when process 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.

Thus process monitoring is accomplished within the existing paradigm of loosely coupled roles and events.

Shared Data

Shared data in a DBMS was the systems integration strategy of the 1980's. While it would be nice for each role in an application to be completely independent, it is common for them to have requirements for the same data.

One approach is to have each application (or role) access the DBMS independently. While this works and is the basic for most client-server applications today – it results in the business logic associated with that information being duplicated across all such roles. Thus the shared data approach should be limited to legacy systems.

Copyright © 2002, Data Access Technologies, Inc.

Business Object Entity Components

The EDOC-ECA architecture provides for specific "Entity Manager" components that encapsulate the management of shared information behind services and event interactions. These entities become roles in the system – representing the shared data instead of a system actor, but in all other respects they are the same kind of component. These components wrap one or more DBMS tables to provide the required large-grain interactions.

Security

As systems become more open, security becomes more important and challenging. Optimally users would use a system with security being invisible unless they exceeded their privilege and this capability would be automatic once a capability was authorized. In practice, security is never perfect and always somewhat annoying. Today normal enterprise requirements may outstrip the security capabilities of web services and application servers. A security threat analysis should be made part of the systems architecture.

> Note: While we are familure with the concerns and structures, security is not our core competency – security professionals should be engaged to evaluate risk.

Role based security

The evolving security models have identified the "Role" as the primary unit of authority, capturing a set of capabilities together. The roles in a collaborative business process can directly correspond to roles in the security system and be automatically integrated into the authorization infrastructure.

Connection Security

The ability for a connection be secure is the most basic capability. In a web environment his is usually provided by HTTPS, which may be sufficient for low-security applications. Proprietary security infrastructures may be required for more stringent requirements.

Identity and credentials

Besides the security of a connection, the identity of actors in a role is essential to both authorization and application based security. Standard credential mechanisms are just becoming available but most are not available across a wide range of infrastructure or supported by standard application servers. Credential certificate authorities charge an annual fee per user, which may be prohibitive. The enterprise may require an internal certificate authority.

Authorization

Authorization is the management facility that authorizes certain actors (Knows as principles in security-speak) or groups of actors to play given roles. The authorization functionality should be able to span all business processes within the managed domain.

Copyright © 2002, Data Access Technologies, Inc.

Application security

Application security is functionality that must be enables or disabled based on business logic. Such application security requires that the business logic be able to have access to the identity of the principle and test if that principle may be in given roles. Most application servers are providing this functionality.

Systems management

The systems environment will encompass a large number of physical systems supporting an even larger number of processes, roles and technology components. An organized systems management architecture should be available across the set of infrastructures and systems. Various solutions such as Tivoli, Unicenter and SMS are products for such purposes. Care must be exercised to ensure that the platform stack is or can be supported by the systems management tools of choice. In the Java and MS environments a standard systems management API is available that is supported by most tools and application servers. This functionality may have to be extended to provide visibility of individual long-livened processes and components.

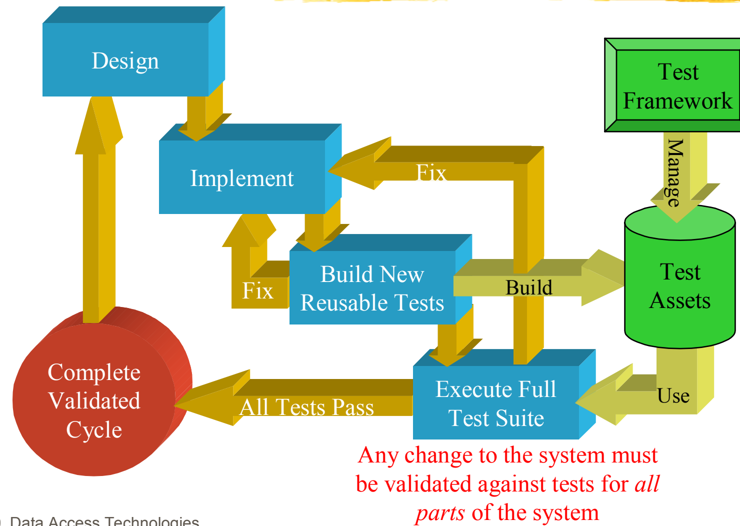
Validation and Quality Control

Problem

- Change and evolution is constant and desirable
- Changes and evolution cause unexpected problems in unexpected areas
- Programmers only test the part of the system they are working on (on a good day)
- As a result, unexpected problems emerge at the (expected) end of a project - causing systems to be late and fragile
- Testing needs to be an integral part of the development process
- Tests must be reusable assets
- The test suite for an application (the set of test assets) must validate any change to the system against the entire system
- Test capabilities must be built into the code
- To make this practical, tests must run under an automated facility

Copyright © 2002, Data Access Technologies, Inc.

Validation Cycle



Copyright © 2000, Data Access Technologies

Ubiquitous Services

Tying the "graph" of processes together is a set of services that must be able to have visibility and implementation across the platforms of interest. These services have been previously mentioned and include;

- Repository
- Name Service
- Certificate Authority

Copyright © 2002, Data Access Technologies, Inc.