GigaTS

Mapping the E-Business Frameworks Landscape

D 2.2.10

Telematica Instituut

Colophon

Date :	October, 2001
Version :	1.0
Change :	finalised
Project reference :	GigaTS/D2.2.10
TI reference :	TI/RS/2001/060
URL :	https://extranet.telin.nl/docuserver/dscgi/ds.py/Get/File-18715/
Access permissions :	GigaTS
Status :	Final
Editor :	Marc Lankhorst
Company :	Telematica Instituut
Author(s) :	Marc Lankhorst
	Johan Koolwaaij
	Diederik van Leeuwen

Synopsis:

E-business frameworks are building blocks that are part of the business logic or 'middle tier' of e-business applications. This document aims to create insight in the e-business frameworks landscape, and give a feeling for the different types of products and services available in the marketplace.

Giga Transaction Services

The project *Giga Transaction Services* supports organisations in the development of innovative transaction services. It does so with state of the art knowledge, methods and software tools that allow for effective development of new services. GigaTS takes the business perspective as a starting point, looking at networks of organisations and the way e-commerce technology can support them. Methods and tools are rooted in a combination of technological and business knowledge of currently available and future components and e-commerce applications. In this way re-use of components is promoted and time-to-market of services is reduced. Fast and effective design and introduction of e-commerce services is our central objective.

Giga Transaction Services has three main results:

- *technology scans and demonstrators*, aiming at (next generation) Internet technology as well as general e-commerce tools, components, frameworks and standards;
- *component libraries for e-commerce and electronic trade,* capturing the essential elements of business models, transaction scenarios and ICT components for e-commerce;
- a tool-supported environment for *Rapid Service Development* (RSD), a methodology for the design and development of transaction services, linked to software and e-commerce development tools.

These products are incrementally built. On a regular basis, results are made available to the public, and in seminars and on our Web site [GigaTS] we actively disseminate the results.

Giga Transaction Services is a knowledge-providing project under GigaPort Applications. The Telematica Instituut carries out the larger part of the work. The project emphatically tries to actively transfer knowledge and results to companies and organizations in pilot in a next generation Internet setting. Such pilots can consist of experiments with new services, the design or evaluation of e-commerce services and architectures, or company specific scans and demonstrators.

Acknowledgments

We thank Henk Eertink, Wil Janssen and Nicky Boertien for their valuable comments on earlier versions of this document.

Table of Contents

1 Introduction

In modern multi-tier Web application architectures, the ‘middle tier’ comprises the Web application server, on which the business logic, i.e. the main functionality of the Web applications is implemented (see Figure 1). At first glance, this middle tier might appear to be the simplest of the three tiers depicted here. However, it holds the business logic, which is the central part of the application and contains most of the functionality. In developing a Web application, getting this part right is perhaps the most important. It needs to be stable and reliable, and yet adaptable to changing requirements and circumstances.

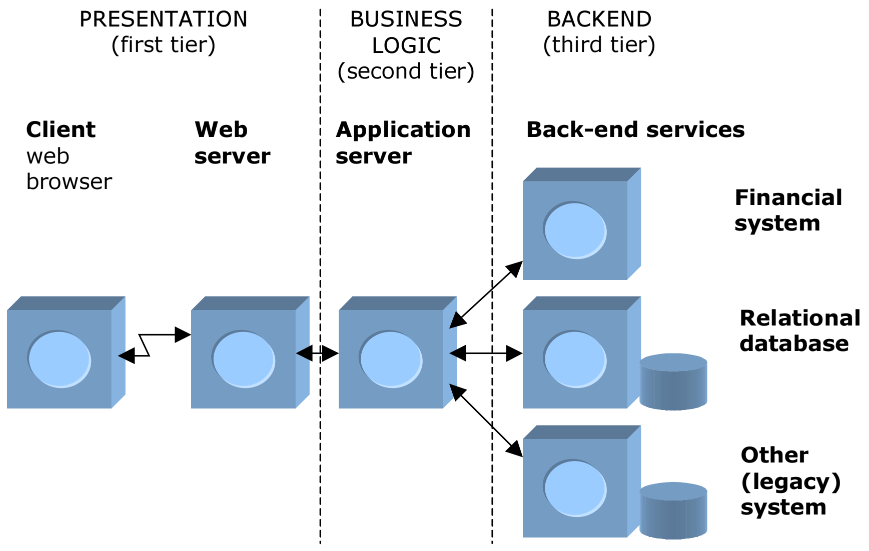

Figure 1: Multi-tier Web application architecture.

To aid the developer in quickly creating quality applications on this middle tier, many vendors offer what might be called ‘e-business frameworks.’ However, the products sailing under this banner are very diverse. First, there is the *scale* of the objects or components. On the one hand, we have frameworks that contain large-scale business components comprising the functionality of an entire application, e.g., a customer relationship management (CRM) component. On the other hand, frameworks exist that provide small-scale business objects, e.g., an orderline object, or support for applications such as database transaction management or security services.

Another important dimension in describing e-business frameworks and their role in application development is that of *infrastructure vs. business content*. We distinguish different levels at which e-business frameworks are used; the emphasis in these levels shifts from infrastructure to business content:

- middleware: e.g. an application server on which the Web application runs;
- application support: low-level support for applications, e.g. database access;
- business objects: source-level entities that have a concrete business meaning, e.g. an order;

- business components[1]: independent units of deployment that can be composed to form an application, but reside within the span of control of a single organisation, e.g. a CRM component;
- e-business applications[2]: entire applications serving a business goal and possibly spanning multiple organisations, e.g. an e-marketplace application that integrates suppliers, buyers, and intermediaries.

These levels or layers are depicted in Figure 2.

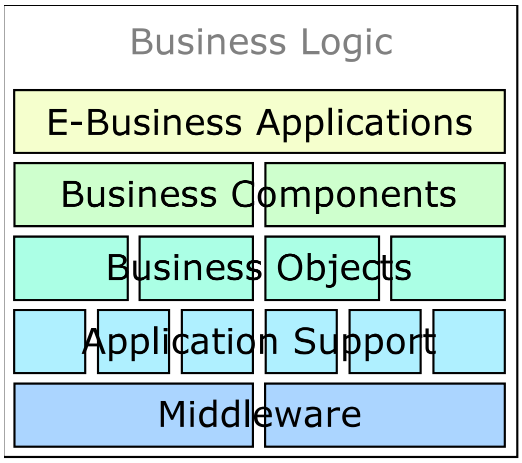

Figure 2. Layers in the Business Logic tier.

Strictly speaking, e-business frameworks cover the business logic of an application, and Web application middleware is not part of the logic itself but provides necessary support. However, there can be close relation between objects or components on the higher levels and the supporting middleware; several business logic solutions require their own middleware, and conversely, standards exist which ensure independence from specific vendors. We might thus consider the independence from specific middleware technology to be a third dimension. Furthermore, we see a trend that functionality formerly implemented in the application support layer is generalised and migrates to the middleware layer. For the purposes of this analysis we therefore include Web application middleware.

Communication between the components that constitute a distributed e-business application will be supported by another type of middleware. This could e.g. be messaging middleware, or XML/SOAP for Web services. Support for this might be incorporated in the web application middleware (i.e., the application server) or might be separate. In the stack of Figure 2, we do not show this type of middleware in a separate layer.

The scale of components or objects is highly correlated with these layers. On the support level, objects are typically small-scale. Business objects contain more functionality, and will

[1] The term “business component” is sometimes also used for non-software entities that comprise a well-delimited, self-contained actor or process in the application domain.
[2] Note that we only add the “e-“ before business applications; this is the level where organisational boundaries are transcended and the business turns into a true e-business.

therefore be larger. On top of that, business components have a more independent character, which adds to their functionality and size, and may comprise several business objects. And finally, entire e-business applications are of course the largest of these, and may be constructed from a number of business components and objects.

In the following sections, we describe the different levels in more detail and give concrete examples of products in each category. The levels are not always clearly separated. For example, a small e-business application might be used as a business component in another context. Furthermore, many products provide components at more than one of these levels. It is not our goal to define crisp, mutually disjunct categories, but rather to create insight and characterize the different types of products and services available in the marketplace.

We will illustrate these levels with the Queten demonstrator case, which is used as a running example in the GigaTS project [BFJ+00]. It concerns a chain of companies consisting of vegetable growers, wholesalers, an auction, carriers, and banks, their interactions and the transaction services used for supporting these interactions (Figure 3). The demonstrator is built around the auctioning functions of the Queten case. These functions have been implemented using a multi-tiered Web/WAP architecture.

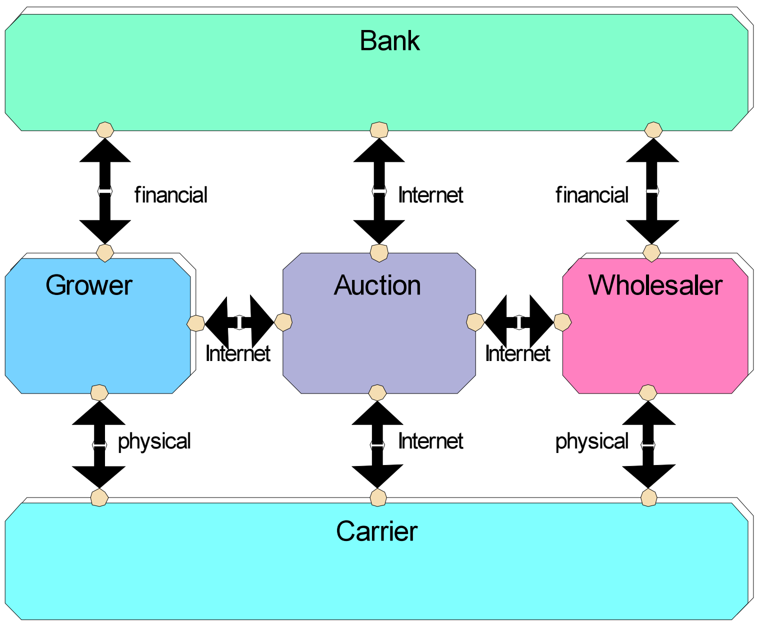

Figure 3. Actors in the Queten case.

2 Web Application Middleware

2.1 Description

Web application middleware is the foundation on which all Web application functionality is built (see Figure 4). It is generic functionality embedded in a platform that at least supports a component model such as Enterprise JavaBeans or COM and in most cases provides additional services such as e.g. persistency, naming & directory services, transaction support, and database connectivity.

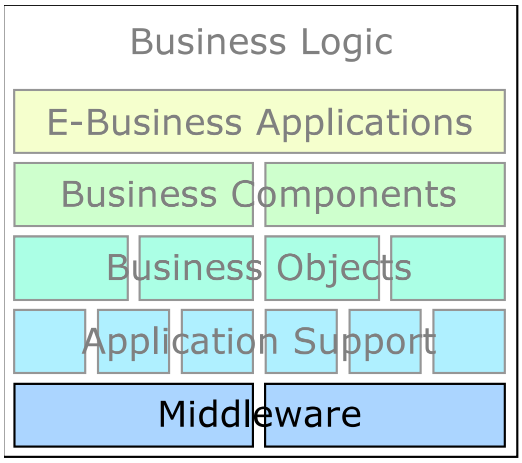

Figure 4. Web application middleware, the foundation of Web applications.

The Java world

Many of the available Web application servers implement the run-time environment as described in Sun's Enterprise JavaBeans (EJB) specification [EJB]. Following the definition as in [Mons00], EJB is a standard server-side component model for component transaction monitors. An EJB is a simple class that provides two types of methods:

- business logic methods. A client program calls these to interact with the business data and processes.
- lifecycle methods. These are called to manage the EJB on the application server and provide e.g. functions for starting, terminating and storing the EJB.

EJB objects 'live' in the business logic layer, and are implemented in Java. Once implemented, they can be expected to work in any EJB-compliant application server. Most current application servers conform to the Java 2 Enterprise Edition (J2EE) standard [J2EE], which guarantees this interoperability.

The EJB specification defines two different types of Beans:

- Entity Beans, which are used for modelling business concepts and can be persistent;
- Session Beans, which are responsible for managing processes or tasks.

Table 1 compares Entity and Session Beans.

Entity Bean	**Session Bean**
Represents data in a database	Fields contain conversation state
Shares access for multiple users	Handles database access for client
Persists as long as data exists	Life of client is life of Bean
Transactional	Can be transaction-aware
Survives server crashes	Does not survive server crashes
Fine-grained data handling	No fine-grained data handling

Table 1. Entity beans vs. session beans.

EJBs are not the only component model in use on Java-based application servers. Oracle, for example, has its own component framework, the Business Components for Java [Oracle_BCJ], which is different from the EJB model and tailored towards accessing databases. IBM's SanFrancisco framework is based on JavaBeans, not EJBs. Note that 'Enterprise JavaBeans' and 'JavaBeans' are unrelated. Both of them are component models, but the JavaBeans model is used to define Java objects in a generic way, while the Enterprise JavaBeans model is used for distributed objects in a specific transactional environment.

Next to a component model, application servers provide a host of other functionality. In the Java world, common functionality — integrated in the J2EE environment — includes:

- the Java Naming and Directory Interface (JNDI), offering a unified interface to multiple naming and directory services;
- Java Database Connectivity (JDBC), providing access to databases, spreadsheets, and flat files;
- the Java Transaction Service (JTS), for database transaction management;
- the Java Messaging Service (JMS), for building message-based applications.

The Queten demonstrator is built using the Enterprise Java Beans platform, and is deployed on an IBM Websphere Application Server [IBM_WAS]. The three tiers in the web-based Queten auction application consists of the clients, the EJB application server (the grey box in the middle), and the database, as shown in Figure 5.

Clients of the Queten auction application, e.g. PCs, WAP phones or PDAs, communicate with the application server via a web server, using high-level and platform-independent calls. The user interface is a set of HTML/JSP/WML pages, processed by a browser, that get input from and show information to the user. Behind the HTML/JSP/WML pages is a servlet that passes data between the client and the EJB container. This container holds the EJBs that implement the business logic of the application and communicate with the database.

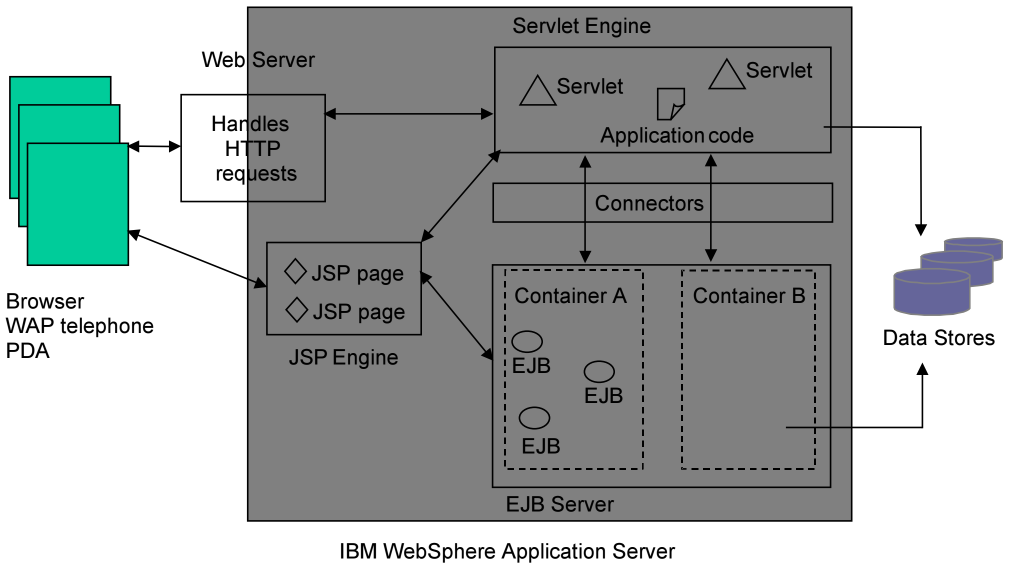

Figure 5. Queten auction application and IBM WebSphere Application Server architecture.

The Microsoft world

The Microsoft model for components is the Component Object Model (COM) [Micr00]. This is a binary component model, as opposed to Java's source-level model. Not only server-side components are covered by COM, but also compound documents, user interface controls, etc. The Distributed Component Object Model (DCOM) enables components to communicate directly over a network, either over Internet protocols such as HTTP, or over other network transports. Microsoft Transaction Server (MTS) provides server-side component services such as a TP monitor, ORB, connection management, security services, and component management and configuration. The unification of MTS and COM/DCOM resulted in COM+. The architecture of Microsoft's current server-side technologies is described by Windows Distributed interNet Applications Architecture (DNA). This provides a multi-tier development model, and services for building enterprise-level systems on the Windows platform. It consists of three layers: data services, application services, and presentation services. The application services form the middle tier, where DCOM, MTS, the Internet Information Server (IIS), and Active Server Pages (ASP) are used to handle business logic.

At the middle tier, Microsoft's products have a rather less clean separation between business logic and presentation than the Java world has. In ASP pages, code, contents and presentation are mixed up, leading to implementations that are hard to extend and maintain. This problem is supposed to be alleviated with ASP+, the successor to ASP included with .NET (see below).

The most recent development at Microsoft, succeeding DNA, is .NET [.NET] [3]. Microsoft is betting the farm on Web services, and .NET is the platform that supports these and aims to integrate all Microsoft's products. The .NET platform introduces a new language-neutral component model that is compatible with COM. The .NET runtime environment contains a Common Language Runtime that supports components encoded in an intermediate language (IL), akin to Java's bytecode. .NET components interoperate via the XML-based Simple Object Access Protocol (SOAP) [SOAP]. For more information about Web services and their role in application integration, see [FeKL01].

2.2 Examples

Many examples of Java-based application servers can be given. There is no real difference in basic functionality; all servers listed are J2EE-compliant. This list just mentions some of the most common ones and is by no means exhaustive:

- Allaire JRun [JRun];
- BEA Weblogic Server [Weblogic];
- HP Bluestone Total-e-Server [Bluestone];
- IBM WebSphere Application Server [IBM_WAS];
- iPlanet Application Server [iPlanet];
- JBoss Server (open source) [JBoss];
- Oracle9i Application Server [Oracle_AS];
- Silverstream Application Server [Silverstream].

There are several other, non-Java application servers. A notable example is Macromedia's ColdFusion [ColdFusion], which has its own server-side scripting language, called ColdFusion Markup Language (CFML), and has its own set of development tools.

On top of .NET, Microsoft offers many server products [.NET] several of which are targeted towards the business logic tier:

- Application Center to deploy and manage highly available and scalable Web applications;
- Commerce Server for quickly building scalable e-commerce solutions;
- Mobile Information Server to enable application support by mobile devices like cell phones.
- BizTalk Server for XML-based data transformation and orchestration of business processes across applications and organizations;

Many of these products offer more than just Web application middleware, but important parts of their functionality fall within this category.

[3] .NET is currently in beta; the final release is due in 2002.

3 Application Support

3.1 Description

The application support layer performs basic functions that are not tied to specific business applications but have a more generic character. A typical example would be a login function, which can be used in all business applications that require the identity of the user to be established, but has no business significance in itself.

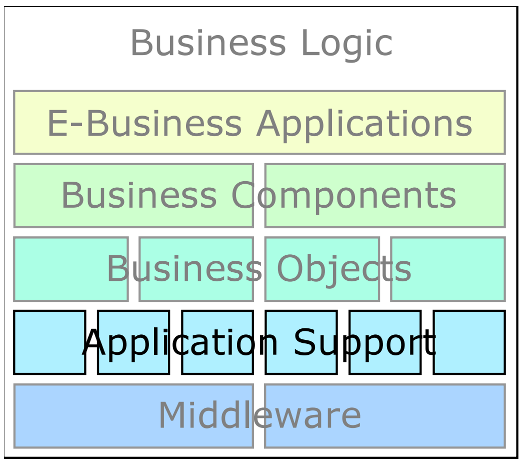

Figure 6. Application support.

Application support functions are often relatively fine-grained. Some typical classes of functionality are:

- Database access, e.g. connection pooling; extensive support for this can for example be found in Oracle9i Application Server [Oracle_AS];
- Data transformation, e.g. for linking legacy databases to new applications; such transformations can for example be defined in BizTalk Mapper [BizTalk];
- Generic presentation functionality, e.g. for multi-channelling purposes; support for this is an important element of IBM's WebSphere Everyplace Suite [IBM_WES];
- Session management;
- Security, authentication & authorisation;
- Installation & configuration.

The Queten demonstrator uses an application support framework developed in the GigaTS project, which is partly based on several other open-source frameworks. It provides e.g. a login service for generic authentication, services for session management, and database connection pooling. This last function is an illustrative example of typical generic application support functionality. To optimize database access, a pool of open connections to the database is managed by a Session Bean. If a business logic EJB needs access to the database, it requests an open connection from this connection pool bean (Figure 7), thus avoiding the overhead of opening and closing a connection for each separate database access. The connection pool bean manages these connections, opening new ones as required, closing connections that have timed out, and protecting the database by performing authentication.

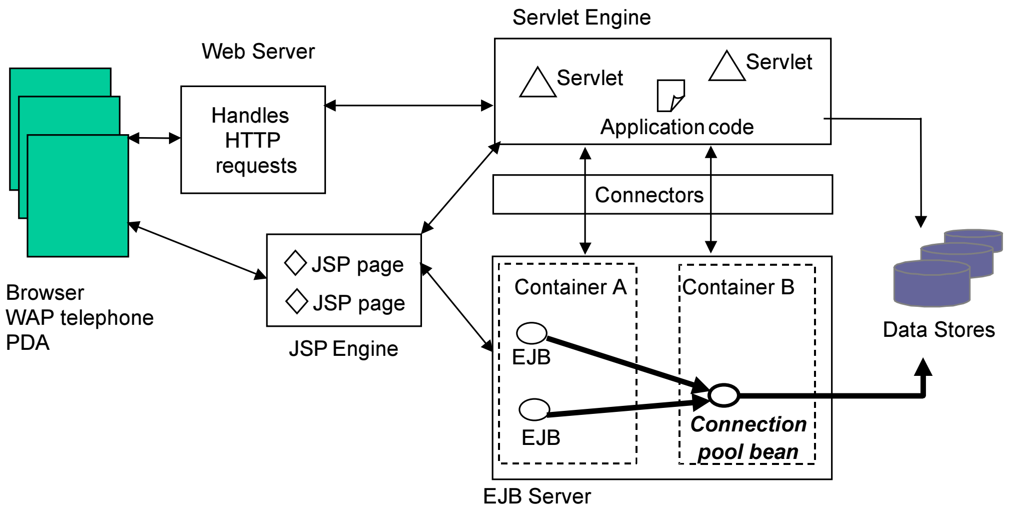

Figure 7. Connection pool bean.

3.2 Examples

Some examples of frameworks providing application support are:

- IBM's SanFrancisco [IBM_SF], in which the Foundation layer manages e.g. object ownership and persistence, distribution, database transactions, queries, naming, notification, and security. Furthermore, it provides the base classes for the Common Business Objects.
- The Foundation services of Evergreen's Open Commerce Framework [Evergreen], comprising e.g. services for session management, security, licensing, data translation, transactions, and management.
- The Expresso Framework of Jcorporate [Jcorp], which includes components for security, database connection pooling, logging, event notification, email connectivity, job control, etc.

3.3 Positioning

Building applications using this type of application support objects helps the developer in quickly setting up their basic functionality. Constructing low-level functionality such as database access and security can be greatly facilitated with application support frameworks. Because of their reliance on specific lower-level functionality, support frameworks are in most cases tied to e.g. a specific type of application server.

There is typically no support for business-oriented functionality in these frameworks. In case the application to be built is relatively special-purpose, this need not be an objection, because it would be hard to find reusable higher-level objects and components anyway. But if the application is more standard, say a simple Web shop, then it might be advisable to use larger-grained, higher-level objects and components, either exclusively or in conjunction with an application support framework.

4 Business Objects

4.1 Description

Business objects solve recurring business problems and represent entities that are meaningful from a business point of view. They are designed to support business processes, can be grouped into business components, and might be combined and specialised by means of object-oriented operations such as object composition and inheritance. Business objects themselves are not meant to be deployed independently, or be directly exposed to the outside world; in our definition, this role is reserved for business components.

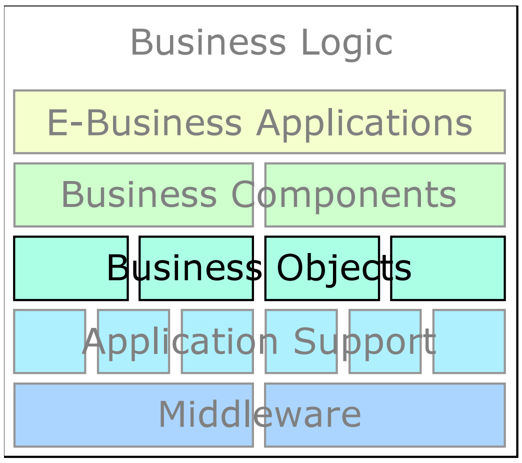

Figure 8. Business objects.

From the viewpoint of data and process modelling, we may distinguish three classes of business objects:

- Entity objects: these represent real world objects that have a specific business meaning, e.g., an order object, and provide encapsulation of the data stored about this real world object. Typically, entity objects can be implemented using entity beans (see Section 2) and their content is stored in some database.
- Association objects: these represent the associations between real-world objects and maintain the relations between the entity objects defined above, preserving consistency as the entity objects change. They could also be implemented using entity beans, and might or might not have a direct database representation. A typical association object might be responsible for the relationship between orders and customers. The responsibility for this relationship might be pushed to the order and/or customer objects themselves, but this could lead to undesirable overhead (both orders and customers must maintain consistency) and create an architecture with unwanted dependencies (if I add an account manager object that relates to customers, I might have to change the customer objects as well).

- Process objects: these perform functions related to the process supported by the e-business application and are responsible for the process state. Typical process objects can be built using session beans (see Section 2). Representative examples would be order entry objects that control the process of entering orders, or workflow objects that control the activities within a business process.

A typical example of a business object in the Queten demonstrator is the BillBase. The BillBase business object, implemented as an EJB Entity Bean, is part of the Billing component, which provides generic facilities for billing customers. The BillBase implements methods for creating bills, searching the bills database by buyer, date or bill number, and other related functionality. It is not specific to the Queten demonstrator, but might also be used in other components or applications that require billing functionality. It serves a concrete business goal and can only be deployed in the context of the Billing component; it therefore clearly qualifies as a business object.

4.2 Examples

A good example of a framework providing business objects is IBM's SanFrancisco [IBM_SF]. Its Common Business Objects layer consists of three groups of objects:

- General business objects that are common across business domains, e.g. Address, Company, Business Partner, Currency, Payment Method, and Project;
- Financial business objects that provide a way for financial status to be affected from outside of the financial domain, e.g. Bank Account, Interface to Banks, and Invoicing;
- Generalised mechanisms that are used to solve problems common across business domains, e.g. Extensible Item, Keys, Validation Result, and Life Cycle.

The first two groups fall into our category of Entity objects; the last group contains Process objects.

Another example is offered by workflow application vendor Staffware. Staffware Enterprise Objects (SEO) [Staffware] is an object-based application toolkit for building enterprise workflow applications by providing access to Staffware Workflow functionality through an object model.

4.3 Positioning

Business objects are especially suited if one needs to build an application for which existing large-grained components are not really suited, but for which the business domain is well-covered by standard business concepts. Business objects must be complemented with application support, and most business object frameworks also offer such supporting objects. It is important to note that the choice for a particular suite of business objects implies the use of specific underlying technology such as eg. a J2EE-compatible application server.

Building an application from business objects might also be the right choice if e.g. legacy databases need to be accessed. It might be the case that the layout of the database cannot be adapted and does not match the concepts of higher-level business components, but can be mapped onto a given set of business objects.

An important limitation of business objects is that they must match the structure of the application domain. For relatively standard domains and applications, such as webshops or back-end financial systems, this is usually not a problem. But for these relatively standard problems, pre-packaged solutions often exist, obviating the need to build an application from business objects. Furthermore, business objects are source-level entities, and composing them requires serious programming.

5 Business Components

5.1 Description

A business component is an independent unit of deployment. It has a 'life of its own' and its existence does not rely on the existence of other components in the same way that business objects rely on each other. Business components can be composed into applications, but unlike business objects, they are typically not specialised and organised into inheritance hierarchies. As opposed to e-business applications, business components do not have a cross-organisational character.

Furthermore, a business component is meant to be exposed to the outside world: it may be visible to and used by external parties that are beyond the span of control of the organisation that manages the component. To that end, a business component has to have clearly defined and published interfaces, which can be used by others to access the service the component provides. The interface might be defined in e.g. WSDL [WSDL] and published via a central registry, e.g. UDDI [UDDI].

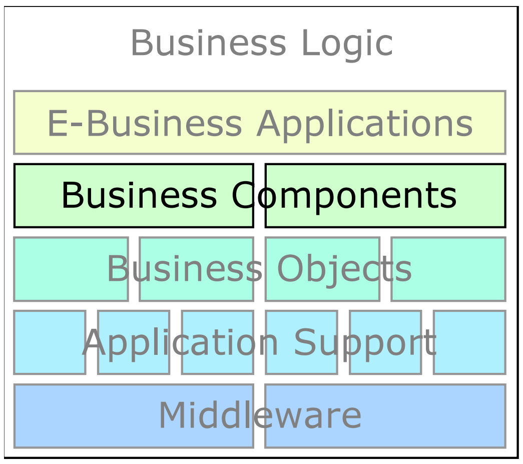

Figure 9. Business components.

A business component can be composed from business objects and may use other business components to provide the service it advertises. Consequently, business components in general have a larger granularity than business objects, but this is not an absolute distinction; large business objects may exceed small business components in size. Futhermore, business components typically represent a vertical slice of functionality, crossing multiple tiers of the n-tier Web architecture, e.g. comprising GUI, business logic, and database functionality (see also [HeSi00]). Typical business components may provide services such as catalogues, order processing, inventory management, or payment.

A typical business component in the Queten demonstrator is the Auctioning component. This comprises functionality for offering items in an auction, bidding for these goods, closing the auction, and for managing the database of items that are auctioned. The Auctioning component runs at the Auction actor (see Figure 3) and provides its functions via a Web interface to the buyers and sellers, i.e., the wholesalers and growers. These have there own components that communicate with the Auctioning component, and they all communicate with a component that performs the financial settlement. In simplified form, the Auctioning component and its context are illustrated in Figure 10.

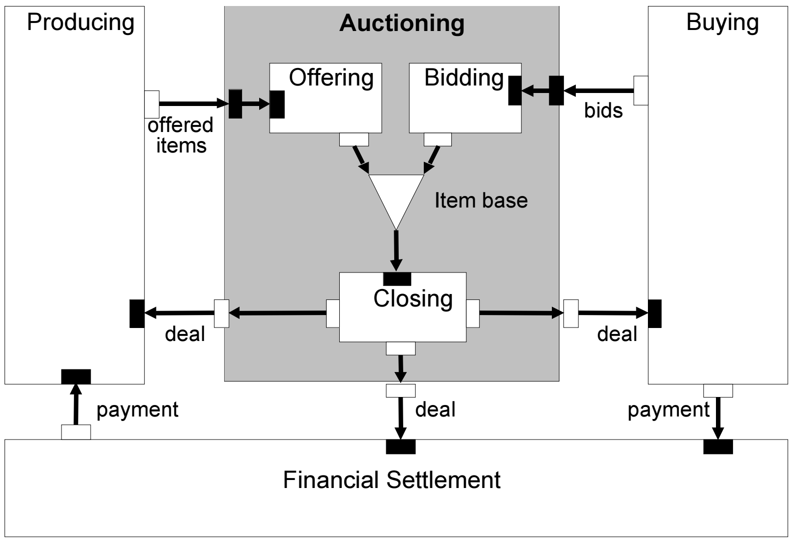

Figure 10. Queten’s Auctioning component in context.

Web services are a new model for using the Web to provide the functionality typically implemented by a business component. A Web service is a collection of functions that are packaged as a single entity, perform a specific business task, and are published to the network for use by other applications. Web services can be seen as building blocks for creating open distributed systems, which allow companies and individuals to make their digital assets available to the global community at large in a simple and effective manner. The spectrum of possible Web services ranges from simple single functions such as specific financial calculations to entire processes such as airline ticket reservation. Web services can be reused across multiple applications, allowing fast and efficient Web application development. An extensive discussion of Web services, their merits and drawbacks is presented in [FeKL01].

5.2 Examples

Several vendors offer relatively large-scale business components. Some examples:

- Allaire's Spectra [Spectra] is an application framework consisting of a set of components covering three major areas: content management, e-commerce and personalization. Spectra requires Macromedia’s ColdFusion application server [ColdFusion] (see also Section 2.2).
- Compoze Software [Compoze] has the Harmony Component Suite, which is geared towards CRM applications. It contains components for scheduling and workflow, task management, Webmail, contacts and distribution lists, and discussion forums.

- Diamelle Technologies [Diamelle] offers their Enterprise Business Components, containing e.g. components for user and profile management, catalogue management, shopping cart, order management, billing and shipping.
- Evergreen [Evergreen] has its Open Commerce Framework. This provides components for merchandise, searching, shopping cart, pricing, inventory, currency conversion, order processing, shipping and handling, tax, and payment.
- EzCommerce provides the EzCommerce Suite [EzCommerce] comprising a number of large-grained business components such as a sales component, a purchasing component, a product catalogue, and a customer component.
- IBM provides the Business Components as part of the WebSphere environment [IBM_BC], containing an assorted collection of components for order capture, text analysis, bank teller, and development & deployment. The upper layer of IBM's SanFrancisco [IBM_SF], the Core Business Processes, holds business components for general ledger, accounts receivable/payable, warehouse management, and order management.
- OhioEdge [OhioEdge] offers a suite of J2EE components for customer relationship management, workflow, product management, communication, and sales.
- OMIX [OMIX] provides OMIX E-Commerce Infrastructure, based on a J2EE component set and offering e.g. logic for product catalogues, shopping carts, user accounts, credit card validation and transaction, clearing, and order processing to fulfilment centres.
- Unify offers the Unify eWave Commerce suite [Unify] which contains a set of 17 EJBs subdivided in 4 classes: foundations services mostly used for configuration and setup (e.g. log and session services), manager services for management of product, order, and event data (e.g. order manager and site property services), and commerce services, which provide the merchandise interaction and order placement functions (e.g. membership services, merchandise services, shopping cart services, and payment services).

5.3 Positioning

The main advantage of using these large-grained business components is speed of development and deployment. Entire applications can be built from scratch with little effort, but the price to pay is limited flexibility: the organisation must adapt to the components used, and not vice versa. However, business components still offer much more flexibility than commercial off-the-shelf products, allowing the application builder to script together a selected set of components, design custom interfaces, and in general allowing a profitable mix of standard and custom elements. They do not require the programming efforts associated with building an application from business objects.

Integrating legacy applications with these components can be a difficult task, as they often bring their own data management and offer limited or no access to existing databases. Especially in cases where the e-business application needs to cover many aspects such as sales, CRM, inventory management, and logistics, for which a host of existing applications need to be incorporated, high-level components may lack the adaptability required for successful integration.

6 E-Business Applications

6.1 Description

We define an e-business application as business-oriented software that integrates the applications of multiple organizations[4]. It can be built from e-business components, which are connected by an enterprise application integration infrastructure that provides messaging, connectivity and security services [Pan99]. In most cases, the Internet will provide this communications backbone, but more traditional EDI solutions based on value-added networks are still quite common.

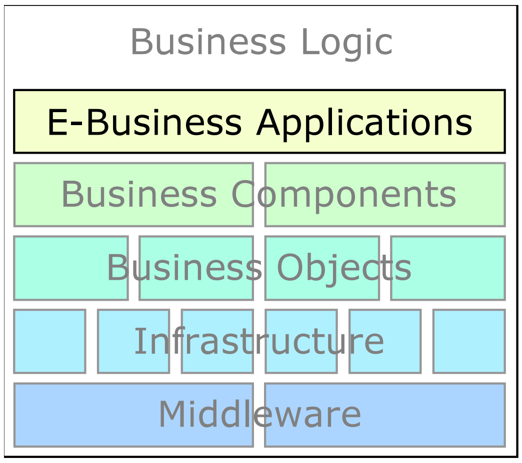

Figure 11. E-business applications.

In the Queten demonstrator case, the e-business application is the collection of business components that together realize the auctioning process. The Auctioning component described in section 5.2 is part of this application, and also the corresponding components at the growers, wholesalers, and banks. This is illustrated in Figure 12.

[4] For a discussion of intra-organisational integration, we refer the reader to [KoSt00], which deals with enterprise application integration (EAI).

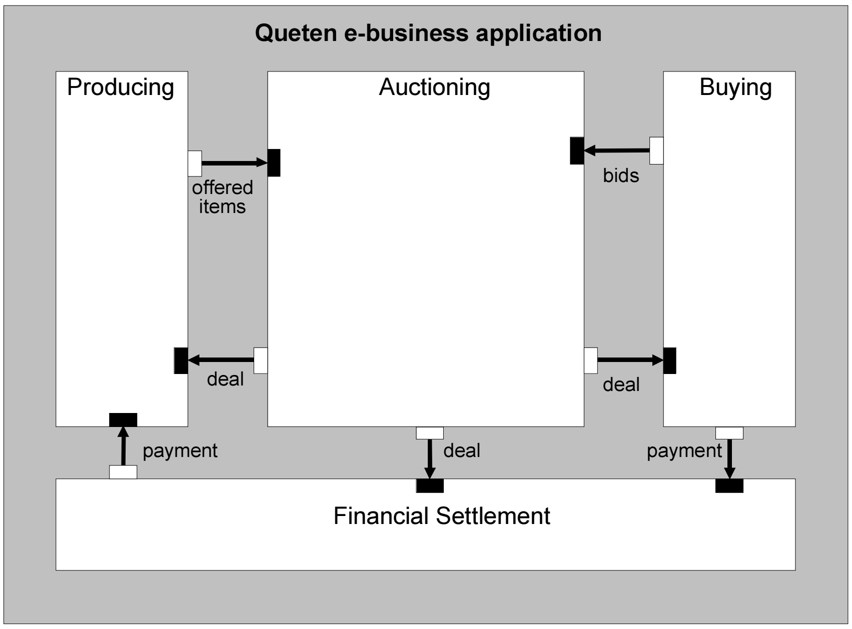

Figure 12. The Queten e-business application.

For the realization of loosely coupled e-business applications, the emergence of Web services is a promising development. Transactions can be described, published, discovered, and invoked dynamically in a distributed computing environment. Traditionally, applications were quite tightly coupled, using a more or less fixed linkage via e.g. remote procedure calls or Java RMI [RMI]. Middleware solutions such as CORBA [CORBA] and messaging middleware have changed that, but are difficult to deploy in distributed environments that have no central control, and may incur considerable overhead at runtime.

6.2 Examples

Several initiatives, standards, and products for inter-organisational e-business applications are arising. As this is an immature field, it is hard to assess the viability of these initiatives. We only outline some of the more prominent ones.

6.2.1 E-business application standards

RosettaNet [RosettaNet] is a consortium of information technology, electronic components and semiconductor manufacturing companies that aim to create and implement industry-wide, open e-business process standards. It has already defined standards on several levels:

- Partner Interface Processes (PIPs) define business processes between trading partners.
- RosettaNet dictionaries provide a common set of properties for PIPs. The RosettaNet Business Dictionary designates the properties used in basic business activities. RosettaNet Technical Dictionaries provide properties for defining products.
- The RosettaNet Implementation Framework provides exchange protocols for quick and efficient implementation of PIPs.

ebXML is a set of specifications for electronic interoperability, allowing businesses to find each other, agree to become trading partners and conduct business. The ebXML architecture provides [ebXML]:

- A way to define business processes and their associated messages and content.
- A way to register and discover business process sequences with related message exchanges.
- A way to define company profiles.
- A way to define trading partner agreements.
- A uniform message transport layer.

In the ebXML view, a business process consists of transactions and collaborations. For example, the business process Procurement consists of Create Long Term Contract, Forecast Component Requirements, Send Planning Documents, Place Order, Ship Materials, and Arrange Payments. Hence, the specification schema supports the specification of business transactions and the choreography of business transactions into business collaborations. Each business transaction can be implemented using one of many available standard patterns. These patterns determine the actual exchange of business documents and business signals between the partners to achieve the required electronic commerce transaction. For more information on ebXML, we refer the reader to [JaKS00] and [FeKL01].

6.2.2 E-business application products

Many vendors are active in the e-business applications market. BroadVision, for example, provides many products, such as Business Commerce for b2b marketing, sales and service, Retail Commerce Suite for online shops, and MarketMaker for building e-marketplaces [Broadvision]. Hewlett-Packard has its NetAction Internet Operating Environment, containing HP Web Services Platform [HP_WSP] (the successor to e-Speak [eSpeak]), which enables business interactions through the ad hoc discovery and interaction of Web services. It is built in Java and supports WSDL [WSDL] and UDDI [UDDI] for service description and discovery. IBM offers WebSphere Commerce Suite [IBM_WCS], an integrated e-commerce solution for b2b, b2c and e-marketplace business models. It offers all kinds of e-commerce functionality, and provides e.g. links to the Ariba Commerce Services Network [Ariba].

The traditional ERP, CRM, and workflow product vendors are also rapidly expanding in the field of inter-organisational applications. SAP AG, for example, has its Internet-enabled mySAP.com platform [mySAP], which provides portal-based access to most of SAP's product suite, and uses a subscription-based business model. It offers end-to-end solutions for e.g. supply chain management, customer relationship management, e-procurement, human resource management, and many industry-specific solutions. Its associated "collaborative business maps" define processes, roles, activities, interfaces, and documents for inter-organisational cooperation, e.g. for vendor-managed inventory, B2B sales, and collaborative engineering.

Baan provides iBaan [iBaan], a direct competitor to mySAP.com, especially in production environments. Like mySAP.com, it uses a Web-based portal environment to link collaborative enterprise applications. Siebel, perhaps the largest CRM product vendor, has its eBusiness Applications suite [Siebel], which has a Web-based architecture that includes components for multichanneling, personalisation, workflow, and integration with back office products such as SAP R/3. Siebel's applications, built on this architecture, cover sales, marketing, service, relationship management, and related fields. Workflow solutions vendor Staffware offers e.g. its Web-based eProcess products, such as Staffware eProcurement and eCRM [Staffware].

6.3 Positioning

The use of a distributed e-business application architecture, as embodied in the Web services paradigm, has many advantages. First, it promotes interoperability by minimising the requirements for shared understanding. By limiting what is absolutely required for interoperability, collaborating Web services can be truly platform and language independent. Second, it enables just-in-time integration. Dynamic service discovery and invocation (publish, find, bind) and message-oriented collaboration yield applications with looser coupling, enabling just-in-time integration of new applications and services. This in turn yields systems that are self-configuring, adaptive and robust with fewer single points of failure. Third, it reduces complexity by encapsulation. Behaviour is encapsulated and extended by providing new services with similar service descriptions. In the same manner, legacy applications can be wrapped and exposed as services.

However, this type of architecture also has a number of drawbacks. An e-business application may comprise external components that fall beyond the span of control of the organisation responsible for the application; the aggregated functionality of the application may critically depend on these components. When unexpected results are encountered from external components, it will be difficult to trace the origin of the problem. Such a component only exposes its interface, e.g. in the form of a Web service, and thus is essentially a black box which the user has no way to control or inspect, maybe only by requesting the service and see if it delivers the desired output.

More tightly coupled Internet-based architectures, such as those provided by ERP vendors like SAP and Baan, offer more control than loosely coupled Web services. Implementation quality is easier to guarantee, but vendor lock-in is a serious danger, although perhaps not as severe as in traditional ERP environments. It also seems that this market is opening up, as shown by the recent alliance between SAP Portals (formerly TopTier Software) and Baan. SAP Portals aims to provide an open enterprise portal solution that can integrate software applications from different vendors.

However, all Internet-based e-business applications are challenged by a number of issues. At this moment and in the near future, the Internet offers no performance guarantees for available bandwidth or latency. The performance of a distributed e-business application is therefore hard to ensure or predict. Many of the features mentioned before, e.g. easy integration by using XML over HTTP, also degrade the performance of the e-business application.

Another major concern in just-in-time e-business integration is trust. How do I know that the service provider that processes my sensitive business data can be trusted, and which security and reliability guarantees can this service provider give me? Technical measures are not enough to ensure trust. Even if the infrastructure is perfectly safe, how many people would trust Microsoft with their private data?

7 Conclusions

As we have seen in the previous chapters, the field of e-business frameworks is very diverse. We have distinguished different levels of functionality and granularity. The “bottom level”, web application middleware, is present in virtually all e-business applications. Above that, however, things become muddled. In practice, we see that most applications are either custom built using fine-grained components that offer limited, often infrastructure-like functionality, or assembled from large grained components that are closer to complete applications.

Sprott also recognizes this [Spro01]: “[…] components tend to be reused either at a relatively large grained or a relatively fine-grained level, but not in between. Relatively fine-grained components (and most if not all web services […]) can be assessed without major investment, and tested in isolation, making integration relatively straightforward. Large grained components (even packaged applications) actually do provide implementation transparancy to a significant extent and do not attempt to provide plug and play reuse.”

Although there are several business object and component frameworks that occupy an intermediate position (see Chapters 4 and 5), it seems that they are not really popular in practice. Furthermore, such frameworks are often extended to include larger grained business components and (near) complete applications, built on top of the original medium-sized components; these are then hidden from the end-user of the framework.

A plausible explanation for the gap between large- and small-grained component frameworks might be that most applications are either relatively standard or special purpose. For a standard application (e.g. a web shop) a set of packaged applications may suffice. The requirements of a special-purpose application will rarely be matched by large- and medium-size business components; only small infrastructure-like components will be useful.

The rapid service development methodology defined within the GigaTS project [FEJ+00] defines a framework (illustrated in Figure 13) that distinguishes seven different aspect areas, called cornerstones, from which models and specifications can be made. In this way, one can focus on one set of concerns at a time, resulting in a lower (perceived) complexity. At the same time, however, the methodology provides links between the cornerstones, thus rendering an integrated framework for business-driven design of transaction services. The RSD concepts for component-based rapid service development [BJSS01] describe how a component-based modelling approach may be used in describing the business perspective of e-business applications. The ensuing model components may be linked to software components that realise the functionality specified in the model.

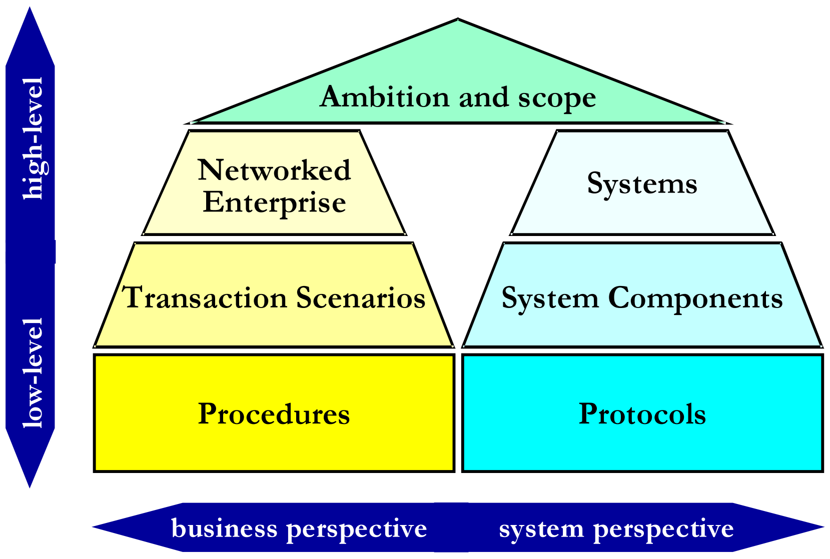

Figure 13. The RSD framework.

The applications, frameworks, components and objects discussed in the previous chapters occupy different positions within the system perspective of the RSD framework. If we look at the business logic stack of Figure 2, we can identify the relationship between the different technology layers and the RSD cornerstones. If we look at the responsibilities assigned to the the RSD system perspective cornerstones, they can be roughly mapped onto with the technology layers (see Figure 14): the e-business applications layer corresponds with the systems cornerstone, the business components and business objects layers correspond with the system components cornerstone, and the application support and web application middleware layers correspond with the protocols cornerstone.

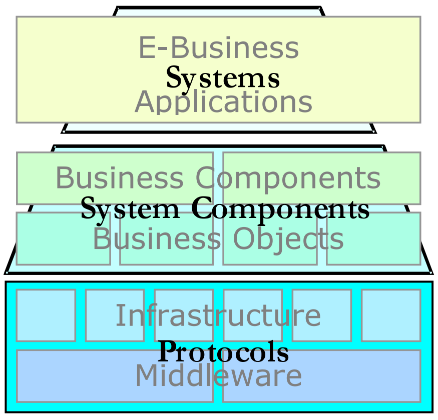

Figure 14. Web application stack overlayed with RSD system perspective.

Depending on the type of application to be built and the technology used, the trajectory through the RSD cornerstones will differ. Different paths through the cornerstones are identified in [FEJ+00], e.g. a top-down approach, an evolutionary approach, and an approach for legacy integration. The relationship between the RSD cornerstones and the technology layers may be helpful in guiding the development process. For software development, a link should be made between the RSD methodology and existing CBD methods, such as those described in [BJSS01].

References

Articles and whitepapers

[BFJ+00] A. Bruins, E. Fielt, H. Jonkers, B. Kloof, P. Oude Luttighuis, A. Smit, and M. Stefanova, *Queten – Illustration and validation of Rapid Service Development*, P. Oude Luttighuis (ed.), GigaTS/WP3/N03. Telematica Instituut, Enschede, The Netherlands, 2000. http://www.telin.nl/dscgi/ds.py/Get/File-9159/

[BJSS01] N. Boertien, H. Jonkers, R. Slagter, and M. Steen, *Component-Based Rapid Service Development*, N. Boertien (ed.), GigaTS/D1.3.4v2, Telematica Instituut, Enschede, The Netherlands, 2001 (forthcoming).

[ebXML] *Enabling Electronic Business with ebXML*, whitepaper, UN/CEFACT, OASIS, December 2000. http://www.ebxml.org/white_papers/whitepaper.htm

[FEJ+00] E. Fielt, G. van den Eijkel, W. Janssen, M. Steen, and P. Oude Luttighuis, *Rapid Service Development Methodology*, E. Fielt (ed.), GigaTS/D1.3, Telematica Instituut, Enschede, The Netherlands, 2000. http://www.telin.nl/dscgi/ds.py/Get/File-9156/

[FeKL01] P. Fennema, J.W. Koolwaaij, M.M. Lankhorst, *Web services: about to integrate distributed applications?* J. Koolwaaij (ed.), GigaTS/D2.2.9, Telematica Instituut, Enschede, The Netherlands, 2001.

[HeSi00] P. Herzum and O. Sims, *Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*. Wiley, 2000.

[JaKS00] W.P.M. Janssen, J.W. Koolwaaij, M. Stefanova, *XML – Hype or hope?* J. Koolwaaij (ed.), GigaTS/D2.2.4, Telematica Instituut, Enschede, The Netherlands, 2000. http://www.telin.nl/dscgi/ds.py/Get/File-11272/

[KoSt00] J.W. Koolwaaij, P. van der Stappen, *ERP, XRP and EAI in virtual marketplaces*, GigaTS/D2.2.8, Telematica Instituut, Enschede, The Netherlands, 2000. http://www.telin.nl/dscgi/ds.py/Get/File-15264/

[Micr00] Microsoft Corporation, *Microsoft Component Services - A Technology Overview*, 2000. http://www.microsoft.com/com/wpaper/compsvcs.asp

[Mons00] Richard Monson-Haefel, *Enterprise JavaBeans*, 2nd edition, O'Reilly, 2000.

[Pan99] Alex Pan, *Enterprise application integration – message broker style*, 1999. http://www.sunworld.com/sunworldonline/swol-08-1999/swol-08-itarchitect_p.html

[SOAP] D. Box et al., *Simple Object Access Protocol (SOAP) 1.1*. W3C Note, 8 May 2000. http://www.w3.org/TR/SOAP/

[Spro01] D. Sprott, Component Software Reuse – The Holy Grail orFool's Errand. *Interact:The Journal of Web Service & Component Based Business*, July/August 2001. CBDi Forum. http://www.cbdiforum.com/secure/interact/2001-08/index.php3

[UDDI] uddi.org, *UDDI Technical White Paper*, 6 September 2000. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf

[WSDL] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, *Web Services Description Language (WSDL) 1.1*. W3C Note, 15 March 2001. http://www.w3.org/TR/wsdl

Web sites

[.NET] http://www.microsoft.com/net/whatis.asp

[Ariba] Ariba Commerce Services Network, http://www.ariba.com/solutions/ariba_product.cfm?solutionid=9

[BizTalk] Microsoft BizTalk Server, http://www.microsoft.com/biztalk/

[Bluestone] HP Bluestone total-e-server, http://www.bluestone.com/products/total-e-server/default.htm

[Broadvision] Broadvision, http://www.broadvision.com/

[ColdFusion] Macromedia ColdFusion, http://www.macromedia.com/software/coldfusion/

[Compoze] Compoze Software, http://www.compoze.com/products.html

[CORBA] OMG, CORBA, http://www.omg.org/corba/

[Diamelle] Diamelle Technologies, http://www.diamelletechnologies.com

[EJB] Enterprise Java Beans, http://java.sun.com/products/ejb/

[eSpeak] e-Speak, http://www.e-speak.net/

[Evergreen] Evergreen, http://www.evergreen.com/products/index.html

[EzCommerce] EzCommerce, http://www.ezcommerceinc.com/

[Flashline] Flashline, http://www.flashline.com/

[GigaTS] http://gigats.telin.nl/

[HP_WSP] HP Web Services Platform, http://www.hp.com/go/webservices/

[i2] i2, http://www.i2.com

[iBaan] iBaan, http://www.ibaan.com/

[IBM_BC] IBM Business Components, http://www-4.ibm.com/software/Webservers/components

[IBM_SF] IBM SanFrancisco framework, http://www-4.ibm.com/software/ad/sanfrancisco

[IBM_WAS] IBM Websphere Application Server, http://www-4.ibm.com/software/Webservers/appserv/

[IBM_WCS] IBM WebSphere Commerce Suite, http://www-4.ibm.com/software/Webservers/commerce/

[IBM_WES] IBM WebSphere Everyplace Suite, http://www-3.ibm.com/pvc/products/wes/index.shtml

[iPlanet] iPlanet, http://www.iplanet.com

[J2EE] Java 2 Enterprise Edition, http://java.sun.com/j2ee/

[JBoss] JBoss, http://www.jboss.org/

[Jcorp] Jcorporate Expresso Framework, http://www.jcorporate.com

[JRun]	Allaire JRun, http://www.allaire.com/Products/JRun/
[OhioEdge]	OhioEdge, http://www.ohioedge.com/
[mySAP]	mySAP.com, http://www.mysap.com/
[OMIX]	OMIX, http://www.omix.com/
[Oracle_AS]	Oracle9i Application Server, http://www.oracle.com/ip/deploy/ias/index.html
[Oracle_BCJ]	Oracle Business Components for Java, http://technet.oracle.com/products/jdev/info/techwp20/wp.html
[RMI]	Java Remote Method Invocation (RMI), http://java.sun.com/products/jdk/rmi/
[RosettaNet]	RosettaNet, http://www.rosettanet.org/
[Siebel]	Siebel, http://www.siebel.com/
[Silverstream]	Silverstream Application Server, http://www.silverstream.com/Website/silverstream/pages/products_appserver_tf.html
[Spectra]	Allaire Spectra, http://www.allaire.com/Products/spectra/
[Staffware]	Staffware, http://www.staffware.com/
[Unify]	Unify eWave, http://www.unifyewave.com/products/commerce_ejbs.htm
[Weblogic]	BEA Weblogic, http://www.bea.com/products/Weblogic/server/index.shtml