Learn the discipline, pursue the art, and contribute ideas at www.ArchitectureJournal.net
Resources you can build on.

THE ARCHITECTURE JOURNAL™
Input for Better Outcomes
Journal 10

Composite Applications

Microsoft®

Contents

THE ARCHITECTURE JOURNAL™
Input for Better Outcomes
Journal 10

Foreword

Dear Architect,

I can hardly believe that we are already on the 10th issue of the *Architecture Journal*. It's been a fast-paced ride for the past few issues, especially as we've explored subjects such as workflow, data, and software factories. For the 10th issue I started thinking about a theme that "ties everything together." We've published numerous articles in recent issues about creating and exposing services in many different scenarios, so instead I wanted to focus on the consumption of these services. And what better way of investigating this than through the use of composite applications.

I wasn't quite sure what to expect when we put out the call for papers in September. I believe that composition in a service-oriented landscape is a term that's still being defined in the industry. Luckily, we ended up with a great set of articles that cover the area from a variety of viewpoints.

Chris Keyser, an architect in the ISV team at Microsoft, leads off this issue with the first article covering the fundamentals of composition, especially when looking at traditional line-of-business applications. Thomas Demmler from Open Text follows, documenting the architecture that his company has developed to help customers be more context-sensitive when responding to queries through e-mail. As I seem to live in e-mail for most of the day, this is an article from which I took away a lot of good ideas! Next up is Atanu Banerjee. In this issue Atanu takes a deep dive, looking at Office Business Applications (or OBAs for short), and walking through an example in Supply Chain Management to see how a new set of tools can help address some challenges in this area.

We take a quick pause after Atanu's article to chat with Scott Guthrie about his career at Microsoft and his thoughts on architecture. Many readers mentioned how they liked the interview in the previous issue with Jack Greenfield, so we're extending the series by getting more thoughts from luminaries in the industry. To end the composite applications theme for this issue, we have Mario Szpuszta, with an introduction and some recommendations for using the composite application block and smart client software factory, based on his experience with a customer he's been working closely with in Austria.

To close out this issue we have two great articles. Since the previous issue on data was so popular, we're delighted to have Semyon Axelrod continue the thinking and share his thoughts on quality data through enterprise information architecture. Wrapping up this 10th issue we'll take a look at five organization roles through the eyes of Per and Sten Sundblad as they talk about better software architecture and how it can lead to business improvement.

With this issue completed, the only thing remaining is for me to hope that these articles on composition will give you a different perspective when thinking about consumption of services for solutions that you're building today. See you next issue!

Simon Guest

Composite Applications – The New Paradigm

by Chris Keyser

Summary

Composite applications offer a long-sought-after business nirvana whereby empowered technical business users can stitch together componentized business capabilities. In many ways, composite applications are the business users' equivalent of Web 2.0 and "mash-ups." While there has been a lot of hype around composites, many vendors have been slow to deliver real value in this area. Technologies are emerging, however, that will change this game, and composition will become an increasingly important aspect of constructing business logic. In this article we'll discuss some of the fundamentals and advantages of using composite applications for today's business challenges.

Business Motivation

For many enterprises, empowering business users to react quickly to rapidly changing business environments and gain a competitive edge is a high priority. Business users have long been beholden to IT to create processes and logic. Composite applications have the potential, however, to move the reuse discussion from the technical domain to the business domain, freeing enterprises from the confines of stove-piped host applications and developers, and enabling them to define behaviors optimized for their business through process flows and metadata.

The recent phenomenon with Web 2.0 in the consumer marketplace indicates that a fundamental shift is taking place and is a leading indicator of users' increasing expectations. More users have increased influence in driving their own experience and are bringing these same expectations to the workplace. Based on working with Web 2.0, users will increasingly expect to exert more control over their work experiences and to participate in them. They will expect business applications to adjust to the way they work, rather than accept a suboptimal experience. These Web 2.0 capabilities will mature as they drive into the enterprise. Ultimately, Web 2.0 is not really about the technology. It's about social networks and users' control of their experience—disposable applications created by end users represent the ultimate ability to respond to quickly changing landscapes. In the enterprise, the way to achieve this movement of power to the end user is through a composite solution that considers the business user as well as the developer.

A composition platform provides a mechanism for multiple technology vendors to participate in a solution without hardwired and fragile dependences. Greater value can be realized from a much broader spectrum of application vendors with much greater ease—a composition framework provides an interaction model that allows components to be effectively decoupled and abstracted from dependencies on other components. As systems increasingly move to composition, they will also be increasingly metadata-driven. The movement to metadata will open opportunities for new sets of capabilities, such as self-healing and version resiliency to applications. The composition model therefore brings many aspects of service orientation to the entire landscape.

Composition—A Walk-Through

Composition means different things to different people. Today, line-of-business (LOB) vendors' systems predominantly manage well-known steps that represent discrete points in time when moving a lead to a sale. This is often referred to as a structured process.

In recent years, vendors have started to use familiar applications—for example, Microsoft Office—to front a better user interface on top of this process for interaction at specific points within an existing process. These improvements enhance the productivity and usability of the process and extend the reach of application logic to a much broader set of users.

Figure 1: Example of a structured process for Customer Relationship Management (CRM)

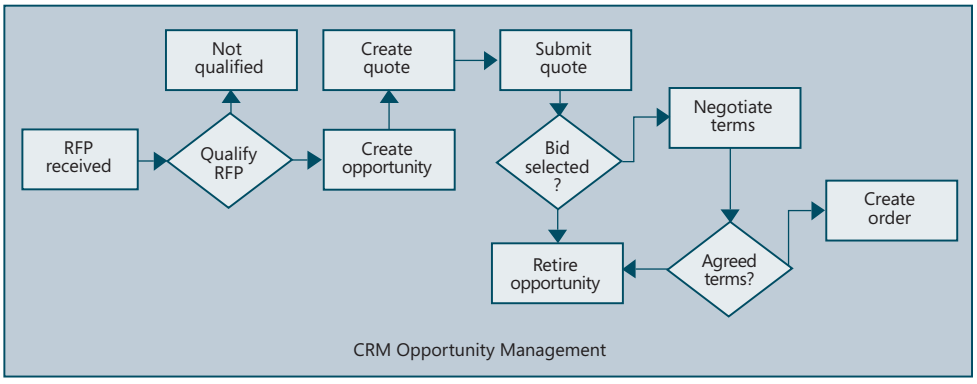
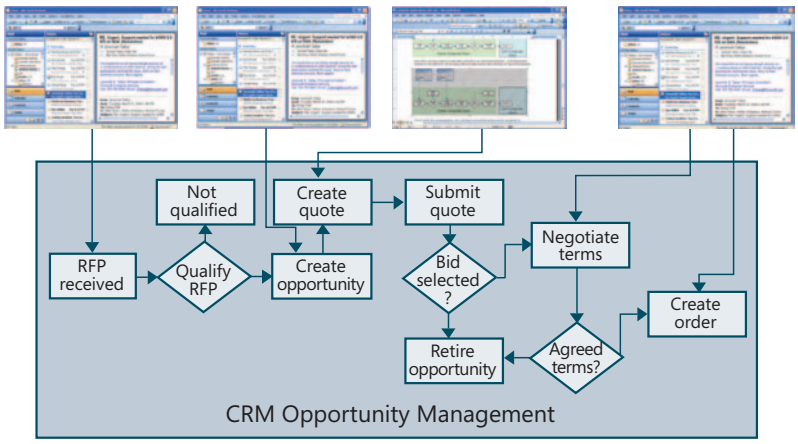


Figure 2: Using Microsoft Office for composition



These capabilities alone should motivate businesses to adopt them.

While these applications are a major improvement in extending reach, they are generally just surfacing existing functionality in a more usable way. Even with these applications, though, traditional LOB applications may be inhibited from progressing beyond this level of control. The inhibitors include:

- **Legacy System Construction**—Partitioning and componentizing these complex applications interlaced with many implicit dependencies are daunting tasks, and it will take a long time to truly unwind hardwired logic. LOB vendors eventually intend to move beyond their tight-coupled legacy systems into loosely joined capabilities based on service-oriented concepts that enable participation at a deeper level in the "composition" space.
- **Loss of Control**—It is scary for traditional LOB vendors to give up some control of data and process during the course of decomposing and surfacing capabilities through services and configurable user interfaces.
- **Pervasive Horizontal Nature of Collaboration**—Customers want one collaboration infrastructure. LOB vendors may not be well positioned to both scale up/down as necessary, or provide pervasive generic, easily managed capabilities.

New user interfaces in front of existing processes have not really given businesses any additional insight or control. Although the application may appear easier to use, the result doesn't improve an employee's ability to accomplish the task. As an example, this simplified process in Figure 1 describes turning a lead into an order for a custom-engineered product. Taking a lead to a sale actually involves much more work between elements than is represented in the initial structured process. This work is often collaborative in nature. Innovation frequently takes place outside structured processes in the collaboration interactions, where imagination predominates, where the creativity of information workers is truly tapped, and where the critical business differentiation occurs.

Although using a new interface, such as Microsoft Office, improves the view of the data, it does not in itself solve the collaboration problem. Integration of LOB systems into document-centric applications offers a particularly rich opportunity to compose information worker-oriented workflow extensions to the LOB process. With many portal solutions now incorporating workflow technologies (for example, the integration of Microsoft Office SharePoint and Microsoft Windows Workflow), it will increasingly become the collaborative workflow solution for LOB applications—blurring the line between managing documents and managing business processes. The example in Figure 2 illustrates how an account team would assemble a proposal in Microsoft Word to respond to an RFP. This is an example of using composition to extend the process across multiple-user experience channels (Microsoft Word, Microsoft Excel, and Microsoft Outlook), and have cooperating processes, one ad hoc and one structured, running on different systems to accomplish one logical business task.

A composite can also bridge processes running in independent LOB systems. For a case where some value-added processing needs to be injected, a composite service makes perfect sense, as is commonly the case today through integration middleware. This is illustrated by an information worker taking a confirmation response for a quote e-mailed to a customer from within Microsoft Outlook, and invoking a composite service to bridge CRM and ERP systems, creating the sales order in the CRM system and the internal order for processing in the ERP system.

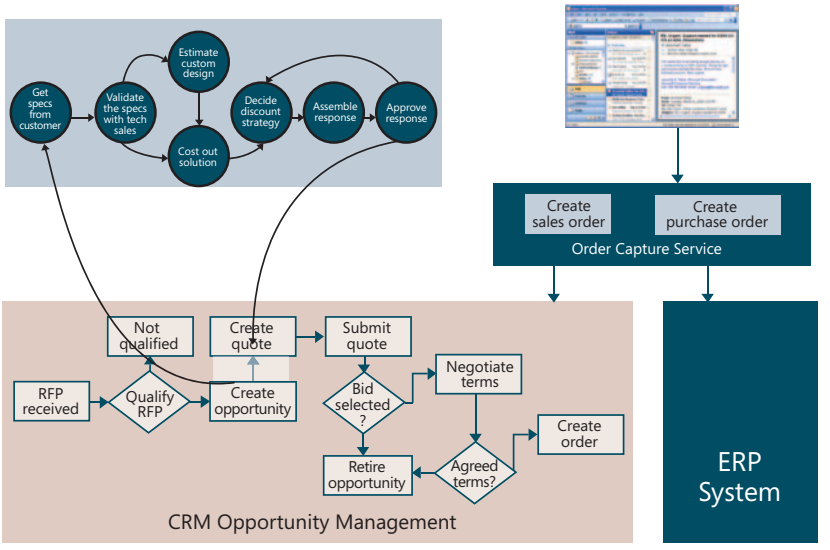
Layers of Composition

Now that we have a good idea of what composition means, it's worth understanding other types of composition that can take place. I like to classify these compositions by the layer at which composition occurs.

- **User Interface and Client Logic Composition**—Views can be assembled by loosely coupled user-interface parts that cooperate through a surrounding framework. These views can be assembled into sequences that have compatible interfaces through triggered events and information to compose higher-level functions. Such composition may take place through portals on the desktop or on a Web server. Examples today can include the Microsoft Patterns and Practices Composite Application UI Block, Microsoft Office SharePoint's composite application framework, Eclipse Rich Client Platform, and a number of other portal frameworks. Web parts, a term used to define the parts that make up a page, are enabling technologies for building a composite framework. Web 2.0 mash-ups typically compose at this layer, generally composing directly in the browser using Asynchronous Java and XML (AJAX) and represent informal and unanticipated compositions.
- **Services Composition**—Multiple services can be stitched together with process flow, metadata, and rules to provide value-added services. A somewhat trivial but common example is the addition of third-party credit checking to a purchase order service. Service Composition is often performed by integration engine capabilities, for example, Microsoft BizTalk Server.[1]
- **Data/Information Composition**—A third layer of data/information composition is often one of the hardest problems to solve, and most

[1]http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/dngrfSOAChallenges-EntityAggregation.asp

Figure 3: Composition across independent LOB systems



vendors don't solve this problem outside of well-defined niches. There are several techniques, however, that can help overcome this challenge. For example, Master Data Management (MDM) aggregates, scrubs, and distributes (through a syndication model) specific types of enterprise information. Entity Aggregation has also been discussed for several years and represents a more generalized approach based on ETL/Data Warehouses. Enterprise Information Integration (EII) can also be used as a straight-through processing mechanism to achieve a centralized view of shared information necessary to achieve harmonization of capability execution across a set of disparate services.[2] These approaches are the reality today because legacy systems do not have notions of strong-partitioned data boundaries along service-capabilities lines. Nor do systems today make it easy to build services that manage information along these partition boundaries. Pat Helland wrote extensively about the notions of reference data, resource data, activity data, and message data in his Metropolis series.[3] Maarten Mullender also took this up in his series on service agents.[4] Frameworks like the Information Bridge Framework (IBF) allow developers to navigate entity views and relationships described in metadata to enable composition locally by the application rather than through a mid-tier server.

- **End-to-End Process Composition**—In this case, process is a higher-level composition that stitches together distinct interactions through different mechanisms, typically over an extended period, often across a number of business systems. An example would be Employee Provisioning. The walkthough in Figure 3 illustrates interactions using multiple client applications accessing a back-end process across time. Even in cases today where LOB systems capture structured processes like quote-to-cash, there are many opportunities to take advantage of interactions occurring around the LOB structured process using process composition. Process is also important to other areas of composition within the boundaries of the composition, like user interface and service mentioned above, but use of process in those cases is distinct from this case. In those cases the processes are internal to the boundary of the composed surface—for instance, a workflow defined within a composed client for a task-driven UI (wizard-like), or a workflow internal to a composed service that orchestrates a number of secondary service invocations and business rule evaluations to provide a single unit of business functionality.

The Emerging Application Paradigm

Composite application frameworks have the potential to change the way that applications are constructed, delivered, and experienced by end users. At some levels, however, this complicates the life of application developers, since now more thought needs to be given to which parts of the experience should be surfaced through which vehicles. It also puts pressure on vendors to think harder about developing true service-oriented applications, carefully considering service boundaries for capabilities in composite environments. As composition frameworks become easier to use, and operational and support models evolve to the point that composite applications are as easy to run, then the power of drawing on many vendors will be a strong incentive to accelerate change.

A successful framework needs to significantly reduce the friction across every level of composite boundaries. Collaborative scenarios are not a prerequisite for a composite application but are a natural place for a lighter-weight solution that is relatively simple yet high in value. The ability to compose elements of an application will move upstream to technical business users as these platforms increasingly move toward model, workflow, and rules-driven approaches, and bring to the surface more configurable attributes from business logic.

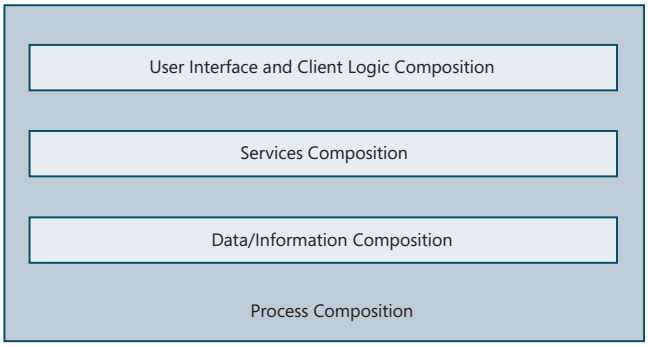
To realize this, we need to achieve interoperability at both the syntactic and semantic levels with line-of-business applications. Solutions to a number of difficult problems need to become accessible to everyday developers to achieve a full-blown composition infrastructure. Each of these areas warrants an article on its own, and several have already been explored in depth by others. These include:

- **Identity**—In particular, to have mechanisms to universally recognize a common user identity either through single store, or through federation. As cloud-based services mature, identity will be an important service that emerges. To realize a model of Software as a Service (SaaS), cloud-based identity stores will need to federate with local identity stores. The identity and authorization credentials must seamlessly pass between composite clients, composite services, and back-end service logic. Even on the desktop, experiences that cross applications need to be seamless to be useful in the long run. This will require broad adoption of mechanisms for federating and propagating identity.

[2]http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/dngrfSOAChallenges-EntityAggregation.asp
[3]http://msdn.microsoft.com/seminar/shared/asp/view.asp?url=/architecture/media/en/metrov2_part1/manifest.xml
[4]http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/concurev4M.asp4

- **Context**—For the parts of a composite application to work together to deliver a rich experience for sophisticated applications, they must have a shared notion of context. Mechanisms for passing context between cooperating composite applications and within a composite application need to be addressed .
- **Process Integration Between Composition Engines**—Projecting a view of the process interface defined within a particular service facilitates wiring complementary logic. As an example, I could design a document workflow that exposes a user interface and manages a Microsoft Excel document for sales-pipeline management and integrates into a multistep, roll-up approval process for the forecasts running in my sales force automation (SFA) business application. By projecting that roll-up process interface into a design environment, I can more easily build the cooperative workflow that triggers events or actions in a back-end system. Additionally, I could use the projected process interface to understand and interpret the current state of the corresponding process in another system.
- **Entity Definition**—For composition to cross boundaries of applications, and for entities to be used in many formats, we need a central notion that separates the conceptual entity from the physical representation. Once the notion of entity is defined, it can be reused through different physical representations. A centrally managed definition of entity, and the relationships between entities, will extend across a composite experience. Nontrivial problems remain to be solved: How will entities be managed across organizational boundaries? How will platforms make it easier to transform entities from one form to another? Clearly standardization approaches in the past have had mixed results at best. Microformats are an interesting demand-driven approach to creating de facto standardization. Entity Data Management and .NET Language Integrated Query (LINQ) are the first steps in this direction for the Microsoft platform.
- **Data/Information Management**—This relates to the entity problem but is slightly different. As mentioned in the previous section, today services and applications often don't have strong notions of data boundaries, or the mechanisms to make data boundaries enforceable while still delivering high performance. Notions of resource data, reference data, activity data (related to context), and message data need to be strongly identified and supported by design patterns, tooling, and infrastructure. Dealing with complexities like data ownership, data versioning, reference data syndication, and multiple valid versions is typically not considered when building applications today, nor does infrastructure make solving these issues easy.
- **Eventing Infrastructure**—In reality today, most Web services are request/response in nature. More sophisticated approaches supported by the WS-* protocol stack are needed for richer interaction patterns required by sophisticated applications.
- **Repository/Discovery Mechanisms**—UDDI gives one level of discoverability but has limitations. WS-Policy and WS-MetaDataExchange add additional layers to discover information for services and capabilities. This area needs to continue to mature.
- **Modeling and Metadata Frameworks**—Developers need to increasingly be aware of, and surface, developed functionality through models and metadata to pass more control through configuration to technical business users—and ultimately end users—to control the way applications behave and are assembled. The model components will be assembled using workflow and rules-based systems. Many of the concepts within software factories reflect this shift to both build better domain-specific solutions for developers, and to also pass control up the organization away from developers where warranted. This will require careful analysis by application architects and developers to determine whether to give up some level of behavior control and where to constrain configurability to ensure that correctness and consistency are not violated.

Figure 4: Layers of the Composition Stack



Conclusion

Composability is a paradigm shift in computing from brittle, monolithic, developer-centric applications solving one particular problem, to agile, contextual, user-driven applications to support particular business processes. As with the shift to service orientation, this will require not only refactoring existing code, but also rethinking new code. It will also require new infrastructure to host the applications. The Microsoft 2007 Office System, especially with the inclusion of SharePoint server support and support of Workflow, is a first step toward these applications. In the future, the trend will continue so that users can effectively configure and control their own work environment, passing data seamlessly among applications, and perhaps across enterprise boundaries, to complete their tasks. Much still needs to be done to accomplish this vision, but spurred on by user expectations from advances in Web technology, it will be hard to turn back now.

Acknowledgments

I would like to give special thanks to Daz Wilkin and Jon Tobey for helping crystallize and refine this paper.

About the Author

Chris Keyser is the lead architect for Microsoft's Global ISV team. Chris likes to write code and has a strong interest in Web services, and distributed and real-time systems. He spent the decade prior to joining Microsoft working for a series of start-up companies (B2B/e-commerce, voice biometrics, and physical layer network switching), using a variety of technologies in real time and business system development. For the first five years out of college Chris raised havoc in the United States Navy as a nuclear engineering officer on board the USS *Virginia*. That was a total immersion experience in systems engineering, an education that is very useful to this day. He graduated from college with a double major in computer science and history in 1984.

Context-Driven Access Via Microsoft Office

by Thomas Demmler

Summary

Aggregating document-centric content from heterogeneous systems is an ongoing challenge for system architects—as it is for the information workers who require that content. It is a major challenge to help customers improve the efficiency of their sales and customer service operations. Open Text solved this problem using Microsoft Office and the Information Bridge Framework.

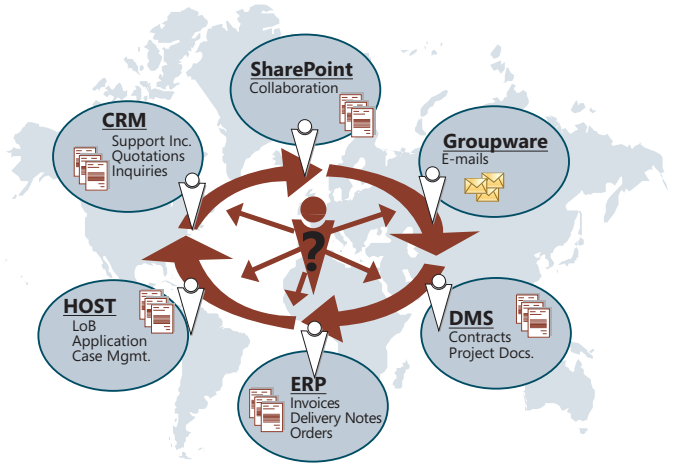
One of the main challenges in collaborative workflows today is that sales and customer service staff often have to spend a lot of time searching within heterogeneous systems, such as financial systems and document management systems, to find the information needed to respond to customer inquiries. To meet this challenge, Open Text developed Livelink ECM—Customer Information Management, a composite application based on our Enterprise Connect technology framework. This enables Microsoft Office users to access the document-centric content they need from the standard Outlook interface, irrespective of the enterprise system that is the source of the content.

When a sales or customer service employee receives an e-mail inquiry from a customer, Customer Information Management recognizes pieces of information in the e-mail, such as order numbers, customer numbers, customer names, and so forth, and tags them. The sales or customer service employee can then launch a single, full-text search across all enterprise systems for documents, business objects, and records relating to the tagged item. The results are displayed in a pane within Outlook, and the sales or customer service employee is transparently logged in (via single sign-on) to the information source of an item when the employee clicks its link. This article describes the architecture on which the Customer Information Management composite application is based.

Aggregating Enterprise Information Access

As Figure 1 illustrates, critical business processes often depend on information that spans multiple enterprise applications, such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Document Management Systems (DMS), and others. Business processes are not as efficient and accurate as they could be, because information workers are spending too much time searching for information in different applications and aggregating it manually.

Figure 1: Complex challenges of enterprise information access



Case Study: Streamlining Customer Information Access for Sales and Customer Service Workers

One of our customers in the heavy manufacturing industry based in Western Europe needed to streamline its operations to remain competitive with lower-cost competitors. One area in which it was looking for improvements was customer inquiry processing.

Typically, a sales executive or customer service worker would receive an inquiry by e-mail. E-mails contained questions about shipments, invoices, purchase orders, and other information stored in various modules of SAP® financial and ERP systems. Staff would then have to take the information contained in the e-mail (such as invoice numbers, PO numbers, customer numbers, etc.) and log in to different systems to gather the information necessary to respond to the customer. Not only was this a time-consuming process, but the staff often did not know whether they had found all pieces of relevant information.

Using a combination of Open Text's own technology and Microsoft's smart tag and Information Bridge Framework technologies, Open Text developers created a composite application (Customer Information Management) that places all customer-related information directly at the fingertips of sales and customer service staff in the Microsoft Office environment. Customer Information Management is the first of several planned composite applications based on Open Text's Enterprise Connect technology framework.

Key Concepts

Before looking into the architecture, it's important to cover the following key concepts: business objects and smart tags, defined below.

What Is a Business Object?

When a business transaction is carried out, there are many entities within it that interact by means of different types of business documents. For example, in the case of a simple transfer of goods, entities such as the vendor, customer, and distributor interact with one another using business documents that include purchase orders, invoices, and delivery receipts.

Within enterprise applications such as SAP systems, each of these entities and documents is represented by electronic objects. For example, customers and vendors are represented by their profiles or records in a database, and scanned images of purchase orders are stored in a repository. We use the term business object as a generic way of referring to these electronic objects that represent business entities.

What Is a Smart Tag?

Smart tags are a feature of Microsoft Office applications such as Microsoft Outlook and Word. They appear as indicators applied to text recognized by smart tag recognizers as representing a particular type of information. For example, the name of a recent Microsoft Outlook e-mail message recipient can be recognized and marked with a smart tag indicator.

The sentence below was typed into Microsoft Word with the smart tag recognizer for "person names" enabled. The smart tag recognizer recognizes "Thomas Demmler" as a person name, because e-mail was recently exchanged with him, and the recognizer tracks the names of e-mail correspondents.

Figure 2: Smart tag displayed in Word

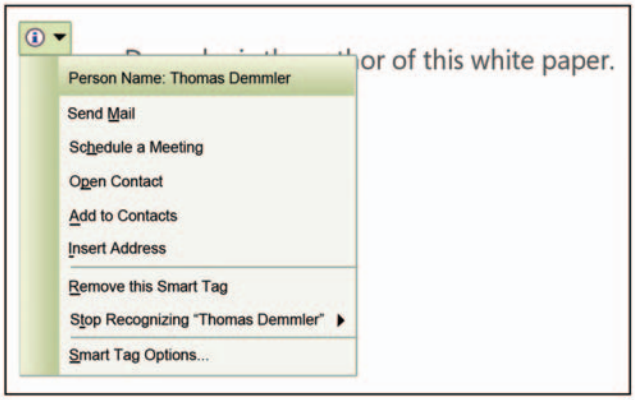
Thomas Demmler is the author of this white paper.

The recognized text is marked with a smart tag indicator, which is represented by a lavender dotted underline. When users place their mouse pointers over the tagged item, an icon appears that provides access to a menu of available actions.

Figure 3: Icon linking to smart tag action menu

The following image (Figure 4) is an example of the menu of available actions for a Microsoft Outlook contact recognized with a smart tag.

Figure 4: Smart tag actions for a name recognized as an e-mail contact



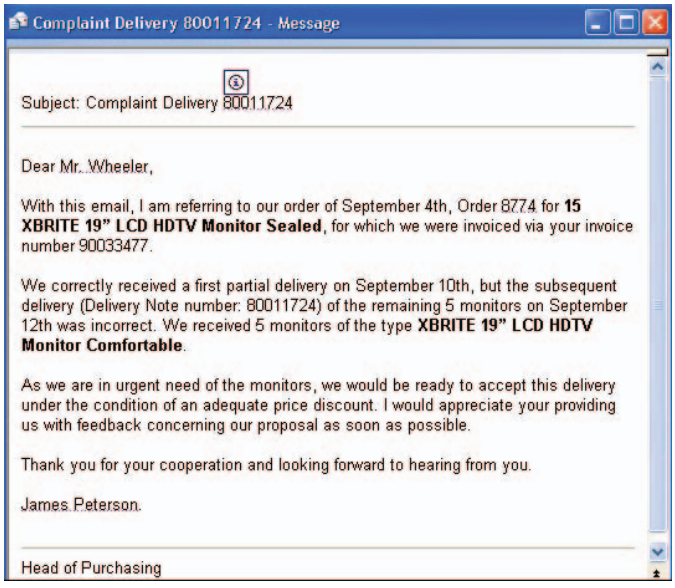
Open Text's Enterprise Connect leverages the Microsoft Office Smart Tag technology to tag information types, such as a customer name or customer number, and then provides an extended menu of actions that can be performed on the tagged information, such as searching for all information related to the customer in SAP ERP or other enterprise applications, including Open Text's own Livelink ECM document management system. The results of the search are displayed in a special pane in the Microsoft Office application's UI, the task pane.

Aggregating Information Access in the Microsoft Office Environment

This section illustrates how end users experience Enterprise Connect technology in Microsoft Outlook and other Microsoft Office applications.

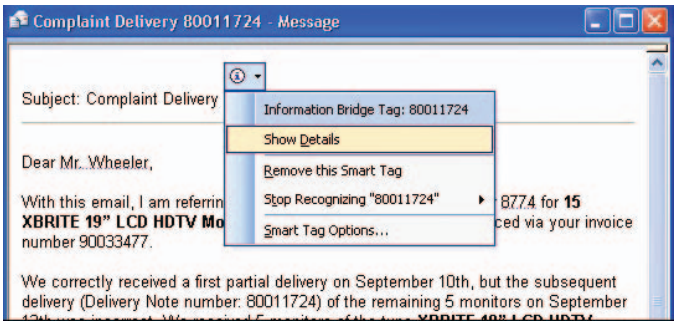
The image below displays an e-mail sent from a customer to an account manager. In the e-mail, the customer references a delivery number, which is recognized and marked with a smart tag.

Figure 5: Smart tagged delivery number in a Microsoft Outlook e-mail



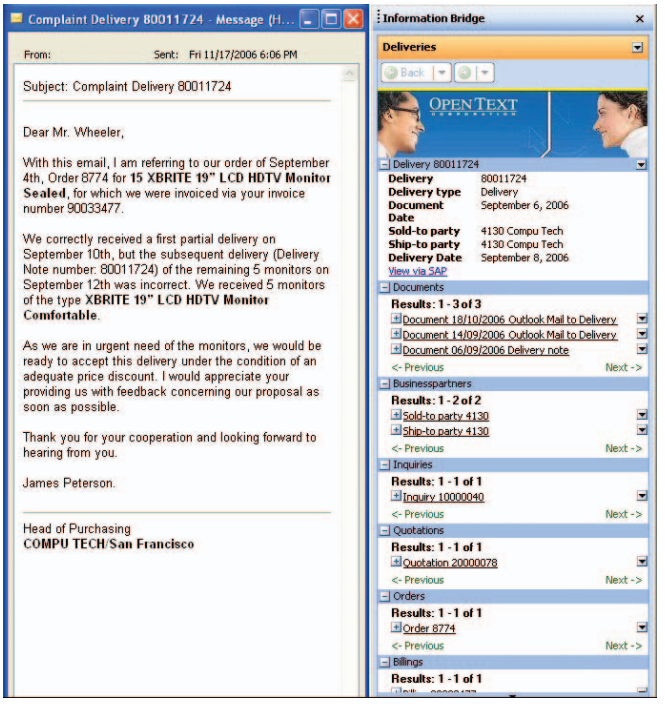
When the account manager places the mouse pointer over the tagged delivery number, it opens the action menu, as shown in Figure 6.

Figure 6: Action menu for smart tagged delivery number



The menu lists the **Show Details** action. When the account manager selects the **Show Details** action, all enterprise information related to the delivery number is searched for, and the results are displayed in the Information Bridge task pane within the Microsoft Outlook interface, as shown below.

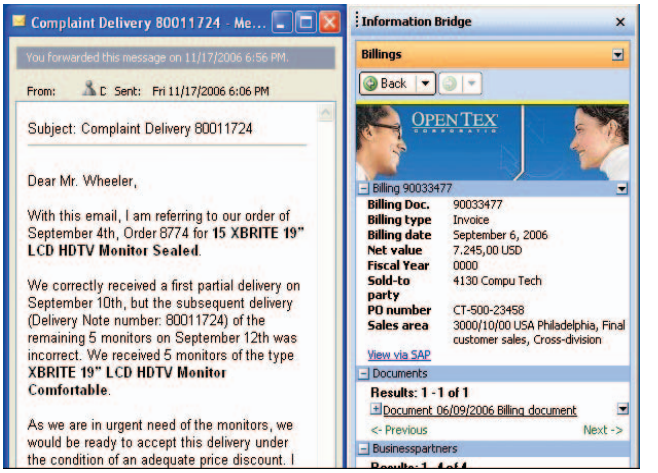
Figure 7: Information Bridge pane displaying result of Show Details action



The Information Bridge task pane displays links to all business objects relating to the delivery (orders, documents, billings, inquiries, and so on) from SAP systems, Open Text's Livelink ECM document management system (DMS), and other online sources. In the example above, the Information Bridge pane is shown in the Microsoft Outlook interface, but it can also be displayed in Microsoft Word and other Microsoft Office applications. Documents that are attached to the delivery business object, such as offers, documents, orders, inquiries, or billings, can be directly accessed from the Information Bridge task pane.

With a single click, the account executive can access the information displayed in the Information Bridge pane, such as the invoice number referenced in the customer's e-mail message.

Figure 8: Invoice details from SAP displayed directly in the Information Bridge pane

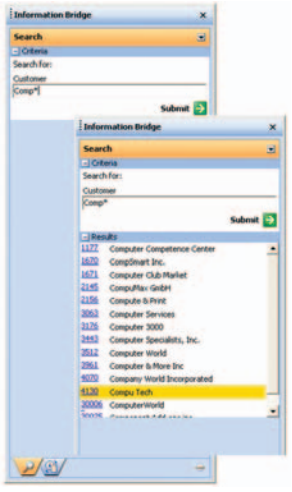


The account executive can also quickly access orders in SAP, contracts and other documents in Open Text's Livelink ECM DMS, and more.

The Information Bridge task pane also has a **Search** tab, as shown in the following image. This tab enables information workers to search directly for business objects, rather than initiating the search from a smart tag.

Users can also attach the e-mail itself to the business object in SAP by dragging and dropping the e-mail message from Outlook onto the task pane.

Figure 9: Search tab in Information Bridge task pane

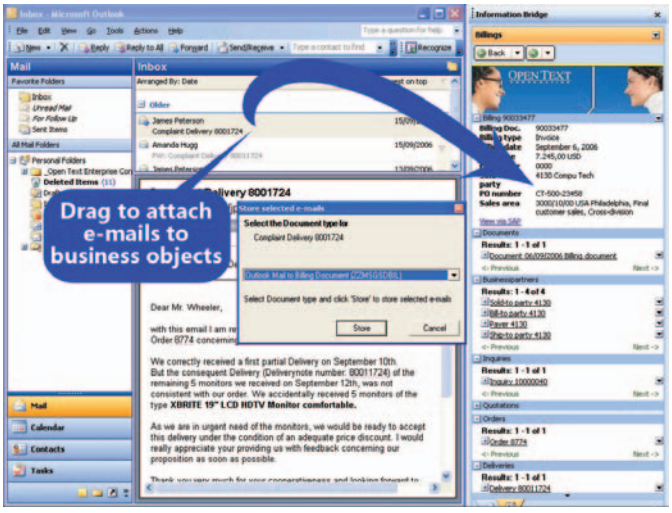


Open Text's Enterprise Connect Framework

Enterprise Connect comprises several interrelated layers. The central layer is the Enterprise Connect Engine, which not only connects the enterprise systems, but also creates a foundation of common business objects and processes.

To mark and display these business objects within Microsoft Office applications, Enterprise Connect uses Microsoft Office's smart tag technology. Enterprise Connect applications provide the intelligence to recognize references to specific business objects in Microsoft Office documents.

Figure 10: Attaching an e-mail to the order in SAP using drag-and-drop



Enterprise Connect Engine

The Enterprise Connect Engine currently runs on Open Text's Livelink ECM framework and will be supported on SAP's NetWeaver™ business infrastructure platform in future versions. The Enterprise Connect Engine is implemented in Java and uses Web Services, which enable communication between Microsoft Office, the Microsoft Information Bridge Server, and enterprise applications. Open Text Enterprise Connect is a Web Service that can be easily integrated with any Service-Oriented Architecture (SOA), such as SAP's Enterprise Service-Oriented Architecture (ESOA).

Livelink ECM DocuLink

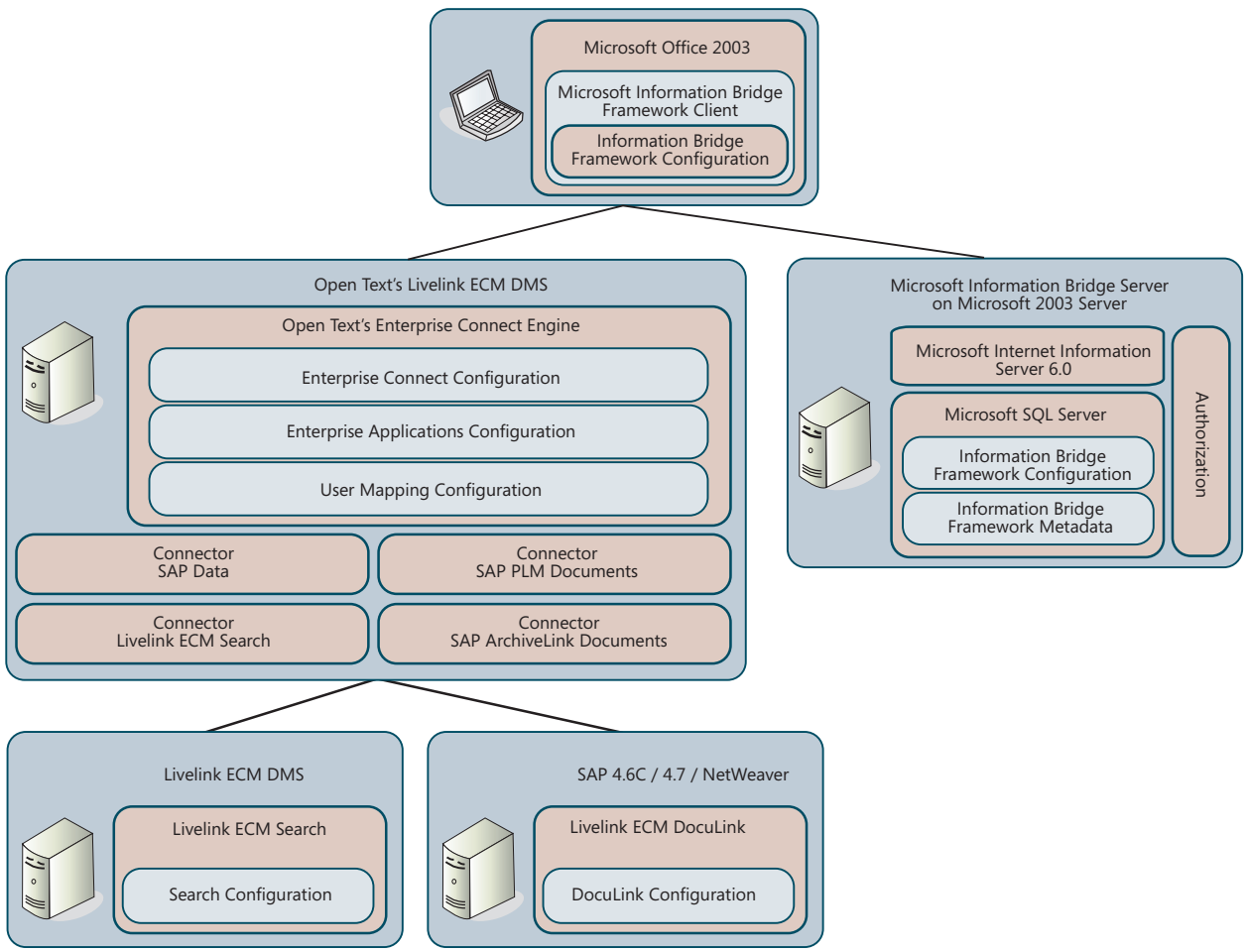
The connectors to SAP systems leverage Open Text's Livelink ECM DocuLink. DocuLink is a module based on SAP's Advanced Business Application Programming (ABAP) language and runs on SAP NetWeaver. It connects end users to all SAP business objects and executes SAP's authorization model, which restricts all unauthorized access to business-critical information in SAP systems.

DocuLink extends the SAP system with a process-oriented and application-spanning view of all business documents—across SAP systems, SAP modules, and even non-SAP systems. DocuLink collects all business documents and reorganizes them according to the business process and the need for document retrieval. The navigation scheme can also be configured to reflect changing business processes.

Livelink ECM Search

The connector to Open Text's Livelink ECM DMS is based on its integrated search engine. This connector makes it possible to include

Figure 11: Open Text's Enterprise Connect Framework



documents, such as project and customer documents, from several Livelink ECM DMS repositories in the Enterprise Connect object model.

Single Sign-on Technology

Enterprise Connect provides end users with single sign-on access to any of the enterprise applications in which business objects reside, because the credentials used to connect to Microsoft Outlook automatically provide them with access to the related enterprise systems. Enterprise Connect provides central administration of access to external systems, making it easier to define individual and group mappings.

Building Blocks of the User Interface

Open Text's Enterprise Connect uses the Microsoft Information Bridge Framework technology to integrate smoothly into the Microsoft Office applications.

The Information Bridge Framework Task Pane Tabs

The Information Bridge task pane is the central location for interaction with the connected back-end systems. The Information Bridge task pane contains two tabs:

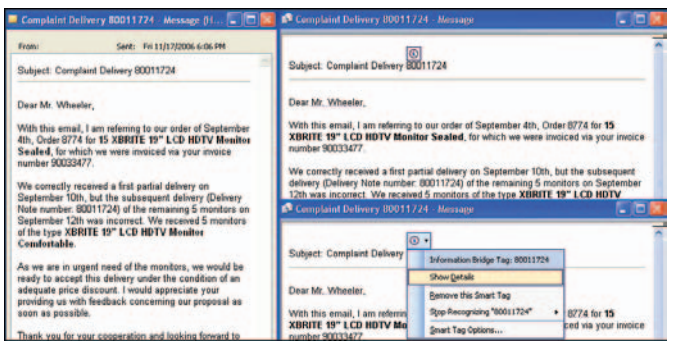
- **Information Tab:** Displays information about business objects
- **Search Tab:** Allows searching for a business object

Once a business object is located on the **Search** tab, information workers can click a result to view information about the object in the **Information** tab. In addition to the **Search** tab, recognized text (such as customer names, numbers, and so on in the case of Customer Information Management) in Microsoft Office documents (such as Word documents and Outlook e-mail messages) that are tagged using Microsoft's smart tag technology also allows information workers to find business objects.

Identifying Business Objects

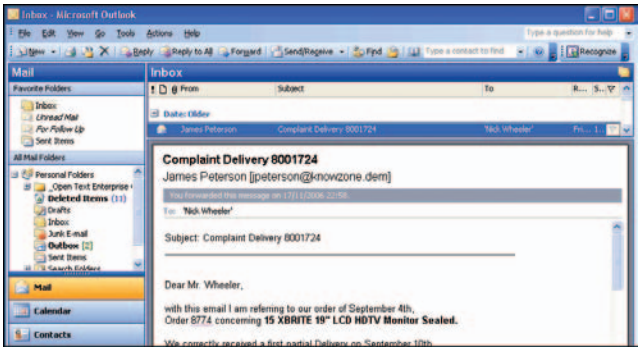
The most convenient way for information workers to identify a business object is by means of smart tags inside an e-mail or a Microsoft Office document. In Figure 12 below, a customer number has been identified in an e-mail. By selecting the Show Details action, information workers can obtain detailed information about the customer, which is displayed on the **Information** tab of the Information Bridge task pane.

Figure 12: Smart tag (Information Bridge tag)



To identify business objects within an e-mail that has just been received, information workers can click the **Recognize** button in the Outlook toolbar to initiate smart tagging. Each Enterprise Connect solution includes a predefined set of patterns (customizable by the administrator via a graphical configuration screen) that the recognizer searches for. These patterns correspond to the business objects of interest for the specific solution. In the example of the Customer Information Management solution described in this article, the predefined patterns recognize customer-centric information such as customer numbers, purchase order numbers, and so on. The e-mail is converted into HTML format, and all recognized business objects appear underlined in the Outlook preview pane or opened e-mail.

Figure 13: Recognizer in Microsoft Outlook



There may be some cases in which a business object identifier (customer name, number, and so on) is written incorrectly or in a way that confuses the recognition technology. In cases in which business objects are not identified automatically, information workers can use the **Search** tab in the Information Bridge pane (see Figure 9) to search using specific attributes. For example, an order can be searched by order number, order date, customer number, and purchase order of the customer. Wildcards for nonspecific searches are also supported.

Views of Business Objects

The **Information** tab provides information about a business object and its context. Enterprise Connect provides a simple user interface model inside the Information tab. Each business object has a set of information (including other business objects related to it) that can be displayed on the tab. The set of information available to each Enterprise Connect solution is based on an Information Bridge Framework configuration created using the Enterprise Connect graphical configuration tool. The information set is deployed as XML to either the Information Bridge Framework Metadata

Figure 14: Native SAP WebGUI

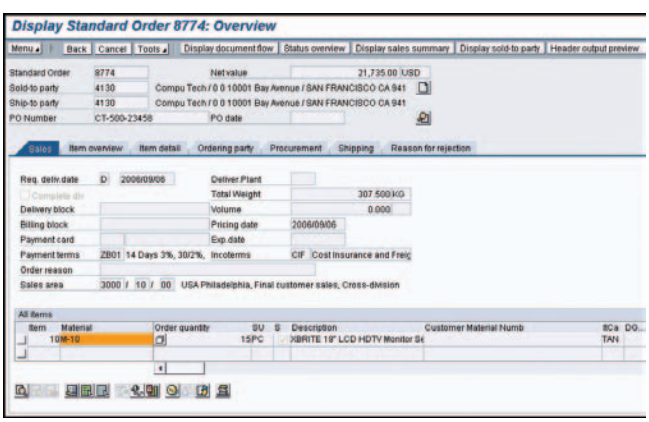


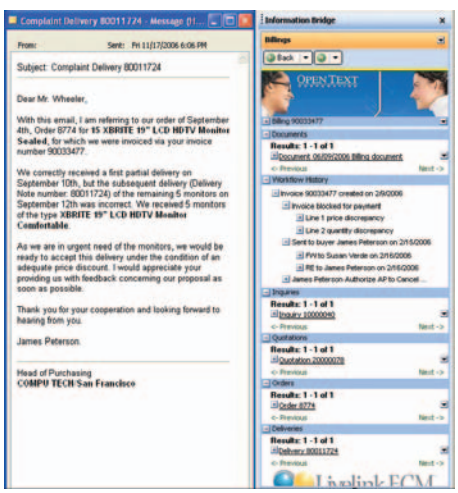
Figure 15: Information tab with subsections grouping related objects

Server or to individual Microsoft Office workstations.

The top section of the **Information** tab displays details about the business object selected. The type of business object (for example, "Delivery") is displayed in the heading of the Information tab. Using the information stored as XML in an Information Bridge configuration, Enterprise Connect provides several different views of a business object, from which end users can select the most appropriate one for the task at hand. If end users are familiar with the back-end system from which the business object originates, they can click a link to open the object in the native user interface of the back-end system. For example, information workers can click to view an order object in the SAP WebGUI.

The set of operations that can be performed on a business object from the **Information** tab depends on what the back-end system supports for that object. For example, the back-end system may support operations such as "Attach document to business object" or "Add new version of document," depending on the business object in question.

Figure 16: Information tab of an invoice object with Workflow History section open



Within the **Information** tab view of a business object, other business objects related to it are grouped into subsections by type.

For a "Delivery" business object, the **Information** tab displays customer details at the top of the pane and a number of sections below it that group other objects related to the customer, such as Orders, Documents, Billings, Quotations, Inquiries, and so on.

For a "Billing" business object, the Information tab displays invoice details at the top and related objects, such as Orders, Documents, and Workflow History, below it.

Conclusion

To address a particular type of content aggregation and access need, Open Text developed Livelink ECM—Customer Information Management, the first of several planned composite applications based on our "Enterprise Connect" technology framework. Enterprise Connect aggregates document-centric content relating to business objects in multiple SAP systems and modules, in addition to documents in Open Text's own Livelink ECM DMS, and provides access to this content in the Microsoft Office user environment.

The Enterprise Connect composite application leverages Open Text's own document management and search technologies in combination with Microsoft's smart tag and Information Bridge Framework technologies to deliver the aggregated content via the Microsoft Office interface.

Appendix: Technical Specifications

Available Enterprise Connect Connectors

- SAP R/3 4.6 C/D, SAP Enterprise (4.7), SAP NetWeaver 2004 and SAP NetWeaver 2004s
- Livelink ECM – Enterprise Server (DMS) 9.2 SP1 and 9.5 SP1
- Livelink ECM – Production Document Management 9.5.5

Client Environment

- Microsoft Windows Office 2003 Professional or higher
- Microsoft Windows 2000 Professional or higher
- Microsoft Windows .Net Framework 1.1

Server Environment

- Information Bridge Framework Server
  - Microsoft Windows Server 2003 or higher
  - Microsoft Windows .Net Framework 1.0 or higher
  - Microsoft Exchange Server 2003 or higher
  - Microsoft SQL Server 2000 or higher
- Platform for Open Text's Enterprise Connect
  - Livelink ECM – Enterprise Server (DMS) 9.5 SP1
  - In future: SAP NetWeaver
- Document Archive
  - Livelink ECM – Archive Server 9.5 and 9.6

About the Author

Based in Munich, Germany, **Thomas Demmler** is currently product management director for SAP-related products at Open Text Corporation. He has a Master's of Computer Science (Diplom-Informatiker) from the University of Karlsruhe in Karlsruhe, Germany. Mr. Demmler joined IXOS AG (acquired by Open Text in 2003) in 1998 and has been involved with the company's SAP-related product lines in various roles since that time.

Building Office Business Applications

by Atanu Banerjee

Summary

The 2007 Microsoft Office system provides a set of servers, clients, and tools to make it easier for enterprises and software vendors to build and deploy composite applications in the enterprise. These solutions, called Office Business Applications (OBAs), are quick to build and deploy; empower end users through extensive personalization capabilities; are easy to change when business needs require; and are built using familiar Microsoft Office tools and applications. This paper shows how to architect composite applications, and how the 2007 Microsoft Office system provides a good platform—familiar to end users—for building such applications.

Globalization, specialization, and outsourcing require people to work in more collaborative ways than before. This trend requires a matching change in the tools that information workers use to gain insight, collaborate, make decisions, and take action. Today, most business applications are effective at automating transactions, but do not enable rich collaboration across functional boundaries. This usually leads information workers to use personal productivity tools to perform the complex interactions required to conduct business. However, this in turn leads to a loss in productivity, as users are forced to cross from one set of tools to another, often manually moving the data through means such as cut-and-paste. These gaps between different business applications and productivity tools need to be bridged for information workers in a way that is seamless, synchronized, and secure.

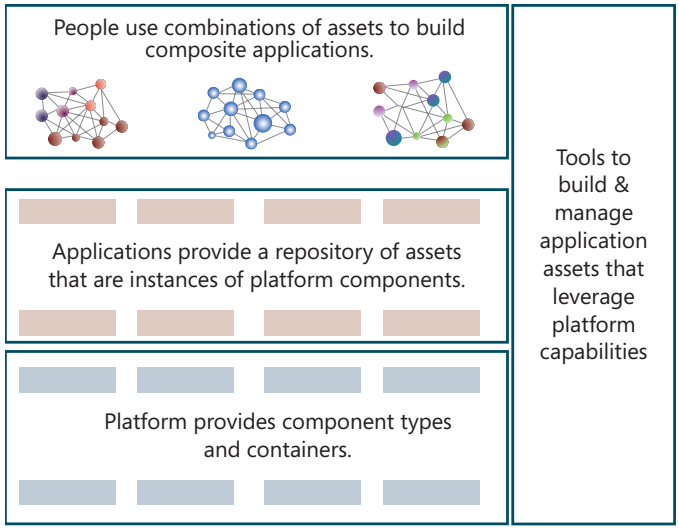
There really has not yet been an effective way to enable composition of business applications in a way that is contextual, collaborative, easy to use, role-based, and configurable to pull in other platform technologies that are needed for building these kinds of composite applications. Ease of use is important, as the platform, tools, and architecture for composition should not introduce unreasonable technical complexity that requires radical retraining of people.

What Are Composite Applications?

A composite application is a collection of software assets that have been assembled to provide a business capability. These assets are artifacts that can be deployed independently, enable composition, and leverage specific platform capabilities.

In the past, an enterprise's software assets usually consisted of independent business applications that were monolithic and poorly integrated with each other. However, to get the business benefits of composition, an enterprise must treat its software assets in a more granular manner, and different tiers of architecture will require such varying assets as presentation, application, and data assets. For example, a Web service might be an application asset, an online analytical processing (OLAP) cube might be a data asset, and a particular data-entry screen might be a presentation asset.

Figure 1 High-level representation of a composite application

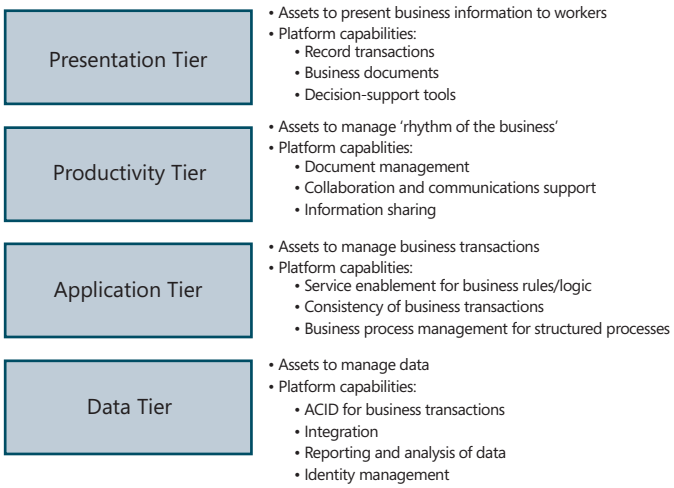


An inventory of software assets by itself does not enable composite applications. This requires a platform with capabilities for composition—that is, a platform that provides the ability to deploy assets both separately and in combination with each other. In other words, these assets must be components, and the platform must provide *containers* (Figure 1).

Containers provided by the platform need to be of different types, which map to the different tiers in the architecture. Enterprise architectures are usually decomposed into three tiers: presentation, application (or business logic), and data. So the platform needs to provide containers for these. However, the three-tier architecture assumes structured business processes and data, where all requirements are made known during the process of designing and building the system. By their very nature, composite applications presume that composition of solutions can occur after assets have been built and deployed—and so need to explicitly account for

Figure 2 The four tiers of a composite application

A composite application requires four types of assets



people-to-people interactions among information workers that are essential to complete any business process. Usually these interactions are not captured by structured processes or traditional business applications; therefore, it is critical to add a fourth tier—the productivity tier—to account for these human interactions. This is shown in Figure 2.

Traditional discussions about the architecture of business applications tend to focus on the application tier as being the connection between people and data. Typically, however, the application tier contains structured business logic, and this holds for discussions about Service Oriented Architectures (SOAs), Enterprise Service Buses (ESBs), Service Component Architectures (SCAs), and most other architectural perspectives in the industry today, including first-generation discussions about composite applications. However, building a composite application requires the architect not only to consider the productivity tier a critical element of the stack, but also to recognize that it contains the most business value.

To expand on the comparison between composite applications and SOA: Both of them target flexibility and modularization. However, SOA provides flexibility at just one tier: the structured business logic in the middle tier. Composite applications target flexibility at all four tiers. That said, a composite application is a great way to extract information from an SOA, and having line-of-business (LOB) applications exposed as services makes it easier to build support for cross-functional processes into a composite application.

Therefore, to design a composite application, a solutions architect must:

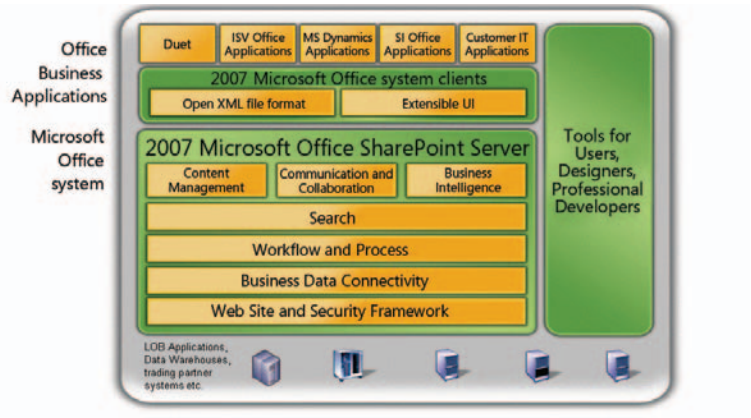
- Choose a composition stack. Pick one or more containers from each tier, and a set of component types that are deployable into those containers.
- Choose components. Define the repository of assets that must be built from this set of component types, based on business needs.
- Specify the composite application. Define the ways in which those assets will be connected to provide a particular cross-functional process. The platform should enable these connections to be loosely coupled.

After deployment, users will have the opportunity to personalize both assets and connections, and the composition stack should enable this through loose coupling and extensibility mechanisms.

The 2007 Microsoft Office System as a Platform for Building Composite Applications

The 2007 Microsoft Office system is such a platform for building composite applications—which are called Office Business Applications (OBAs).

Figure 3 Capabilities provided by the 2007 Microsoft Office system



How Office Business Applications Are Composite Applications

Let us now look at each of the tiers in Figure 2, and examine the types of containers and component types that the platform provides.

Composition in the Presentation Tier

The first type of container in the presentation tier is the Microsoft Office client application (Excel, Word, Infopath). These applications are customizable containers that can now more easily extract information and functionality from LOB applications through custom

Figure 4 Office client applications are customizable containers for Open XML content.

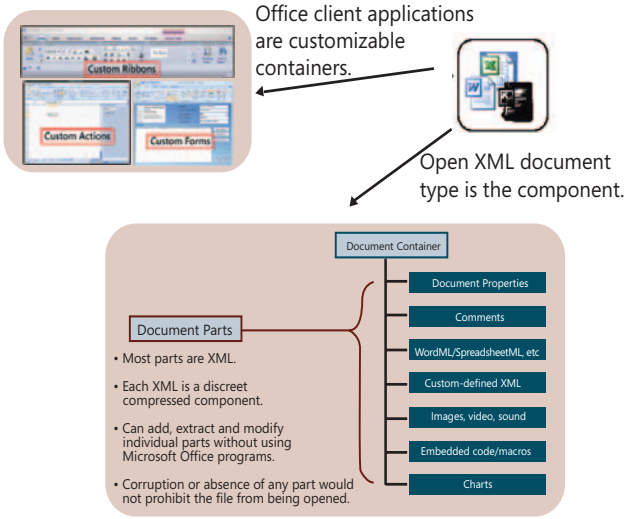


Table 1 High-level 2007 Microsoft Office system capabilities

Capability	Description
Web Site and Security Framework	A common framework for creating different kinds of sites such as team collaboration sites, intranet portals, Internet Web sites.
Open XML File Formats	This enables rich server-side processing of documents. With prior versions of Microsoft Office, parsing the document (e.g., a spreadsheet) using the document object model required an in-memory, running instance of the application used to create the document (e.g., Excel).
Extensible UI	Server-side portal that can be personalized by users, from building blocks that can be extended by solutions providers. Client applications that can be extended with Visual Studio Tools for Office.
Business Data Catalog	A metadata repository to define business entities stored in back-end data stores, to model relationships between entities, and to define actions permissible on entities.
Enterprise Search	To extract data from various enterprise sources through searching.
Workflow	Integrated with Workflow Foundation to host workflows that represent people-to-people interactions and that link UI elements.
Enterprise Content Management	Manage diverse content, with one topology for Web, document, and records management. Support for document life-cycle management.
Business Intelligence	Server-based Excel spreadsheets, plus BI components (dashboards, reports, Web Parts) built into the portal and connected to SQL Server Analysis Services.
Communication and Collaboration	Support for unified communications integrated into the platform.

task panes, custom ribbons, and actions that present users with data and actions in the context of their current activity. The component type that can be hosted within these containers is the Open XML document. This is the new XML representation for Microsoft Office documents—which enables rich server-side processing of information stored within. This is shown in Figure 4.

The second type of container is a Web portal, as enabled by Windows SharePoint Services (WSS), and Microsoft Office SharePoint Server (MOSS).

WSS v3.0 provides the following hierarchy of containers, as shown in Figure 5:

- **Farm**—Installation of one or more load-balanced Web servers, and back-end servers, with a configuration database.
- **Web Application**—IIS Web site, extended to use WSS, which can host site collections.
- **Site Collection**—Container for WSS sites, which exists within a specific content database. A site collection contains a top-level site and optional child sites, and is the unit of ownership, securability, and recoverability.
- **Site**—Container for child sites, pages, and content such as lists and document libraries.
- **Page**—Container for Web Part zones, and Web Parts.
- **Web Part Zone**—Container for Web Parts.
- **Web Part**—Components that display content on a page in modular form and are the primary means for users to customize/personalize pages.

While WSS provides support for building collections of sites for team collaboration, MOSS enables portal functionality on top of WSS, with additional capabilities. For example, MOSS comes with capabilities for enhanced searching, connectivity to business data stores, and Business Intelligence. However, a MOSS portal is a WSS site collection, and so is a hierarchy of WSS sites. MOSS also comes with a Web Part gallery that contains a rich set of Web Parts out-of-the-box, for example to extract Microsoft Office Excel spreadsheets and charts. Solution providers can also provide their own custom Web Parts for application-specific or domain-specific functionality—which can then be uploaded into WSS. Information workers can personalize pages by adding Web Parts from the gallery, removing Web Parts from a page, or rearranging the layout.

Composition in the Productivity Tier

The 2007 Microsoft Office system provides a number of different ways to share information and collaborate on using it. For example, server-side storage provides containers for in-process documents, and other kinds of heterogeneous content with version-control. This allows for collaboration in unexpected or unplanned situations. It is hard to capture all the complex people interactions in a business process

Figure 5 Containers provided by Windows SharePoint Services

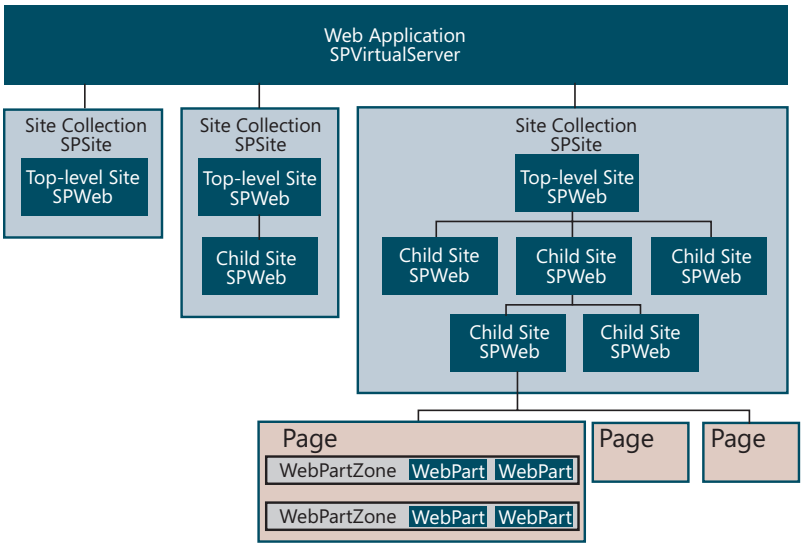
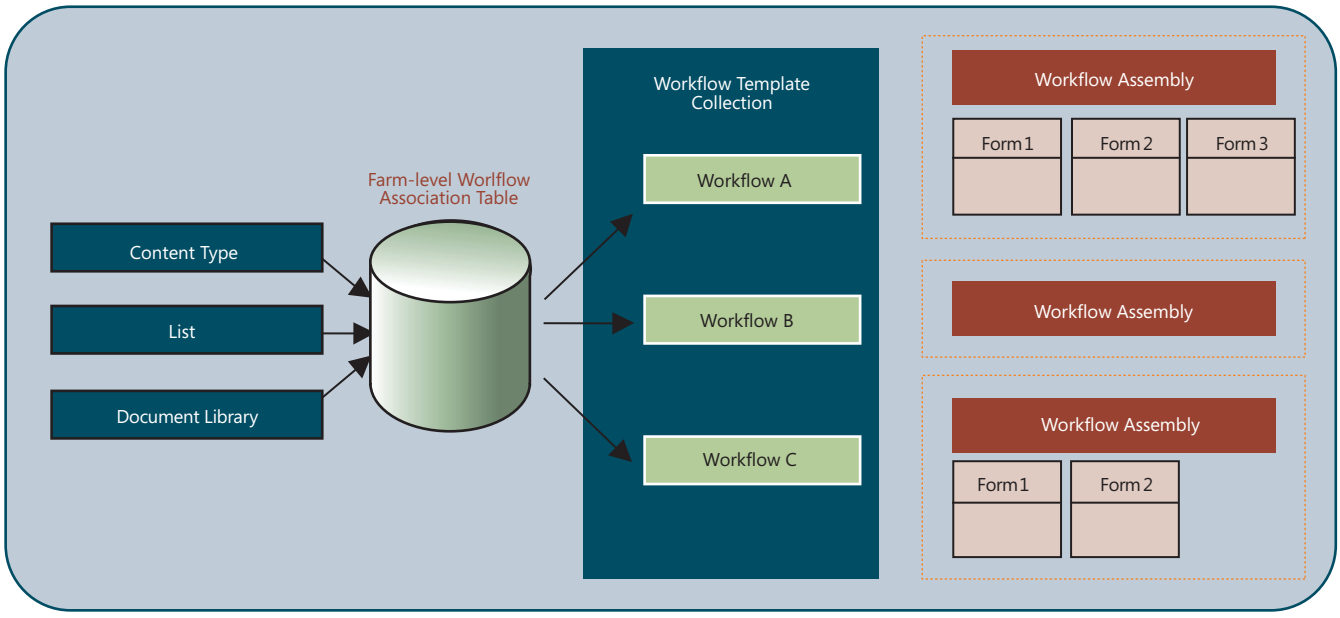


Figure 6 SharePoint Workflow Architecture



management (BPM) system, as work usually never happens as planned. In traditional three-tier architectures, there is little support for this beyond storage of in-transit documents on personal hard drives or e-mail servers, and it can sometimes be hard to decide which document is the correct version. However, the 2007 Microsoft Office system can set up versioned document libraries to store documents at the intermediate stages of a business process, which improves manageability, and also improves the likelihood of graceful recovery from unplanned events. WSS provides the following types of storage:

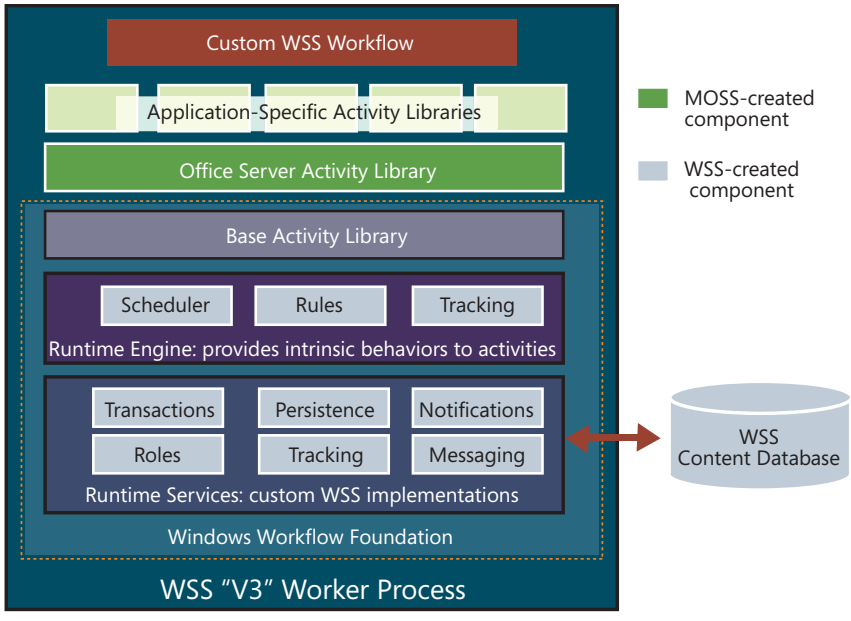
- **List**—Container for items, which could be from built-in list types, or from custom list types. Traditionally, this has been the atomic unit of storage, but now a list can store huge amounts of data, such as knowledge bases or Web content. There is also built-in index and query support.
- **Document library**—Special types of lists that support versioning and source-control of documents. For example, InfoPath forms may be stored in forms libraries, reports in report libraries, and other documents in document libraries.

Information workers can add document libraries themselves and specify particular document templates to be used by all documents in those libraries. For example, a business analyst might use the Infopath WYSIWYG design tool to create a form, and then use that form as a template to create a forms library. Whenever users create a new document within that library, this template will be used to create an empty form.

The 2007 Microsoft Office system also has built-in, server-side support for Windows Workflow Foundation (WF). The WF run-time engine is hosted within MOSS and acts a container for business logic that can be attached to work items and documents in the form of workflows. These workflows may be associated with lists, document libraries, or with particular content types. They are started and completed by user actions, and are managed using WSS task lists. For example, workflow activities create and update task items as required, and users can track workflow progress through history tables. 2007 Microsoft Office client applications are also workflow aware. They can be used for workflow initiation, configuration, completion, and also ad-hoc customization (forwarding/delegation).

Workflows can be associated with lists, document libraries, or particular content types. These associations to WF templates are tracked in a farm-wide workflow association table. WF templates are defined by XML metadata and can include both workflow

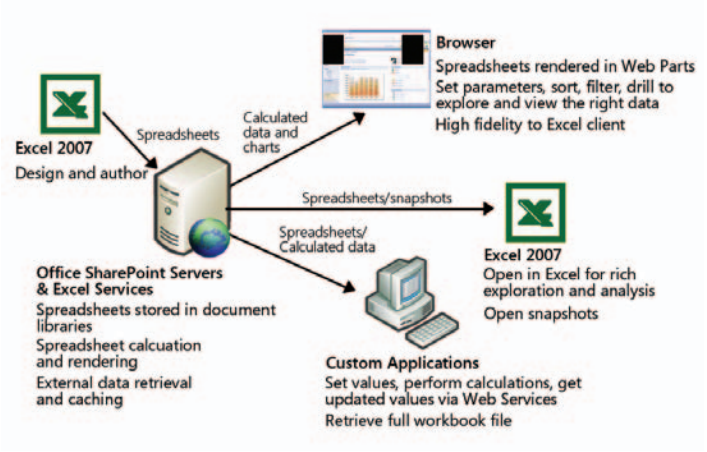
Figure 7 Workflow hosting with WSS



assemblies and forms. For example, when users create a document library, they can choose to associate that library with a particular workflow and specify the conditions under which the workflow gets triggered (for example, a document in a given library might have been modified or created). This workflow could support a particular business process or support document life-cycle management.

Integration of the 2007 Microsoft Office system with WF provides a variety of benefits (Figure 7). Simple business processes can be automated in a way that is seamlessly integrated into the SharePoint UI. Users are empowered through self-service capabilities, such as support for a broad range of user-controlled document routing and tracking scenarios, in a way that reduces the involvement of IT staff in putting together simple applications. WF also provides vertical solutions providers with an extensibility point to deploy their own business rules and logic into the containers provided by the 2007 Microsoft Office system.

Figure 8 Excel Services in the 2007 Microsoft Office system



Finally, the productivity tier must also provide a lightweight way to create and publish information and reports. This is supported by the 2007 Microsoft Office system, which integrates into SQL Server Reporting Services and provides the following components:

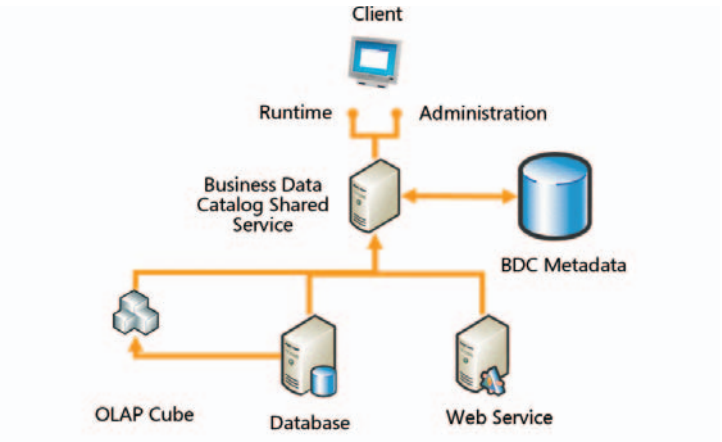
- **Report Center**—A template to create WSS reporting sites
- **Report Library**—A document library with special support for storing reports
- **Dashboard—**A WSS page assembled from reporting Web Parts
- **Report Viewer**—Web Part to view reports made available from SQL Server Reporting Services
- **Web Part**—to view Excel graphs and tables
- **KPI (Key Performance Indicator) Web Part and List**—Information workers can choose to source metrics for this in several ways.

A dashboard might be provided to users as a reporting template built around a specific business function, such as sales, marketing, or inventory management.

Composition in the Application Tier

Typically, structured business logic lives within the application tier. This might be an LOB application such as an Enterprise Resource Planning (ERP) system, or an orchestration across multiple systems such as a BPM

Figure 9 Business Data Catalog (BDC)



system. The application tier will typically include both transactional systems and decision-support systems. There are multiple ways for OBAs to enable composition in the application tier, as well as to consume composite services exposed by other platform technologies (Microsoft and non-Microsoft).

The first level of composition in the application tier is through packaged activity libraries created using Workflow Foundation, and deployed into the 2007 Microsoft Office system. Previously, we discussed how workflows can be attached to lists, document libraries, and particular content types. Figure 7 shows how the WF run-time provides a container for application-specific activities that are packaged and deployed as activity libraries, and on top of which workflows are assembled.

The second level of composition in the application tier provided by the 2007 Microsoft Office system is through Excel Services. This is a server-side Excel calculation engine that is built into SharePoint Server. It provides browser access to live, interactive spreadsheets that are deployed on the server, and also Web service access to server-side Office Excel calculations. With Excel Services, existing Excel power users can now provide server-side application logic in a way that is familiar to them. This means that MOSS is now a container for application logic, as shown in Figure 8.

A third way for OBAs to enable composition in the application tier is to consume composite services exposed by other platform technologies. The 2007 Microsoft Office system can be plugged seamlessly into a services-oriented architecture. If the enterprise has a services backbone already under development, these interfaces can be consumed from the 2007 Microsoft Office system. This can be done in a couple of ways. The first is by invoking Web service interfaces in activities that have been built into workflows deployed into the productivity tier. The second is through the Business Data Catalog (BDC), which is described in the next section.

Composition in the Data Tier

The 2007 Microsoft Office system also comes with a Business Data Catalog (BDC) that runs within the server as a shared service. It can read data from multiple types of data sources—databases, analysis services cubes, and Web services—and then display this data in the portal through SharePoint tables and lists. This acts as a metadata repository

for descriptions of business-data entities and their attributes, and for mapping these entities back to data stores within the enterprise, as shown in Figure 9.

While the BDC cannot be used to create an entity that maps across multiple data stores, it is possible to define relationships that link entities—such as parent-child relationships. So, the BDC can be used to create lightweight linkages between data entities across the enterprise—almost like an enterprise thesaurus. BDC-defined entities can then be plugged back into the 2007 Microsoft Office system, for example in SharePoint lists. This allows users to compose pages with linkages to back-end data and to traverse data tables by following relationships between entities.

Although the BDC enables data composition, it provides read-only access to data. However, users can use the BDC to model actions that can be taken on a data entity, where an action is defined by a name, a URL, and a set of attributes from the entity definition to be passed back to this URL. This can then be used from within a drop-down menu on a SharePoint list. The URL can correspond either to a Web service or to a server-side document such as an InfoPath form, with code-behind to pre-populate the form from the context that gets passed in from the BDC. Information workers can then use the portal to create lightweight applications in SharePoint with tables and lists that extract BDC data and actions.

Table 2 List of application assets for composition

- Open XML Documents
- Workflows
- Business activities
- Business rules
- Schemas
- Metrics
- Web Parts
- Dashboards
- Sites
- Pages
- Data connections
- Authorizations
- Reports

Summarizing capabilities for composition

Figure 10 shows how some of the capabilities of the 2007 Microsoft Office system map to the tiers in Figure 2. Table 2 provides a list of asset types that are candidates for composition.

Building an Office Business Application

There are two steps to building a standard business process as an OBA.

1. Build a process pack, which contains process metadata, and packaged application components.
2. Deploy the process pack onto production systems.

Both of these steps are described in detail below.

Build a process pack

1. Build user interfaces for document-centric processes into Microsoft Office client applications, using Visual Studio Tools for Office (VSTO).
2. Build WSS sites with task-specific Web Parts, pages, dashboards,

Figure 10 Capabilities of the 2007 Microsoft Office system application platform, mapped to tiers

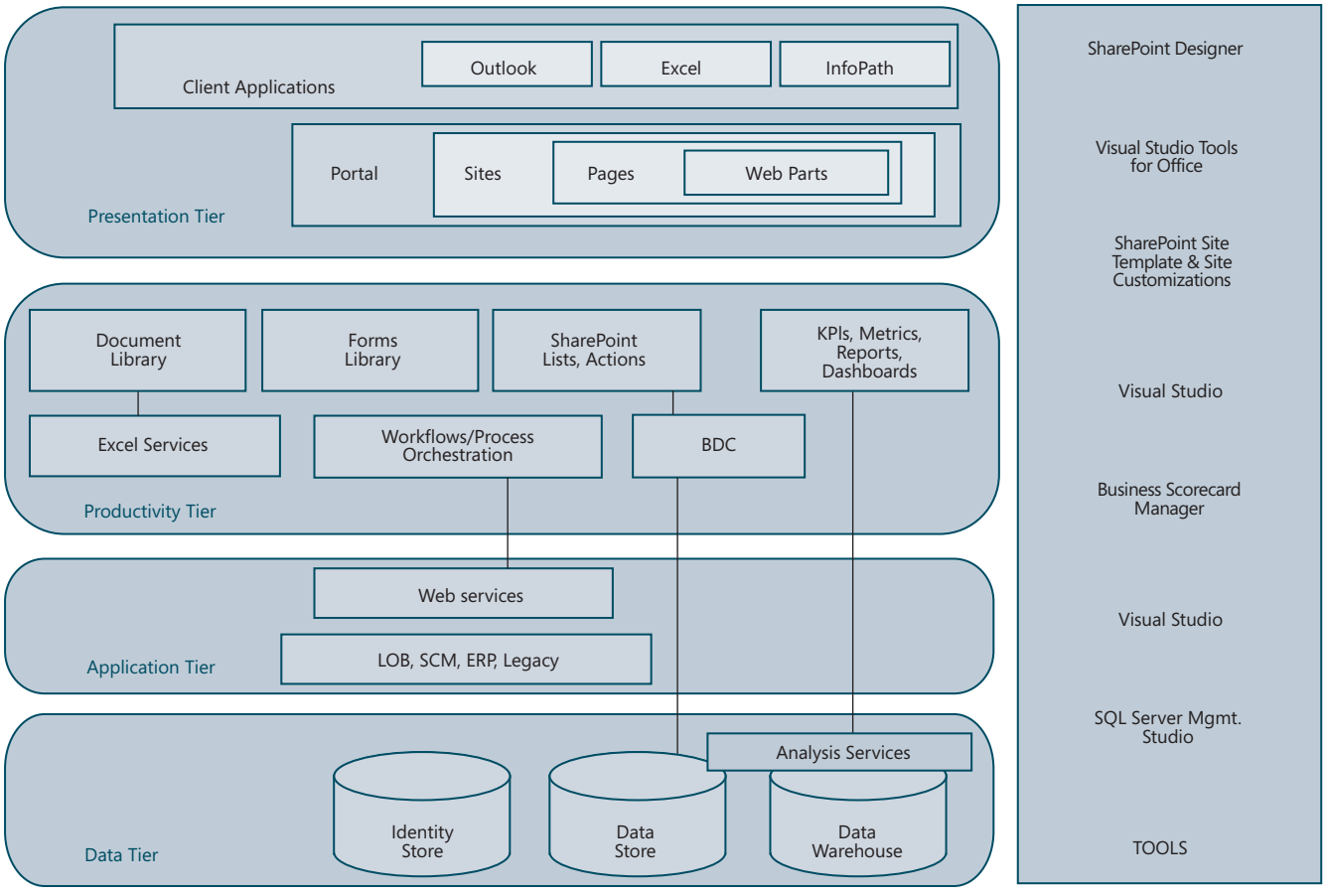
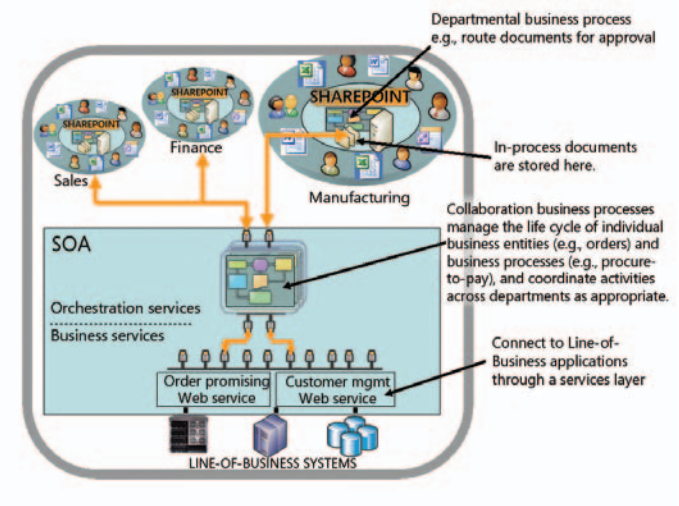


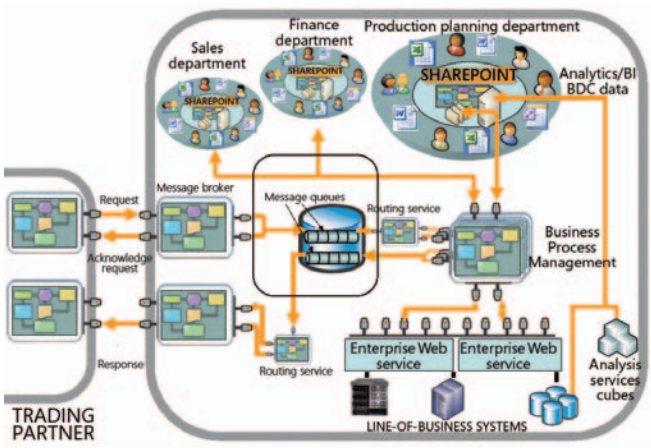
Figure 11 Provision OBAs in departmental sites for team collaboration, coordinate activities with business-process models, and connect to LOB systems through a services backbone



lists, and document libraries, where each site is designed for a particular business function or process. These sites can be used to create site templates for packaging a standard business process into an OBA solution.

3. Use Workflow Foundation to wire together lists and document libraries in the sites, with server-side rules and business logic for server-side processing of in-process documents. Package these workflows into assemblies for deployment.
4. Define touch points from workflows in the OBA to back-end LOB systems. The process pack metadata should include the Web services interfaces for this integration. The actual connections will need to be made during the deployment phase.
5. Define BDC entities for data connections required for cross-functional processes built into the OBA.
6. Add decision support by defining metrics, reports, dashboards, and Excel charts and tables to be used by the WSS sites.
7. Package solution as a process pack (WSS site templates, WF assemblies, Office documents, and so forth.) using the component types in Table 2.

Figure 12 Connecting the OBA to systems at "the edge"



Deploy process pack to production systems

1. Deploy WSS sites to the SharePoint server on the production system.
2. Configure connections from workflows to LOB applications using Web services—or other custom adapters.
3. Configure data connections by connecting BDC data entity definitions to actual data stores.
4. Provision users.

Deploying a composite application in the enterprise

One way to deploy an OBA in the enterprise might be as follows:

1. Deploy SharePoint sites at the departmental levels.
2. Connect multiple departments.
3. Connect business processes to LOB applications.
4. Add data connections for cross-functional processes.
5. Connect business processes to systems "at the edge."

Build departmental SharePoint sites to host local documents and processes

SharePoint sites must be set up at the departmental level to enable team collaboration, as shown in Figure 11. These sites will have document libraries to store in-process documents. Information workers on the team will have their own personalized pages, customized from the templates available on the team site.

Note that Figure 11 is a logical view of the architecture, as all these departmental sites would not typically be running on separate servers. Instead, multiple sites could be running on a single server in line with Figure 5, and the physical architecture (that is, the deployment landscape for SharePoint servers) would be chosen based on other factors like load, availability, and geographic dispersion of teams.

In-process documents are stored in document libraries and are associated with workflows that get invoked whenever a document is created or edited. Such a workflow might run validation rules on documents; apply approval policies and actions to the data; cleanse, validate, or filter the data contained within; or update back-end systems.

In addition to business process workflows, in-process documents undergo a life cycle of their own, from authoring and collaboration through management and publication, to archiving or destruction. Whenever a document reaches one of these stages, the appropriate workflow can be set to be triggered, such as for managing the archival process.

Connect multiple departments

As shown in Figure 11, business-process models coordinate activities both within a team and across departments. Within a department, business processes can be modeled using WF activities that are deployed into the SharePoint server that is supporting that department. Coordination of activities across departments can be accomplished using collaboration processes that manage the life cycle of individual business entities (orders), and also the life cycle of business processes (procure-to-pay).

The interdepartmental business processes could either be located centrally in IT data centers, or closer to the information workers within the departmental servers. For example, long-running workflows running centrally might be running on Microsoft BizTalk Server, whereas departmental processes would be running in WF workflows within the SharePoint server.

Connect business processes to Line-of-Business applications

Service orientation is one of the ways to expose current business applications into an OBA.

In that sense, SOA promotes modularization in the application tier, which is a basic requirement for composition, and which is why OBAs and SOA are complementary solutions. This allows the assembly of new cross-functional business applications that extend beyond the boundaries of current applications.

SOA is not the only way to connect LOBs to OBAs. A services backbone can also be exposed using other integration technologies, such as custom adapters.

Add Data Connections for Cross-Functional Processes

The BDC (Figure 9) can be used to connect back-end data stores to the 2007 Microsoft Office system by displaying data in SharePoint lists and Web Parts. This makes it possible to build composite applications for cross-functional processes into the SharePoint portal, using a combination of BDC, SharePoint lists, and Workflow. For example, the BDC can be used to define entities that have a parent-child relationship (such as order header and order details), and SharePoint lists could display them. It would be possible to drill down from the list displaying header information, to the corresponding details information, by following the parent-child relationships. Furthermore, actions can be modeled in BDC metadata, which means that these are displayed as menu items on the SharePoint list and selecting a dropdown from this menu will mean that context from the currently selected row will be passed into the URL defined for the action.

Connect Business Processes to Edge Systems

Often the scope of an OBA is not contained within an organization. For example, an OBA might support a business process that needs to consume a service that is offered by a hosting provider (Software as a Service—SaaS—scenario). Alternatively, the OBA might need to support a business process that offers a service to another organization. This is especially common in supply chain management scenarios, where trading partners are involved. Here, there needs to be a way to transfer documents from one information worker to a counterpart in the trading-partner organization. This needs to be secure, reliable, asynchronous, and transparent.

One way of putting together an end-to-end architecture for this is shown in Figure 12. Message brokers have been set up at the edge of the organization to send and receive messages and documents from trading partners. Messages from different trading partners can potentially be received in multiple message formats and delivered over multiple channels, such as Web services, EDI, e-mail, RosettaNet, and so on. Furthermore, messages can be exchanged in a variety of different patterns: one-way, asynchronous two-way, or synchronous two-way messaging. These message brokers must handle each combination of these message interchange patterns and message formats.

After the message is received by the message broker, it is processed into the single canonical format that is required for downstream services, and the transformed message is persisted to a message queue to decouple public processes from private ones. Next, the message is retrieved from the queue by a routing service that examines the message and routes it to the intended recipient. But before the document reaches its intended recipient, it might have to be preprocessed by enterprise application services such as LOB applications, or BPM orchestrations. Business rules might be applied to messages, to ensure validity and to enforce corporate policies. The result of all this processing is a document that can be processed by a human with enough information to make a quick decision. For example, a purchase-order request from a customer might be fed into an order-promising service, and the response from this service might be used to generate an XML document that corresponds to an InfoPath form with a candidate PO Confirmation. Next, this generated form might be placed into a SharePoint forms library for an information worker in the sales department to approve.

After the information worker has reviewed the proposed confirmation and made changes if necessary, the worker submits the form. This kicks off the workflow for the return trip, which updates LOB systems and then posts the information from the form as an XML document into a queue for outbound messages as a response to the original request. The message broker then converts back into the format used by the trading partner.

There are multiple ways to implement the message brokers at the edge, such as BizTalk Server. This would provide a scalable and manageable solution that would also come with standards-based accelerators and adapters, such as the RosettaNet accelerator for trading partner collaboration. The queues that decouple internal and external processes could be implemented using Microsoft SQL Service Broker 2005.

Sample Office Business Application

To provide technical resources and guidance on building OBAs with meaningful business value, a reference implementation for an OBA is available online at http://msdn2.microsoft.com/en-us/architecture/aa702528.aspx. This Reference Application Pack for Supply Chain Management covers scenarios for collaboration between the different levels of a multi-tier supply chain.

Conclusion

The 2007 Microsoft Office system provides a powerful set of capabilities to build composite applications, which are called Office Business Applications (OBAs). This enables cross-functional solutions offering a composite user interface that exposes business functions and capabilities across a heterogeneous set of back-end IT assets. These solutions also provide collaborative business capabilities that span the gap between traditional business applications and personal productivity tools.

About the Author

Atanu Banerjee is an architect on the Architecture Strategy Team at Microsoft. He has ten years' experience in the software industry, having worked earlier on solutions for supply chain management and advanced process control. He joined Microsoft from i2 Technologies, where he served as chief architect for its supply and demand management product line. Before that, he worked at Aspen Technologies' advanced control systems group. He received a Ph.D. from Georgia Tech in 1996, and holds a bachelor's degree from IIT Delhi, India.

Architecture Journal Profile: Scott Guthrie

Scott Guthrie is a general manager in Microsoft's Developer Division. He runs the development teams that build CLR (Common Language Runtime), ASP.NET, WPF (Windows Presentation Foundation), WPF/e, Windows Forms, IIS (Internet Information Server) 7.0, Commerce Server, .NET Compact Framework, and the Visual Studio Web and Client Development Tools. As part of the new Architecture Journal Profile series, Ron Jacobs sat down with Scott to ask him about his career and thoughts regarding architecture.

RJ: Today we're going to talk about you and your career for people who think, "This sounds like a cool job." What's your role now at Microsoft?

SG: I run our .NET developer platform group. This group includes the CLR (Common Language Runtime), the .NET Compact Framework, IIS (Internet Information Server), ASP.NET, Atlas, Commerce Server, Windows Presentation Foundation, Windows Forms, and our development tools for targeting Web applications in Visual Studio.

RJ: Wow, that's a lot of surface area!

SG: It's a lot of fun. It encompasses our core application models, the runtime, the tools, and all the engines they run on top of. It's a lot of cool stuff, and a lot of things to play with.

RJ: Most people will remember you from your association with ASP.NET. Let's go back to the early days here at Microsoft. How did you get started?

SG: I started with the IIS team back in '96-'97, working on our core Web server technologies, and was involved in shipping a version of IIS. After IIS 4 was released, we started looking at next-generation Web programming model pieces. At the time, we thought, "Maybe we're done. Is there anything left to do in terms of feature set?"

We started talking with a lot of customers and looked hard at the types of applications they were building. We very quickly learned that there was still a lot left to do. People were struggling with code/content separation and how to write clean code. We used to joke that "write once, read never" code was being produced. From a tooling and runtime administration perspective, there were many challenges in making our existing infrastructure work really well. To help make this happen, we formed a small team to think about future architectures with IIS. This is the team that invented the HTTP.SYS kernel driver that we introduced with Windows Server 2003. With a colleague, I started looking at Web programming model pieces and wrote the initial prototype for what became ASP.NET.

RJ: It seems that you were thinking of taking this to the next level and taking advantage of something that was supersecret at the time. I remember those days of .NET; you used to call it ASP+ back then.

SG: We originally called it XSP; and people would always ask what the X stood for. At the time it really didn't stand for anything. XML started with that; XSLT started with that. Everything cool seemed to start with an X, so that's what we originally named it. In the first six months, we didn't use .NET. The CLR didn't exist—it was just starting around the same time we were—so we were doing most of our prototyping in C++, JavaScript, and ActiveScript script engines. We knew we wanted an object-oriented environment, and we really liked the characteristics a managed programming model provided in terms of garbage collection, nice encapsulation, and object-orientation techniques. We actually started writing production code in C++ though, because at the time we didn't really have a good runtime platform on which to build. We got about two weeks into it when we met up with the CLR team; at the time that team had no partners inside the company building on top of them. The only compiler they had was this thing called "simple managed C," which we affectionately called "smack." We ended up saying, "Maybe we should build on this." It was a huge risk, and at the time our team consisted of three or four people total. We were allowed to take a bet on it mainly because nobody really cared if we failed. Thankfully, we did and it paid off in a huge way. The rest is history, so to speak.

RJ: I remember those days. In the early days of the CLR, few people were willing to take that bet. Many teams said "I don't think so," but you guys did, and it paid off in a wonderful way.

SG: Yeah, people were just terrified of the whole idea of garbage collection on the server, which we take for granted today. They said, "There is no way you can build an app with garbage collection running in the background. Your server will never scale." There were lots of doomsday scenarios. From a project perspective, one of the things we did paid off in a huge way. We said, "We're going to bet on managed code, and it won't be a wrapper around some native stuff. We're going to bake it deeply into the platform. We're going to write about 95 percent of our code in managed code itself."

The reason we did that was twofold. One was to take full advantage of the extensibility it provided and really bake object-oriented

extensibility very deeply into the platform. And, second, we knew that customer applications would be managed code and that our percentage of code on a call stack would be relatively small compared to the customer's share. If even *we* didn't think we could write in managed code, we were kidding ourselves if we thought customer applications would sail. It was a great forcing function. From day one through creating more complicated samples, we were tuning the core CLR engine along the way, and that translated into a huge customer savings when we started getting more complicated customer applications on top. It was a good bet.

RJ: It's interesting that you look at this as a way to drive improvement down into the engine. You said, "Not only are we doing this, but we're making the engine better by doing so."

SG: I think that was a huge bet, but it was an approach that worked out well. The fact that we were a small team and starting with a new code base helped tremendously. If we had been a larger team or had an existing large legacy code base, it would have been more difficult because COM interoperability didn't exist back then. But the fact that we were starting from scratch and able to start small really helped drive those core improvements deep into the engine. As we got bigger and our feature set flushed itself out, it just kept paying dividends.

RJ: I think it's cool that the people who were managing you let you make that call.

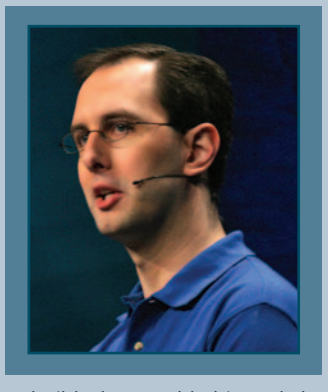
SG: It was definitely a gamble, but it was a calculated one. There was a huge upside if we could make it work, but the downside was that there were three or four of us, and we could always do something new the next year. Microsoft has often made these big bets, and usually they pay off in a big way. Occasionally they don't—and they can fail spectacularly—but as a company we try to make sure that we bet big on a couple of key things.

RJ: Did you have to persuade anybody to let you take the risk, or was it easier than that?

SG: We certainly had to persuade a number of people along the way. One thing we did early in the project was get running code in prototypes that we could show people. Often when you're working on a project that's new or something that hasn't been done before, it's easy to put together a bunch of PowerPoint slides that sound good, but it's especially valuable to actually show code and walk people through code that's running. Not only does the prototype prove that it's real, but also you just learn a terrific amount by doing it.

One of the things I try to do with my team is to prototype early, build sample apps early, especially demo apps that we can walk the customer through and say, "Look, here's how you can build an app." We try to do that as early as possible to learn what works and what doesn't, so we can react accordingly. About a month and a half into the ASP project I wrote a prototype, and we were able to walk people through. Here was this component control-driven model—we didn't call them controls back then; they were declarative tags or components—and here was this event-driven way to program for the Web. We were able to build apps and discovered quickly that some of the things we had come up with were really impossible to code. At the same time we learned that "it would be really cool to have this feature, or that feature," and we iterated along the way. That helped tremendously when we were trying to help people realize that we weren't complete lunatics. We were able to show code, and they were able to get it. It was still a gamble.

Scott Guthrie's Career at Microsoft



Scott Guthrie joined Microsoft in 1997 and first worked on IIS4 and the Windows NT Option Pack. Shortly after its release, he designed and prototyped a new server programming model originally code-named "XSP," and together with Mark Anders subsequently formed and staffed a new team in 1998 to build what would ultimately be called ASP.NET.

Scott became production unit manager (PUM) of the ASP.NET team in early 2002, and shipped ASP.NET V1.1 as part of Windows Server 2003. During this time he also led the incubation of the popular Web Matrix development tool, a free ASP.NET development tool that helped spark new thinking about tools for Web development, as well as a new approach for targeting programming hobbyists and enthusiasts. In late 2002 he also became PUM of the Web tool features within Visual Studio and was responsible for the development of the new Visual Web Developer standalone product that will ship as part of the Visual Studio 2005 family, as well as all Web development features in Visual Studio. Visual Web Developer and ASP.NET 2.0 entered their first widespread public beta in the summer of 2004 and will ship in the first half of 2007.

In late 2003, Scott's team merged with the IIS team, and he became PUM of a unified Web Platform and Tools team that combines the assets of IIS, ASP.NET, and Visual Studio. Concurrent with finishing up ASP.NET 2.0 and Visual Web Developer, the team is now actively developing the next major version of Microsoft's Web Application Server, which will ship as part of Longhorn.

Now a general manager in Microsoft's Developer Division, Scott runs the development teams that build CLR, ASP.NET, WPF, WPF/e, Windows Forms, IIS 7.0, Commerce Server, .NET Compact Framework, and the Visual Studio Web and Client Development Tools.

Scott graduated with a degree in computer science from Duke University in 1997.

RJ: It almost sounds like the test-driven development mindset. Let's do short iterations; let's get to something that works. We'll eat our own dog food, and we'll write against our own APIs to understand what it feels like to use them.

SG: It's definitely the same type of principle. I differentiate between test-driven development as a methodology for how you can drive quality early and how you can provide a base that lets you refactor and adapt your code base without having to worry about regressions. We certainly follow that philosophy internally when we develop production code. I think there's also value in doing a prototype phase even before you get to production code. That's one of the successful things that we did with ASP.NET. We said that we're going to throw away every line of code we're going to write for the next couple of months. Let's all agree on that. We're not going to say, "Oh let's take this and adapt it; we can clean it up." No. We're going to throw it away. We're going to "deltree" this subdirectory at some point, and that way we can be more adventurous about trying new things. We don't have to worry about making sure that everything's robust because it's going to be in the final version.

We actually did that for a few months and said, "We're done, delete it; let's start over from scratch; now let's write the full production code and make sure we bake in quality at the time." I think a lot of teams could benefit from that. The hardest thing is making sure you delete the prototype code. Too often projects develop with "Well, it's kind of close." It's very difficult to start with a prototype and make it robust. I'm a firm believer in starting with a prototype phase and then deleting it.

RJ: It demonstrates that we value the learning more than we value these files and these bits from the prototype phase.

SG: Every time you work on a project, if you rewrite anything, whether it's from scratch or not, the code gets better. Partly it's because you understand the problems and pitfalls of the last approach and can reflect and improve on it. The challenge is that you can't easily do that time and time again. But when you're first starting out on a project or a brand-new area where it isn't clear how you get from Point A to the finished product, having a dedicated period where you prototype and try things out is supervaluable.

RJ: Some people call that the "architectural spike." Here's a new area, and we're going to explore. Changing directions, though, what were you doing before you came to Microsoft?

SG: I actually joined Microsoft right out of college. I interned with Microsoft while I was in college. I was involved in a couple of startups in college and high school, did some development, and had some fun there, but I joined Microsoft straight out of college.

RJ: We've been talking to a lot of people who find architecture interesting. There seem to be very few people who are purely architects in that they just design stuff and never write code. Most people are a mix: They spend some time developing, some time architecting. What advice would you give to somebody who has been doing development but wants to do more architectural thinking?

SG: Writing code is valuable for an architect. Not necessarily production code you check in, but constantly trying out new technologies, new approaches, and feeling how the system works. I don't write a lot of production code these days, but I spend an hour or two every day writing code. It may be samples, prototypes, or some fun personal project—whatever it is, I'm trying things out, thinking of ways to structure things. Being hands-on is very valuable from a code architect's perspective.

The other thing I would recommend is taking a look at core systems theory and how to architect very robust systems. Consider some of the principles you want to think about and apply them as you're doing it. That doesn't mean thinking about what the lines of code look like, but thinking about simplicity, or robustness, or fault tolerance. Those types of things are core in successful systems; whether it's a client application, server application, or a game. An architect who thinks hard about those kinds of principles and can marry them with a good coding background can provide a tremendous amount of guidance to teams.

Those principles are not about playing with a wizard or checking out cool new stuff, but studying how the process-address space works in a Windows or Unix application. What is threading and how do you deeply internalize what it looks like on a multiprocessor or multicore system? It's about absorbing that type of knowledge, thinking about its ramifications, and spending some focused time thinking hard about the trends, where the technology is going from a hardware and a software perspective, and considering how to adapt and take advantage of it. That's what I recommend doing.

RJ: At Microsoft we have developers, program managers, and architects. People are often curious about the role of the architect. What's your expectation for the role of the architect on the team?

SG: There are a couple of responsibilities that we hope or expect an architect to bring to a team. One is a very deep, solid background in architecture, development, and the software principles I was talking about. With that type of background, our hope is that a process of osmosis will take place—that some of it will rub off on other team members. Hallway conversations or informal office chats can provide a tremendous amount of leadership to a team, especially when you supervise junior and senior developers.

We look for an architect to pave the way regarding what the product should be doing from a technical perspective. Often architects do more advanced prototyping work and investigations of where we should take the product. We look to them to recommend where we should go, and from an implementation perspective, we ask them to look at both the next-generation product and the current product to identify areas we should clean up. For example, which areas should we factor slightly differently? What are some practices we can implement throughout the code base to improve it?

RJ: In addition to deep, solid, technical skills, what other attributes do you think contribute to a successful architect?

SG: The hardest thing, at least at Microsoft, is that very deep technical people who want to go up the architect track need to make sure that they can marry their technical skills to an ability to work both within and across teams in a company.

Some of those softer skills are harder to build, meaning that an architect needs to be hands-on, but to do so in a way that doesn't threaten developers or other teams. They should also avoid "I own this, you own that" conversations. Architects have to be able to

work across teams very flexibly. They need to do so in a way that doesn't leave people feeling as if the architect is just diving into the most interesting problem for the moment and then flying off when things get hard. Other team members have to believe that the architect is committed to the team and is part of a long-term relationship that provides value on a problem. Those are the types of skills an architect needs to develop. The super-senior architects who have the biggest impact can marry deep, deep technical and design skills with people skills and collaborative abilities.

RJ: A lot of people tell me is that the rate of change is accelerating and new stuff is coming out all the time. You mentioned how important it is to stay up-to-date on these things, but there are only so many hours in the day. How do you stay up-to-date?

SG: It's hard, especially in the development space. When I think of the pace of innovation that's going on right now and the rate of information flow, I certainly can't remember a time when it was going this fast. I think back to a time, the Internet battles of the '90s, when Internet Explorer was competing with Netscape. At the time, it felt as if we were shipping constantly, and there was a lot going on.

From the development perspective, I think we're in a phase right now where the pace is even more accelerated than it was then. It is certainly very hard to stay up-to-date. You have to find time to do it. You have to spend focused time keeping an eye on what is happening. I think blogs are a great mechanism for doing that. I subscribe to Bloglines, which is a great free service. I probably subscribe to 300 or 400 blogs, and I try to spend 20 to 30 minutes a day in the morning and the evening reading through what everyone posts. It gives you a good sense of what the hot topics and the interesting ideas are.

Part of keeping up means spending an hour a day of focused time doing prototyping; trying things out, either with your own product or other technologies; getting a good grasp on what pieces are out there and how you can use them. The other important task, when you're looking at any new technology, API, methodology, or programming approach, is to look hard not just at the interesting thing itself, but also try to extrapolate its useful principles so you can apply them elsewhere. So if it's a Java refactoring book, great. There are some specific Java refactorings there, but what are the broader refactoring concepts you can internalize and apply to VB or C#? If it's an AJAX JavaScript framework that's very good at doing one specific task, great. Now, step back and try to recognize which of its aspects could be applied on another JavaScript framework. An architect should be good at looking at something and extrapolating the interesting aspect in and of itself, as opposed to the individual element of the technology.

RJ: As you look back over your years here at MS, is there anything you regret?

SG: As you look back, there are things you would do differently. Sometimes it might be a technical thing you've done, the way you've implemented a feature, and you think, "Everyone abuses that feature." Or they do things not quite the way we intended them to be done. Certainly, when we've built a development platform as broad as .NET, I could come up with a dozen or so things that in hindsight that I wish we had done slightly differently. There are also ways that you approach things, or ways that you work with different teams and think, "Gosh, I wish I'd handled that conversation slightly differently." So there are definitely lots of individual examples I could come up with.

Overall, I'm happy about where .NET is, so we've been fairly successful with where we've taken it. But there are lots of things I wish we'd done slightly differently, such as "Gosh, I wish we hadn't sealed that class," or, "Gosh, I wish we hadn't unsealed that class."

> "THE SUPER-SENIOR ARCHITECTS WHO HAVE THE BIGGEST IMPACT CAN MARRY DEEP, DEEP TECHNICAL AND DESIGN SKILLS WITH PEOPLE SKILLS AND COLLABORATIVE ABILITIES"

If there's one significant thing I wish we'd done differently, it would be that we'd spent more time early on thinking hard about the client-installation process for building .NET client apps. I think the approach we took with a single redistributable that you download is not any worse than any other Windows redistributable, but I wish we'd taken the opportunity early on to get a less impactful installation and simplify client app deployment. That is something we're spending a lot of time on right now, and it's going to get dramatically better in the future, but I wish we'd done that six years ago and spent more time thinking through some of those scenarios a bit earlier.

RJ: In your office you have all these speaker badges going way back, and you've had chances to go many places and meet a lot of people. What highlights have you had? Is there anybody in particular that sticks out in your mind?

SG: One of the things that's fun about working on a developer platform is just seeing the range and diversity of apps people have built on our stuff. Whether it's MySpace, which is the largest social networking platform in the world—there are a billion-and-a-half page views a day using .NET—or the London Stock Exchange or National Health Service of the UK, or a whole bunch of companies on Wall Street, Costco, Dell.com, or Match.com, there are tons of cool customer apps built on Microsoft technology. Many of these are on the Web; others use different technology. If you go to the Walt Disney World properties, the meters that run the "Fast Pass tickets" run on the compact framework and the CLR. If someone knocks on the door from the U.S. Census or the U.S. Postal Service, the device that person is holding also runs on the .NET framework.

That for me is the highlight: seeing how .NET is being used all over the place. Sometimes in weird, whacky ways, sometimes for mission-critical apps, but each time in a unique way that, frankly, you might not have thought of. I think the hallmark of a good framework doesn't lie in the applications people build on it that you had expected them to build, but in the fact that customers and developers were able to take it far beyond what you had imagined. For me, that's the highlight of .NET.

Architecting Composite Smart Clients Using CAB and SCSF

by Mario Szpuszta

Summary

Microsoft's offerings for building composite smart clients include Composite UI Application Block (CAB) and the Smart Client Software Factory (SCSF) from the Patterns & Practices Group. This article unveils the architectural details of CAB and SCSF and shows you how to design composite smart clients using CAB and SCSF. The examples are from an integrated bank desktop smart-client project undertaken at RACON Software GmbH, a software house of the Raiffeisen Banking Group in Upper Austria.

Architectural Guidance for Composite Smart Clients

Smart clients are extremely useful if you want to reach a well-known set of users with high user experience requirements. Very often, we tend to reduce the advantages of smart clients to easy aspects, such as a cool-looking user interface. While this is definitely important, when taking a look at the definition on MSDN, there are more vital factors. According to this definition, a smart client is a Windows application compliant with the following criteria:

- In addition to providing rich user interfaces, a smart client uses *local resources* available to the clients, including hardware resources as well as components and locally installed applications.
- It is a connected application exchanging information with services on the Internet or the local enterprise network.
- Although a connected application, a smart client is offline—capable of making connectivity as transparent as possible to the user.
- Last but not least, intelligent deployment and updating are key criteria for a smart client. This includes mechanisms such as ClickOnce (no-touch) deployment and automatic updates.

While these general concepts apply to smart clients, enterprises often have additional requirements. Their users want to work with integrated desktops—a common shell that hosts different types of applications, with seamless integration and a common, streamlined way of working with them. Such types of smart clients are called composite smart clients. A composite smart client allows the client solution to be composed of a number of discrete functional pieces—so-called modules or plug-ins—that are integrated within a common host environment.

Composite Smart Clients—Real-World Scenario

I was personally involved in creating the architecture for a common bank desktop for Raiffeisen Banking Group, Upper Austria, in Linz. The Raiffeisen Banking Group is the largest private bank group in Austria with about 2600 bank branches. Software development for banking applications is primarily managed by RACON Software GmbH, which is mainly responsible for software of all the Raiffeisen banks in Upper Austria. Typically, these banks have lots of different applications, such as management of loan processes or applications for transactions in securities and foreign-exchange businesses. A requirement is that these applications are able to share common aspects, such as customer management, including search and access to their accounts.

After consolidating the services-side landscape into a large set of Web services, RACON Software GmbH realized it had a client-side landscape built with lots of different applications using different infrastructures. The result of this was duplicated functionality and the propagation of many different flavors of user interfaces throughout the bank. The goal was to introduce a common desktop as a new foundation for all of these applications to streamline the user experience and the maintenance of their banking applications. The primary business drivers included the need to lower the training and maintenance costs by minimizing the number of different applications and by consolidating common client services into a single infrastructure.

The idea was to create a "bank-shell" as a central entry point hosting all applications. Application access would be role-based, although the fundamental look and feel, and behavior of all applications hosted in the shell should be the same for every user. As some banks are equipped with low-bandwidth connections to the backbone, only, intelligent caching strategies were also required to keep the applications responsive. In addition, for some employees a disconnected, offline capability was a requirement. Finally, they required a rich user interface as well as integration with Microsoft Office applications installed locally on the client. With this setup of requirements, although RACON Software GmbH wanted to integrate existing Web applications into the bank-desktop, a pure Web application was not an option.

As we started to investigate the technical aspects, we found that the requirements for the architecture of the smart client needed to support the following aspects:

- Pluggable architecture to enable modules being loaded dynamically based on configuration and user/role assignment on application start-up or even later at runtime.

- An infrastructure for central management, registration, and configuration of common services, such as Web service agents required by all or the majority of the modules.
- Support for loosely coupled communication between modules and components of these modules based on a publish/subscribe pattern.
- An existing infrastructure for common patterns, such as Model-View-Controller, Model-View-Presenter, or the Command Pattern.

With the requirements in place, we started to look at frameworks that we could build upon to realize this solution.

Composite UI Application Block

Released in November 2005 by the Microsoft Patterns & Practices team, the Composite UI Application Block (CAB) is a framework for implementing composite smart clients. It supports most of the requirements mentioned previously and was our initial choice for implementing the "bank-shell" of Raiffeisen Banking Group in Upper Austria. The CAB provides the following functionality:

- Dynamically loading independent yet cooperating modules into a common shell based on a central configuration
- Support of the composition pattern at several levels for functional pieces such as user interface elements, user interface processes or client-side services
- Event-Broker for loosely coupled communication between functional pieces loaded into the client application
- Ready-to-use command-pattern implementation
- Base classes for MVC implementations
- Framework for pluggable infrastructure services, such as authentication services, authorization services, module location, and module-loading services

Unlike many other client-side application frameworks, the CAB is not based on a predefined shell-application that is extensible via plug-ins—it is much more a framework for building both the extensible shell-application and the modules that can be dynamically plugged into this shell. For this project it was useful, as it put the CAB into the role of a meta-framework on top of which you can build your own frameworks. Finally, the CAB makes it possible for you to decide how much you want to use from its offerings and how much you want to extend or adopt from them.

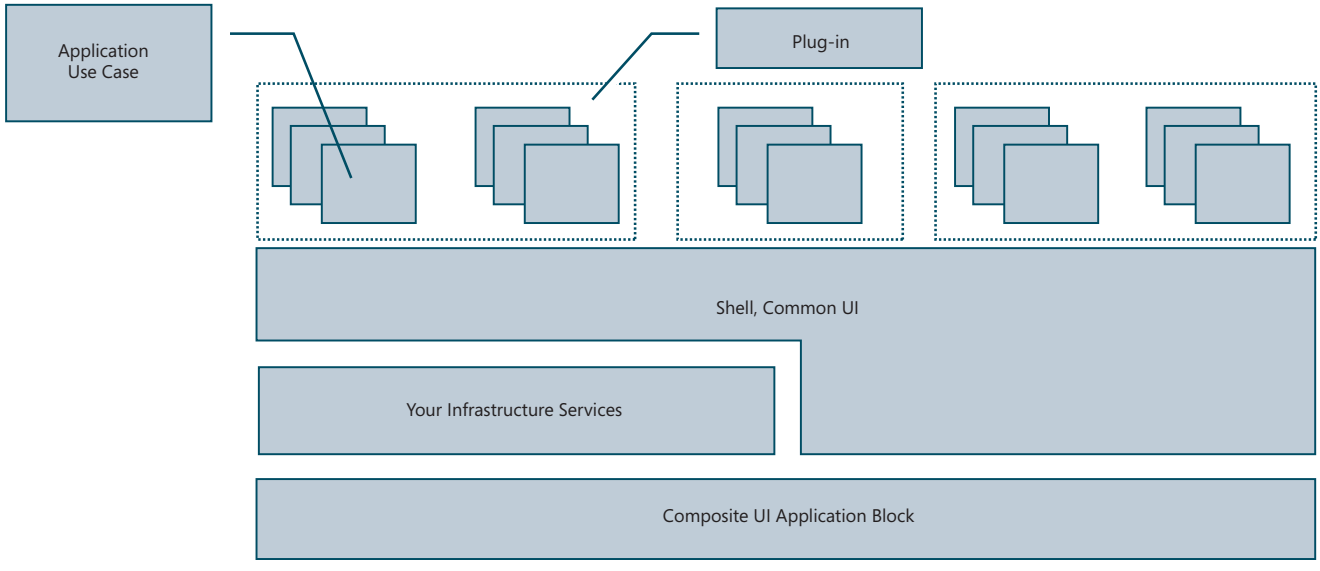
We decided that the overall design of a composite smart client based on CAB should adhere to a layered approach where CAB provides the fundamental framework with all the necessary functionality for building a pluggable, extensible architecture. The shell and the infrastructure libraries are your application foundation (your framework on top of CAB), whereas it uses CAB-functionality to dynamically load plug-ins (called modules) containing the implementations of the business-use cases. Figure 1 outlines a typical architecture of a CAB-based composite smart client.

Another interesting point is that the CAB object model enables a way of designing composite smart clients with a use case-driven approach. Its object model clearly separates use case controller classes from other components, such as views and their controller—or presenter—classes. The use case controller classes, called WorkItem, are responsible for putting all the necessary aspects for a use case together. This means they are responsible for managing the necessary state of a use case; for putting together the necessary views and controllers; for managing the workflow to complete a use case; and finally for providing the managed state to all of these components. Together with the layered architecture introduced in Figure 1, for our project it helped us better align the structure of the project team with the development processes for creating composite smart clients using the CAB.

We learned that three aspects affected the development process for CAB-based smart clients: the shell, the infrastructure services, and the actual use cases. We therefore recommended three primary iterations for development:

1. Start achieving a high-level understanding of requirements common to most of the use cases. Common UI-related requirements are candidates for being integrated into the shell

Figure 1: Layering of CAB-based smart clients

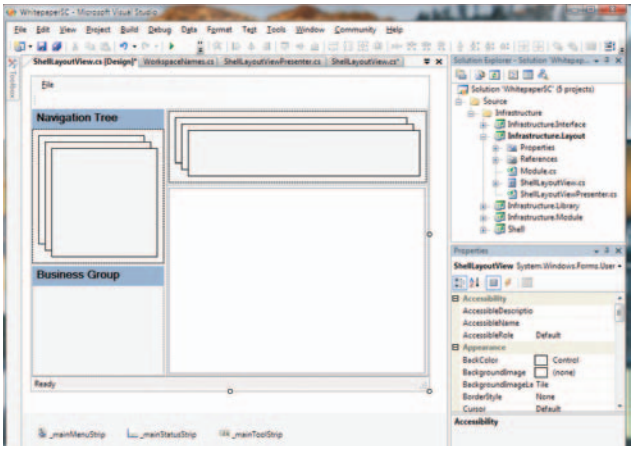


directly or at least affecting the shell's layout and design. Non-UI related functionality that is common to all (or most of) the use cases is a candidate for central services.

2. Create detailed use case diagrams. Use cases are good candidates for becoming WorkItems (and sub-WorkItems) in your application design. Relationships between use cases are candidates for either using the command-pattern or the event-broker subsystem of CAB. Logically close-related WorkItems, such as WorkItems implementing use cases for the same actors (roles of users in your company) are good candidates for being packaged into modules together.
3. Refine the use case diagrams; analyze relationships between use cases as well as reusability and security aspects of your use cases (WorkItems). During this refinement you might need to refactor your WorkItem packaging slightly. For example, typical findings are WorkItems that are reused independently of others and therefore should be packaged into a separate module.

The use cases identified during the detailed use case analysis are the foundation for identifying WorkItems in CAB. When taking the use case-driven approach for designing WorkItems, each use case maps to a single WorkItem. Any relationship between use cases in the diagram is an indicator for two things: First, it can mean a containment

Figure 2: A typical example for a shell with Workspaces and UIExtensionSites

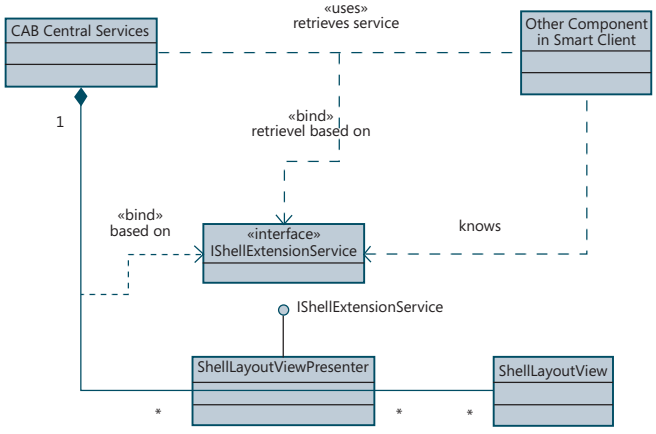


of one WorkItem within another one; second, it definitely leads to communication between these WorkItems. Communication can be implemented via services, the event broker, or commands. Typically, a refinement and a detailed analysis of the relationships between use cases are indicators of which type a relationship exists (parent-child or just communication between the WorkItems) and therefore will help in deciding which communication mechanism to use. This refinement will affect your strategy in packaging WorkItems as well. Therefore, it is essential to keep this refinement-iteration in mind before you start developing the broad range of WorkItems. There is more on WorkItem identification and packaging later in this article.

Designing Your Shell and Infrastructure Services

As outlined previously, one of the first tasks is the design and implementation of the shell. The shell is the actual application and is responsible for loading and initializing base client-services, loading and

Figure 3: Shell implementing your custom IShellExtension interface provided as a central service



initializing modules, as well as providing the basic UI of the application and hosting the views loaded by WorkItems. The CAB provides all the necessary base classes for supporting such functionality—a base class for a shell-application that is able to authenticate users based on a central CAB-authentication service and that is able to dynamically load modules based on the user's role from a so-called profile catalog, as mentioned earlier.

As the basic user interface of an application is provided by the shell and is equal for each and every module loaded into the application, the shell needs to have some extension points and must fulfill requirements common to all use cases. Typical examples for common user interface elements that are good candidates to be integrated into the shell are menus, toolbars, task panes, and navigation panes. The important point here is that some parts of the shell will just be extended (such as adding a menu to the menu bar) and some will be replaced completely as modules are dynamically loaded into the application. For these purposes, the CAB provides two types of shell-extensibility points you can use when designing your own shell: UIExtensionSites and Workspaces. Workspaces are used for completely replacing parts of the user interface, whereas an UIExtensionSite is used for extending existing parts of the shell, such as adding menu entries or adding tool-strip controls. Figure 2 shows a typical example for a shell designed in Visual Studio 2005.

Workspaces and UIExtensionSites are publicly available to all services and modules loaded into the smart client application. The CAB allows you to access these in a very generic fashion as follows:

```
workItem.UIExtensionSites["SiteName"]Add<ToolStripButton>(
                                new ToolStripButton());
```

However, to avoid tight coupling and errors, I recommend a layer of indirection between the shell and components that want to add the shell. Based on the functionality the shell needs to provide to other components, you can design an interface implemented by the shell (or the shell's main view presenter) for extending the shell and register the shell as a central service used by other components of the CAB as shown in Figure 3.

This is a very simple yet powerful concept because the components loaded into the smart client (either modules with WorkItems or

modules with central services) do not need to know details about Workspace- and UIExtensionSite-names. Components loaded into the smart client can access these services as follows:

```
IShellExtensionService ShellService =
    WorkItem.Services.Get<IShellExtensionService>();
ShellService.AddNavigationExtension(
    "Some new Menu", null, someCommand);
```

Still, the implementation of this IShellExtension interface can leverage the existing CAB infrastructure to keep the shell-UI-design decoupled from the shell-service implementation, but the developers creating the broad masses of WorkItems do not need to know any details, such as names of UIExtensionSites or similar shell-details. And, you can abstract other shell-related functionality through such a shell-service, for example, a messaging or a help-system integrated into the shell. But remember, it is important that the shell-interface exposes shell-UI-related functionality, only. That means it is the right point of access to allow components loaded into the smart client to extend menus, tool-strips, a task bar displayed in the left part of the Window, or add a message to a common message pane (as introduced in the examples earlier in this section).

Infrastructure Services

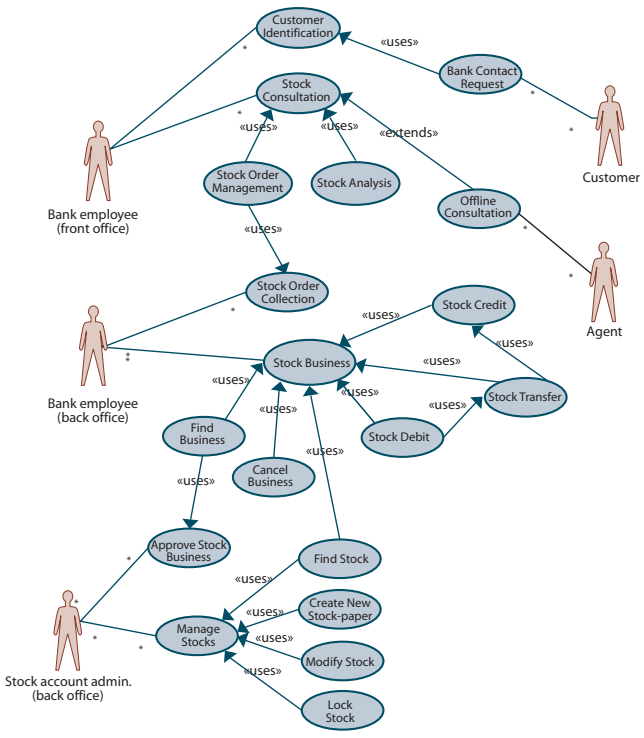
As mentioned earlier, infrastructure services encapsulate common functionality to modules and components loaded into the smart client. As opposed to shell-infrastructure, these services are not bound to UI-specific tasks. More often they encapsulate client-side logic. Out-of-the-box CAB introduces many infrastructure services, such as an authentication service and a service for loading a catalog of modules configured somewhere (by default in the ProfileCatalog.xml file stored in your application directory). Of course, you can introduce your own services as well. Typical examples for central infrastructure services not introduced by CAB are:

- Business functionality-related authorization service (CAB introduces authorization for loading modules only, but not for specific actions that can be performed within modules and the application)
- Web service agents and proxies encapsulating calls to Web services and offline capability
- Context services to manage user-context across WorkItems (see section "Context and WorkItems" later in this article)
- Services for accessing application-centric configurations
- Deployment services using ClickOnce behind-the-scenes for programmatic, automatic updates

We learned that it is always important to start with defining interfaces for your common services, as CAB allows you to register central services based on types as follows:

```
MessageDisplayService svc = new
MessageDisplayService(_rootWorkItem);
_rootWorkItem.Services.
Add<IMessageDisplayService>(svc);
```

Figure 4: A Sample Use Case Diagram



Infrastructure services need to be encapsulated into separate infrastructure modules loaded before other modules with actual use case implementations will be loaded.

Designing and Packaging WorkItems

WorkItems are components responsible for encapsulating all the logic for specific tasks the user wants to complete with the application. As such, WorkItems are the central and most important parts of your composite smart client, as they provide the actual business functionality to the users. CAB is able to manage a hierarchy of WorkItems that are responsible for completing work together. For this purpose a WorkItem contains or knows about one or more views (called SmartParts) with their controller-classes and models. The WorkItem knows which SmartParts need to be displayed at which time and which sub-WorkItems need to be launched at which time. Furthermore, a parent-WorkItem is the entry point into a specific task. Finally, it manages the state required across SmartParts and sub-WorkItems.

During development we found that there are two ways to identify WorkItems for your composite smart client: a use case-driven approach and a business-entity-driven approach. The best way to explain this is with some examples. (Please note that for the purposes of this article these are not representative of the project I was involved in, or any other banking group.)

Use Case-Driven Strategy

One of the biggest advantages of the architecture of CAB is that it allows you to design your WorkItems based on use case diagrams. Often you will have a one-to-one mapping between use cases and WorkItems—typically one use case will be encapsulated into

Figure 5: Excluding Use Cases

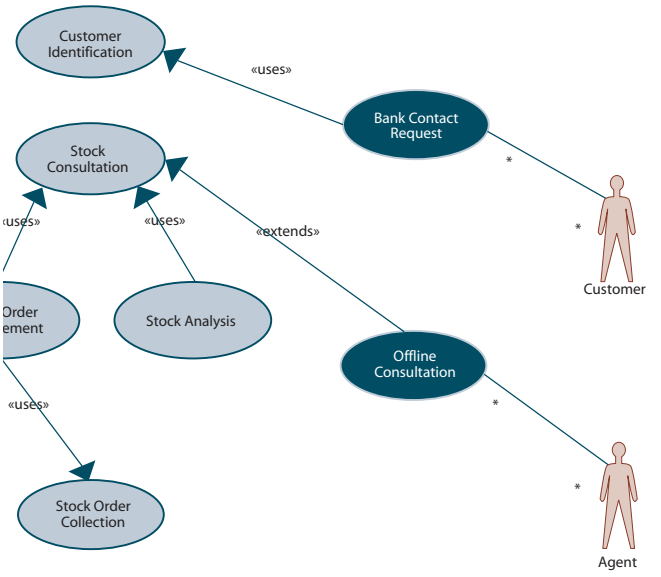
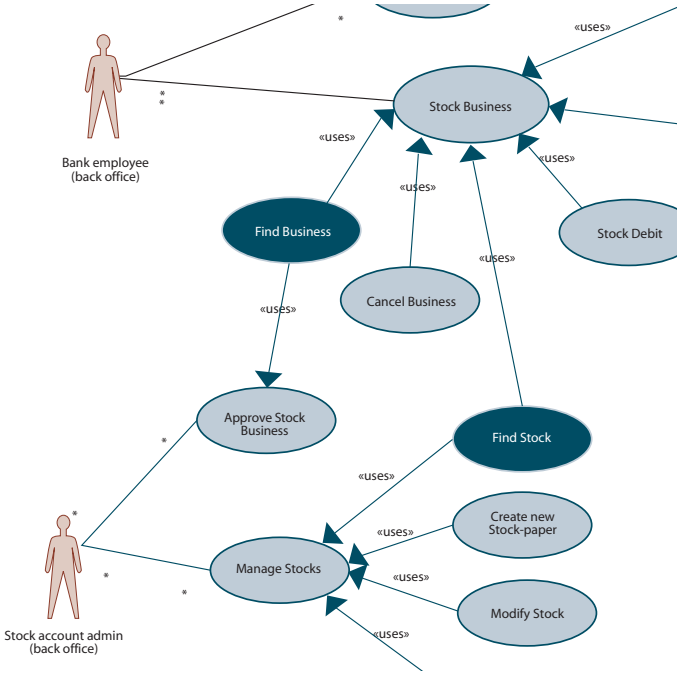


Figure 6: Sub-WorkItem or parent-WorkItem



a WorkItem. Therefore WorkItems are nothing but use case controllers implementing the user interface processes necessary for completing a use case (task) and putting all the required parts together for doing so. The use case diagram shown in Figure 4 was designed for a stock-management system that has been used for consulting with customers about their securities transactions (stock business) and fulfilling customer securities transaction requests.

The application implementing the use cases in Figure 4 supports front office employees for consulting customers and supports back office employees for completing the securities transactions. To achieve this, first you map one use case to one CAB-WorkItem. Relationships between use cases can be of two flavors: Either a use case is a sub-use case of another one or a use case is used by many other use cases other than its own parent. Of course, a use case that is really just a sub-use case not reused by others results in a sub-WorkItem. Pure sub-use cases are sub-WorkItems that are not accessible from outside their parents, whereas use cases used by many other use cases in addition to their own parents need to be accessible either directly or through their parent WorkItem.

Module Controller WorkItems

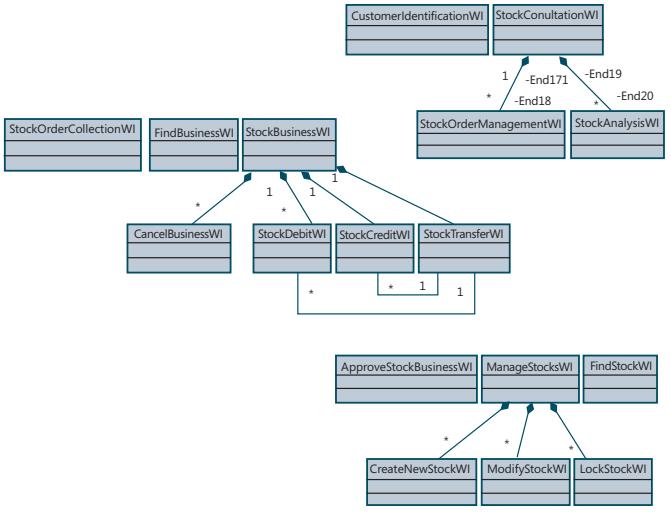
Parent use cases are the entry points for all their sub-WorkItems. Typically, a module has a root-parents WorkItem called Module Use Case Controller. This one is responsible for managing the state required for direct sub-WorkItems and providing the right context for these. Module Use Case Controllers typically add the services a module can offer to others, retrieve service-references to other services they require, register UIExtenionSites and commands for launching one of their sub-WorkItems, and launch their sub-WorkItems. Simple WorkItems without sub-WorkItems or sub-WorkItems themselves typically execute the same steps except that they should register services that are available within their hierarchy level, only. The Module-Use Case Controller WorkItems should be created whenever a module is loaded into the application.

Exclusion of Use Cases

When taking a closer look at the use case diagram in Figure 5, you will figure out that not all use cases will result in WorkItems. Take a look at the "Bank Contact Request" use case. Remember that you want to design a smart client used by bank employees only. A customer requesting a stock-transaction consultation (expressed through the "Bank Contact Request") can either go to the front desk directly or use an Internet-banking or net-banking solution (or something else). In any case, this use case is nothing that needs to be integrated into the smart client for the bank employees, just the opposite. "Customer Identification" needs to be part of the smart client you are designing now.

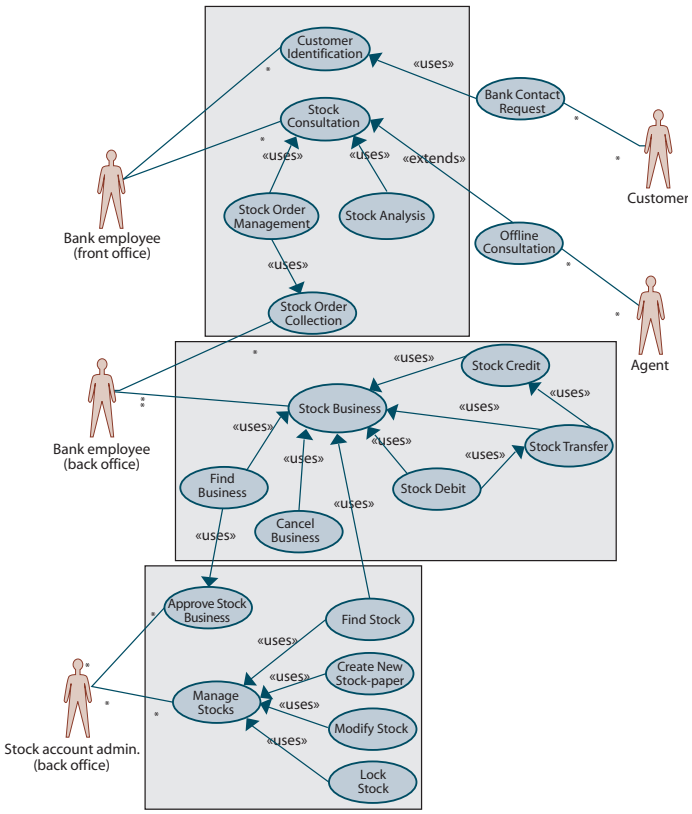
Another example is the "Offline Consultation" use case. What does it mean to your smart client? Is it really going to be a separate WorkItem?

Figure 7: Class Diagram showing the WorkItems for the previous use cases



Does it change anything in the business logic? No, therefore it won't be a separate WorkItem. But this use case is an indicator for something completely different; it's an indicator for the need of offline support. This is something you therefore need to verify against the infrastructure services identified earlier: Web service agents, for example, need to support connection detection, offline reference data stores, and update message queues.

Figure 8: Packaging WorkItems into modules



Sub-WorkItem or WorkItem

Some WorkItems will become parents as well, although they appear just as sub-WorkItems in the use case diagram. This typically happens when a use case logically belongs to a parent-use case, but in a detailed analysis you figure out that it does not share state, context, or anything else with its original parent or other sub-use cases and it does not depend on other commonalities either. In that case, to avoid unnecessary overhead, you should make them parent WorkItems as well. Take a close look at the use cases "Find Stock" and "Find Business" in the use case diagram of Figure 6. These use cases are just used for finding stock-papers or ordered business transactions. They do not depend on shared state or on context of their parent-WorkItem. So they are perfect candidates for becoming parent-WorkItems themselves called through commands registered by the module (more exactly, the Module Controller WorkItem) to which they belong.

WorkItem-Identification—Summary

Finally, after creating your use case diagrams, mapping use cases to WorkItems, and then excluding use cases and refactoring the hierarchy of your WorkItems, you will get a complete WorkItem object hierarchy for your composite smart client similar to Figure 7.

For the sake of performance and simplicity, I definitely recommend keeping the use case diagrams and therefore the WorkItem as simple as possible. The advantage you get with the approach described above is a clear, structured way for identifying WorkItems and especially identifying reusability-aspects of WorkItems.

Business Entity-Driven Strategy

A much simpler approach for smaller, simple applications is identifying and structuring WorkItems based on the business entities processed by the smart client application. You create a list of business entities processed by your application. Typical examples are Customer, Product, Stock, StockCredit, StockDebit, and StockTransfer. For each of these entities you create a WorkItem. As StockTransfer is a combination of both, it uses StockCredit and StockDebit as sub-WorkItems.

Packaging WorkItems into Modules

After you have identified your WorkItems, you need to package them into modules. A module is a unit of deployment for CAB-based smart clients. Basically, you package logically related WorkItems addressing the same business space into a module. Taking the original use case diagram from Figure 4, that would mean you create a module for stock-consultation, one for stock-business WorkItems, and a last one for stock-management, as shown in Figure 8.

But there are some additional criteria for deciding on the packaging of WorkItems into modules. These additional criteria are:

- Security
- Configurability
- Reusability

First and foremost, modules are configured in a profile-catalog containing a list of CAB modules that need to be loaded dynamically when your composite smart client starts. These modules can be configured based on a user's role-membership. Therefore, security plays a central role in deciding on how to package WorkItems into modules. Let's suppose a user is not allowed to manage any stocks, as shown in the use cases above. But definitely, a user not allowed to create new stock-papers or locking stock-papers will need to find these while completing the tasks. When configuring your stock-management module (which contains the "Find Stock" WorkItem according to the packaging in Figure 8) in a way that prevents bank employees from the back office from using it, they are also not allowed to use the "Find Stock" WorkItem. But as they need this WorkItem for other tasks, it is useful to package it into a separate module. If you want to reuse WorkItems independently from others, it makes sense to encapsulate them into separate modules. Likewise, if you need to configure WorkItems independently, also put them into separate modules.

In the example demonstrated in Figure 8, this is true for "Find Stock," "Find Business," and "Stock Order Collection" WorkItems. Therefore, it is useful to encapsulate them into separate modules, as shown in Figure 9.

If you figure out that configuration, security and reusability requirements are equal (or nearly the same) for some of the WorkItems outsourced into separate modules according to Figure 9, you can package them into one module instead of three modules, as well.

For example, if the security, configuration, and reusability requirements of all "Find X" WorkItems are equal, you can create one module containing all the "Find X" WorkItems.

Finally, it's important to note that each module will result in a separate .NET assembly. After you have created your shell and infrastructure services and have identified all WorkItems, including their packaging structure, your development teams can start with the development of the modules and their WorkItems.

Smart Client Software Factory

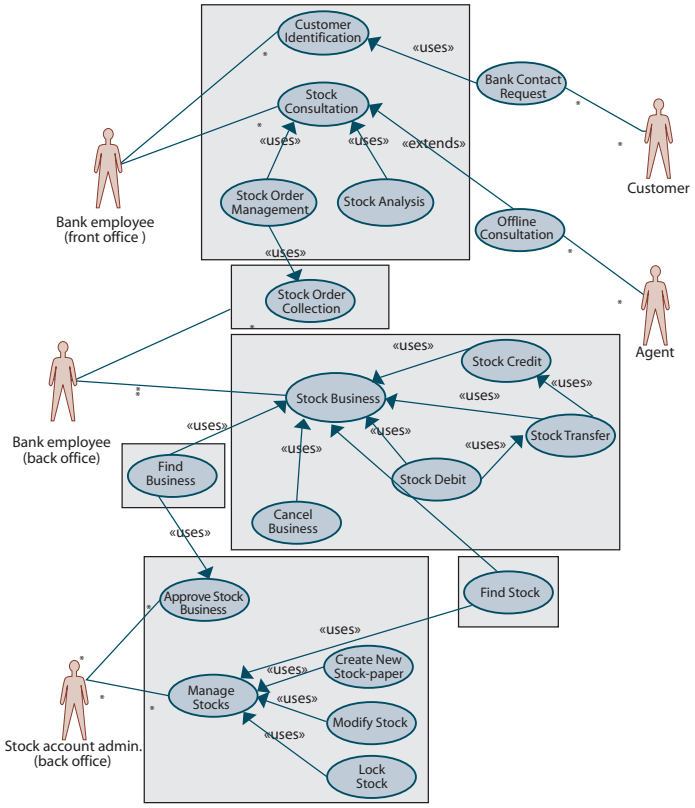
Although CAB provides a great infrastructure, the learning curve for some can be high. Furthermore, working with CAB requires developers completing many manual steps such as creating classes inherited from the WorkItem base class to create a use case controller or to create Controller-classes, View-classes, and Model-classes manually. In real world projects with large teams, this can result in many different practices for completing these steps, as every developer has separate preferences and working styles.

The Smart Client Software Factory (SCSF) is an extension to Visual Studio 2005 Professional (or higher) for automating and streamlining these tasks. The SCSF is based on the Guidance Automation Extensions also provided by the Microsoft Patterns & Practices team. The Guidance Automation Extensions are an infrastructure allowing architects and lead developers to easily create extensions in Visual Studio for automating typical development tasks with the primary target of enforcing and ensuring common directives and guidelines for their projects. SCSF provides such guidelines for CAB-based smart clients and automates things like creation of new modules, views based on the MVP-pattern, and event publications and subscriptions. For more information on SCSF and Guidance Automation Extensions refer to the following MSDN articles:

http://msdn.microsoft.com/vstudio/teamsystem/workshop/gat/default.aspx
http://msdn.microsoft.com/vstudio/teamsystem/workshop/gat/intro.aspx
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/scsflp.asp

The SCSF is a very useful tool that supports developers in creating CAB-based smart clients while adhering to architectural decisions—and also increasing developer productivity because of its integrated, and automated developer tasks in Visual Studio 2005.

Figure 9: New packaging according to security and reusability requirements



Final Words

With the Composite UI Application block and Smart Client Software Factory, the Microsoft Patterns & Practices team provides a powerful infrastructure for creating composite smart clients. The use case-driven analysis process for identifying WorkItems packaged into modules dynamically loaded by your shell enables a powerful way for identifying reusable, central functionality for larger composite smart clients. This avoids implementing things twice and therefore lowers costs for maintenance. While CAB provides all the necessary base classes within its framework, the Smart Client Software Factory adds the necessary guidance packages for automating certain developer tasks and providing guidance to developers. Together, they enable you to create rich composite smart clients in an agile and highly productive fashion.

RACON Software GmbH. is currently implementing the first large set of modules with more than 150 WorkItems, integrated into the common bank desktop framework. The first reviews of the bank desktop framework have shown that the approaches to design and infrastructure work well for making development as easy as possible, especially for line-of-business developers. The Shell-Extension service pattern used makes the integration of more complex views built with Windows Presentation Foundation (WPF) easier because the line-of-business developers do not need to worry about Windows Forms and WPF interoperability, since this functionality is encapsulated as a feature into the Shell-Extension service. Finally, the use case-driven approach offered an optimum solution for identifying WorkItems that are reused across different sets of modules. Overall, the CAB and SCSF took the development team forward in finding a pragmatic approach for creating the common, extensible bank desktop framework for RACON Software GmbH.

About the Author

Mario Szpuszta works in the Developer and Platform Group of Microsoft Austria and supports software architects of enterprise accounts in Austria. Mario always focused on working with Microsoft technologies and started working with the .NET Framework very early. In the past two years he focused on secure software development, ASP.NET Web development, and Web services, as well as integration of Microsoft Office in custom applications using Microsoft .NET based on Visual Studio Tools for Office, XML, and Smart Documents. Mario worked with Ingo Rammer as co-author on writing *Advanced .NET Remoting, 2nd Edition* and participated in writing *Pro ASP.NET 2.0 in C#* with Matthew MacDonald.

Quality Data Through Enterprise Information Architecture

by Semyon Axelrod

Summary

Data quality is a well-known problem and a very expensive one to fix. While it has been plaguing major U.S. corporations for quite some time, lately it is becoming increasingly painful. Higher prominence of various regulatory compliance acts, such as SOX, GLB, Basel II, HIPAA, HOEPA, and others, necessitates an adequate response to the problem. For a significant class of data issues (semantics), it is not possible to solve the "data quality problem" by just working with data. Unfortunately, this solution is not commonly understood at all, but the results leave very little doubt about the current (pure data) approach effectiveness. This discussion proposes how to deal with this problem through correct information architecture.

In the classes of applications that heavily depend on enterprise data quality—business intelligence, finance reporting, market-trend analysis, and so on—a typical approach to the data quality problem usually starts and ends with the activities scoped to the physical data storage layer (frequently relational databases).

Not surprisingly, according to the trade publications, most of these efforts have minimal success. Given that in business applications data always exists within the context of a business process, all the attempts to solve the "data quality problem" at the purely physical data level (that is, databases and extract transform load [ETL] tools) are doomed to fail.

This failure is because the physical level does not capture the requisite semantics to accurately communicate data across business processes. As a result, most of the semantic data issues exist at the process and organizational boundaries. The top (or enterprise) level is the focal point with the highest probability for discrepancy. Most companies do not have domain (or ontological) models. If models exist at all, the majority of them exist at the project (subdepartmental) level as logical data models. Enterprise-level business and information models are practically absent, and therefore there is no way to effectively communicate the data across organizational, department, project, or service boundaries.

Successful business data management begins by taking focus away from data. The focus initially should be on creating an enterprise architecture (EA), especially the commonly missing business and information architectures' constituents of it. Information architecture spans business and technology architectures, brings them together, keeps them together, and provides the necessary rich contextual environment to solve the ubiquitous data quality problem. Thus, enterprise business, information, and technology architectures are needed for successful data management.

Once we have agreed on the "common" aspects of the data model, we can extend it locally to suit our needs. This extension is analogous to using a base class in programming. Similarly, having an enterprisewide information architecture with rich semantic details lets us transform data at boundaries by mapping from one definition to another. These interfaces are analogous to the boundaries of services, with the definitions providing the contracts.

Data Quality

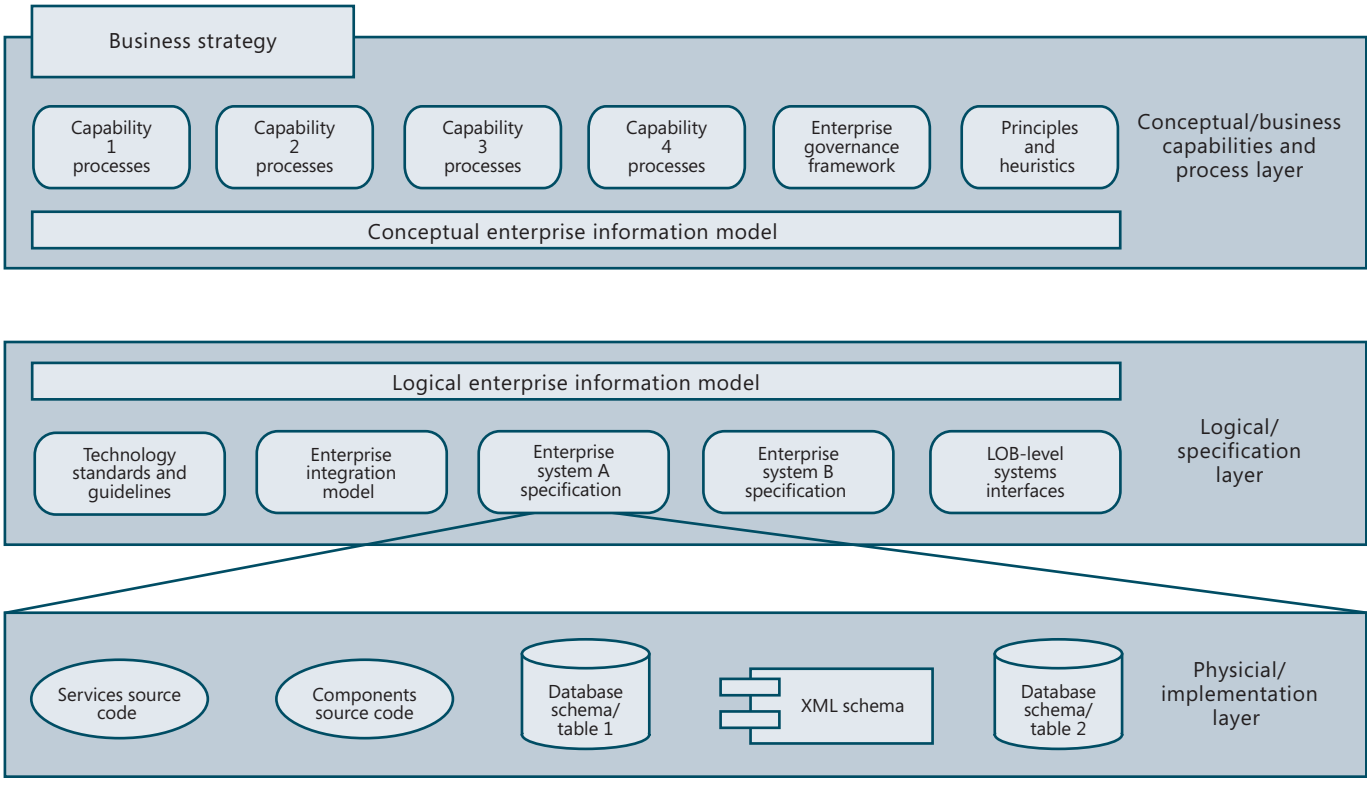
Major business initiatives in a broad spectrum of industries, private and public sectors alike, have been delayed and even cancelled, citing poor data quality as the main reason. The problem of poor information quality has become so severe that it has moved to the top tier among the reasons for business customers' dissatisfaction with their IT counterparts.

While it is hardly an argument that poor data quality is probably the most noticeable issue, in a vast majority of cases it will be accompanied by the equally poor quality of systems engineering in general, including requirements elicitation and management, application design, configuration management, change control, and overall project management. The popular belief that, "If we just get data under control, the rest (usability, scalability, maintainability, modifiability, and so forth) will follow," has proven to be consistently wrong. I have never seen a company where the data quality level was significantly different from the overall IT environment quality level. If business applications in general are failing to meet (even) realistic expectations of the business users, then data is just one of the reasons cited, albeit the most frequent one.

As a corollary, if business users are happy with the level of IT services, usually all the quality parameters of the IT organization effort are at a satisfactory level, including data. I challenge you to come up with an example where data quality in a company was significantly different from the rest of the information systems' quality parameters.

Therefore, what is commonly called the "poor data quality problem" should be more appropriately called the "data quality deficiency syndrome." It is indeed just a symptom of a larger

Figure 1 Enterprise architecture three-layered model



and more complex phenomenon that we can call "poor quality of systems engineering in general." Data quality is just the most tangible and obvious symptom that our business partners observe, not the root cause. (See Resources for an article in *The Architecture Journal 8* that provides an example of how an attempt to work with data architecture separately from all the other architectural issues leads to potentially significant problems.)

Since data plays such a prominent role in our discussion, let us first agree on what data is. Generally, most agree that data is a statement accepted at face value. In this article we are generally talking about the large class of data comprised of variable measurements, although the concepts apply to other types of data.

> **"WHAT IS COMMONLY CALLED THE 'POOR DATA QUALITY PROBLEM' SHOULD BE MORE APPROPRIATELY CALLED THE 'DATA QUALITY DEFICIENCY SYNDROME'"**

A notion that data is produced by measurements or observations is very significant. It points to a very important concept—a notion of data context or metadata—that is absolutely critical to the success of any data quality improvement effort. In other words, a number just by itself, stripped of its context, is not really meaningful for business users. For example, you have a *customer*. For the billing department, that customer is defined partially by one address, and for shipping it is likely defined by another. Without the context, you are not sure which locally unique aspects of the data definition apply.

If we know the context in which the customer interfaced with the enterprise, we now have a much better understanding of the business circumstances surrounding this number. This notion is at the crux of the poor data quality problem—lack of sufficient data context. We typically do not have enough supporting information to understand what a particular number (or a set of numbers) means, and we thus cannot make an accurate judgment about validity and applicability of the data.

IT consultant Ellen Friedman puts it this way: "Trying to understand the business domain by understanding individual data elements out of context is like trying to understand a community by reading the phone book."

The class of data that is the subject here always exists within a context of a business process. To solve the poor data quality problem, data context should always be well defined, well understood, and well managed by data producers and consumers. For example, from credit approval, legal approval, servicing approval, appraisal approval, custodial review, and investor review perspectives we have multiple subprocesses labeled "approval." This context means there must be an agreement on the information architecture that gives the *common* aspects of the data and some way to negotiate and share *local* enhancements.

Data Quality Attributes

According to Joseph M. Juran, a well-known authority in the quality control area and the author of the Pareto Principle (which is commonly referred to today as the "80–20 principle"), data are of high quality "if they are fit for their intended uses in operations, decision making, and planning. Alternatively, data are deemed of high quality if they

correctly represent the real-world construct to which they refer" (see Resources). Again, this definition points to the notion that data quality is dependent on our ability to understand data correctly and use it appropriately.

As an example, consider U.S. postal address data. Postal addresses are one of the very few data areas that have well-defined and universally accepted standards. Even though an address can be validated against commercially available data banks to ensure its validity, this validation is not enough for every purpose. In our previous example, if a business uses a shipping address for billing and vice versa, or uses a borrower's correspondence address for appraisal, the results obviously will be wrong.

To further quantify the problem, Professor Richard Wang of MIT defines 15 dimensions or categories of data quality problems. They are: accuracy, objectivity, believability, reputation, relevancy, value added, timeliness, completeness, amount of information, interpretability, ease of understanding, consistent representation, concise representation, access, and security. A serious discussion of this list would warrant a whole book. However, it is important to make a point that most of these attributes are in fact representing the notion of data context. For the purpose of our discussion on data quality, the most relevant attributes are: interpretability, ease of understanding, completeness, and timeliness.

The timeliness attribute, also known as the temporal aspect of information/data, is arguably the most intricate one from the data quality perspective. There are at least two interpretations of data timeliness. The first concerns our ability to present required data to a data consumer on time. It is a derivative of good requirements and design, but it is of little interest to us in the context of this discussion. The second aspect is the notion of data having a distinctive "time/event stamp" related to the business process, and thus allowing us to interpret data in conjunction with the appropriate business events. It is not hard to see that more than half of the data quality attributes in the previous list are at least associated with, if not derived from, this interpretation of timeliness.

While the temporal aspect of information quality is extremely important for understanding and communicating, it is often lost. For example, in the mortgage-backed securities arena, there are two very similar processes with almost identical associated data. The first is the asset accounting cycle, which starts at the end of the month for interest accrual due next period. The second is the cash-flow distribution cycle, which starts 15 days after the asset accounting cycle begins. This difference of 15 calendar days, during which many possible changes to status of a financial asset can take place, can make financial outcomes differ significantly; but from the pure data modeling perspective the database models in both cases are very similar or even identical. A data modeler who is not intimately familiar with the nuances of a business process, will not be able to discern the difference between the data associated with disparate processes by just analyzing the data in the database, which is why we need semantic definitions for reference.

Architecture

Architecture is one of the most used (and abused) terms in the areas of software and systems engineering. To get a good feel for the complexity of the systems architecture topic, it suffices to list some of the most commonly used architectural categories, methods, and models: enterprise, data, application, systems, infrastructure, Zachman, information, business, network, security, model-driven architecture (MDA), and certainly the latest silver bullet: service-oriented architecture (SOA).

All of these architecture types naturally have a whole body of theoretical and practical knowledge associated with them. Any in-depth discussion about all of the various architectural categories and approaches is clearly outside the scope of this article. Therefore, we will concentrate on the concept of enterprise architecture, and the following definition by Philippe Kruchten provides the context for this discussion: "Architecture encompasses the set of significant decisions about the system structure." Eberhardt Rechtin further clarifies: "A system is defined ... as a set of different elements so connected or related as to perform a unique function not performable by the elements alone" (see Resources).

To emphasize the practical side of architecture development, these two definitions can be further enriched. A long-time colleague of mine, Mike Regan, a systems architect with many successful system implementations under his belt, adds: "Architecture can be captured as a set of abstractions about the system that provides enough essential information to form the basis for communication, analysis, and decision making."

From these definitions, it is clear that system architecture is the fundamental organization of a system. System architecture contains definitions of the main system constituencies as well as the relationships among these constituencies. Naturally, the architecture of a complex system is very complex as well.

Architecture as Metadata Source

A rich contextual environment (metadata) needs to exist to solve the problem, and a comprehensive set of models is needed to produce this desired metadata. The enterprise architecture modeling effort can and should produce these models. There is an emerging industry consensus that enterprise architecture should include a business process model as well as enterprise information model, but as we have already stated previously, both (business and information) models are practically absent.

As previously discussed, conventional data modeling techniques do not contain a mechanism that can provide sufficiently rich metadata, which is absolutely necessary for any successful data quality improvement effort to be successful. At the same time, this rich contextual model is a natural byproduct of successful EA development process so long as this process adheres to a rigorous engineering approach.

To work with such architectural complexity, some decomposition method is needed. One such method is the three-layered model. All modern architectural approaches are centered on a concept of model layers. These are horizontally-oriented groups defined by a common relationship with other layers, usually their immediate neighbors above and below. A possible layering for EA can constitute a conceptual capabilities (or business process) layer at the top, the logical information technology specifications layer in the middle, and the physical information technology implementation layer on the bottom (see Resources). This model assumes an information systems-centered approach. In other words, the

purpose of this architectural model is to provide an approach to successful information systems implementation. A simplified three-layered model is shown in Figure 1. Some key concepts are worth mentioning:

- First, although business strategy is not a constituent of the business architecture layer, it represents a set of guidelines for enterprise actions regarding markets, products, business partners, and clients. The business architecture captures a more elaborate view of these actions.
- Second, the model demonstrates that enterprise information models reside in both the conceptual and in the logical layers, and provide the foundation for consistent interaction between these layers.
- Third, the enterprise IT governance framework is defined in the top conceptual layer, while IT standards and guidelines that support the governance framework are implemented in the specification layer.
- Finally, the enterprise specification layer defines only the enterprise integration model for the departmental systems but not their internal architectures.

Let us expand on the notion of business process architecture and elaborate on the layering details.

A Foundation for Data Quality Improvement

The EA framework is developed along two dimensions: horizontal and vertical. A complete traceability along both dimensions is absolutely necessary for the vertical (between the conceptual, logical, and physical layers) as well as horizontally within each layer but across the organizational boundaries.

Neither dimension of the modeling is new, but the two are not normally combined, and this approach leads to our semantic disconnect and thus to our poor data quality. I am trying to advocate a "hybrid approach." It is quite common that data warehouse sponsors are trying to advocate an enterprise data model. Operational departmental systems each have their own scope, but usually do not have any foundational business models and thus cannot possibly be mapped in a consistent manner to any common model, either horizontally or vertically.

As stated previously, all modern architectural approaches are centered on a concept of model layers. Let us expand on this notion specifically from the information quality viewpoint.

It is important to emphasize the business process layer as the foundation for our EA model. Carnegie Mellon University (CMU) Software Architecture Glossary provides this definition for EA: "A means for describing business structures and processes that connect business structures" (see Resources). Interestingly, this definition is applicable to and definitive of EA as a whole, and not just business EA.

The EA definition used by U.S. government agencies and departments emphasizes a strategic set of assets, which defines the business, as well as the information necessary to operate the business, and the technologies necessary to support the business operations. (See Resources for a link to the original version of this definition.)

This definition maps extremely well to the proposed three-layered view of EA, but a word of caution is appropriate: While the three-layered model provides a good first approximation of EA, it is by no means complete and/or rigorous. It is obvious that both business and technical constituencies of the EA model can and should be in turn decomposed into multiple sublayers.

In the quest to make the business enterprise architecture layer a robust practical concept, it has morphed from the initial organizational chart-centered (and thus brittle) view into a business process-centered orientation, and lately into a business capabilities-centered view, becoming even more resilient to business changes and transformations.

Current consensus around EA accentuates both the business and technical constituencies of it. This business and IT partnership is highlighted even more by the advances of the SOA, which views the business's process model and supporting technology model as an assembly of interconnected services.

In the proposed three-layered view of the EA, the business process layer includes a business domain class model. Since this model is implemented at the highest possible level of abstraction, it captures only foundational business entities and their relationships. Thus, the top layer domain model is very stable and is not subject to change unless the most essential underlying business structures change. The information (or data) elements that are defined at this level of the domain model are cross-referenced against the business process model residing in the same top layer. In other words, every domain model element has at least one business process definition that references it. The reverse is also true: There is no information element called out in the business processes definitions that does not exist in the domain model.

> "SUCCESSFUL BUSINESS DATA MANAGEMENT BEGINS BY TAKING FOCUS AWAY FROM DATA"

It is also worth pointing out that only common, enterprise-level, information processes and elements are captured at the top layer of EA. For example, because of historical reasons a particular enterprise consists of multiple lines of business (LOB), each carrying out its own unique business process with related information definitions. At the same time, all the LOBs participate in the common enterprise process. In this case, at the top (enterprise level), business process layer, only the common enterprise-level process will be modeled. In extreme cases, this enterprise process will consist primarily of the interfaces between the LOB-level business processes.

Each of the enterprise's LOBs will need to have its own three-layered model, where top-level business entities and the corresponding information (or data) elements will be mapped unambiguously to the enterprise-level model entities. Needless to say, only the elements that have their counterparts at the enterprise level can possibly be mapped. By relating LOB-level definitions to the common enterprise-level equivalents, we are eliminating semantic mismatch (ambiguity) between different business units. And since our data elements are cross-referenced with the business process models, we should have enough contextual information to correlate information elements at the enterprise and LOB levels. In the most

difficult cases, UML state transition diagrams should be created to capture temporal and event aspects of the business processes.

Support for Business Processes

The specification layer of the three-layered EA model introduces system-related considerations and defines specifications for the enterprise-level information systems. These are the systems that need to be constructed to support the business processes defined at the top layer of the model. By defining system requirements in terms of the business processes, another major cause of low data quality is eliminated: the disconnect between the business and the technology views of the enterprise system.

For example, it is quite common for more than one system to be operating on data elements from the objects defined at the top business layer. In this case, using my method, each system specification will define its own unique data attribute, but all these attributes are in turn mapped to the one element at the top layer. This top-down decomposition, with local augmentation, helps to alleviate a problem known as the "departmental information silo."

Again, similar to the top layer, in the spirit of correlating data with the process contextual information, business use cases realizations and system use cases (or similar artifacts) are introduced at this level to provide enough grounding for the data definitions. It is important to note that in addition to the enterprise systems, the system interfaces of LOB-level systems (to support business process connection points among the different LOBs) are also specified in this layer.

In this layer, the platform-specific implementations are defined and implemented. Multiple platform-specific implementations may be mapped to the same element defined at the specification layer. This unambiguous, contextually-based mapping from possibly multiple technology-specific implementations to a data element defined at the technology-independent specification level is the foundation for the robust, high-quality, data management approach.

> "THE BUSINESS AND IT PARTNERSHIP IS HIGHLIGHTED EVEN MORE BY THE ADVANCES OF THE SOA, WHICH VIEWS THE BUSINESS'S PROCESS MODEL AND SUPPORTING TECHNOLOGY MODEL AS AN ASSEMBLY OF INTERCONNECTED SERVICES"

Solving the data quality problem is really a matter of solving the information architecture problem. By having enterprise-level definitions of our common data attributes and incorporating ways to communicate local extensions, we can accommodate all required mappings in the domain and support all of our service-level agreements.

It is impossible to overestimate the importance of the two-dimensional traceability in the discussed architectural model. The first dimension—vertical traceability between the model layers—provides a foundation for rich contextual connection between the business process and the system implementation that supports this process. The second dimension—horizontal traceability within the same model layer—provides a foundation for a rich contextual connection between the hierarchical organizational units, as well as the systems implemented at their respective levels.

A robust traceability mechanism is absolutely necessary for high-quality data to become a reality. The architectural model provides a foundation for the information traceability and thus data quality, without which it is not possible to address a cluster of issues introduced by the modern business environment in general and especially by the legal and regulatory compliance concerns.

Resources

"An Ontology of Architectural Design Decisions in Software-Intensive Systems," Philippe Kruchten (University of British Columbia 2004)
www.kruchten.com/inside/citations/Kruchten2004_DesignDecisions.pdf

Analysis Patterns: Reusable Object Models (Object-Oriented Software Engineering Series), Martin Fowler (Addison-Wesley Professional 1996)
www.martinfowler.com/books.html#ap

Carnegie Mellon Software Engineering Institute
Software Architecture Glossary
www.sei.cmu.edu/architecture/glossary.html

"Data Replication as an Enterprise SOA Antipattern," Tom Fuller and Shawn Morgan, *The Architecture Journal* Journal 8 (Microsoft Corporation 2006)
www.architecturejournal.net/2006/issue8/F4_Data/

Juran's Quality Handbook, Fifth Edition, Joseph M. Juran and A. Blanton Godfrey (McGraw-Hill 1999)

OMG
OMG Model Driven Architecture (MDA)
www.omg.org/mda/

Unified Modeling Language (UML) Resource Page
www.uml.org

Systems Architecting: Creating and Building Complex Systems, Eberhardt Rechtin (Prentice-Hall 1990)

About the Author

Semyon Axelrod's information technology career spans three continents and more than 25 years of experience in various areas of software engineering, management, and information systems. He currently specializes in enterprise architecture, IT governance and leadership, strategic business and IT alignment, and legal and regulatory compliance issues. His articles can be found in *DM Review* and *ComputerWorld* magazines. Contact Semyon at semyonaxelrod@yahoo.com or his blog adsys.blogspot.com.

Business Improvement Through Better Architected Software

By Sten and Per Sundblad

Summary

Business software is often challenged. According to studies performed by companies such as the Gartner Group, the Standish Group, and IDC, an astonishingly large portion of development projects fail to come up with anything useful at all. An even larger portion is challenged for reasons such as not supporting the business as well as the business needs to be supported. When businesses need to change fast to keep up with customer needs and with competition, software is often mentioned as the show stopper; the software used is often too complex and too unrelated to the business to allow the business to change as fast as necessary.

This article suggests that software architects must improve their business understanding to help solve this problem. But business people must also learn how to better communicate to software development teams what they expect from them. New architect roles, both on the business side and on the IT side, are needed for the creation of IT systems that solve business problems well, and that also help the business to be agile, able to change at least as fast as its main competitors.

Business software exists for one reason only: to support the business and its activities, or to help change the way business is performed. There's no other reason for business software. If it doesn't support the business the way the business needs to be supported or help change the business, then it doesn't matter how technically brilliant the software is. Its value lies in its capability to increase the productivity and efficiency of the business.

In principle, there are three ways in which business software can support a business and its activities:

1. A business process improvement project is started to improve the functionality of a specific business process. New or improved software is designed to help improve the way people can perform the business process.
2. New ways to use technologies become available, making it possible to totally change the way business is performed. For example, Web services have profoundly changed the way a Web shop might interact with suppliers of goods sold by the Web shop. "Web 2.0" might have a similar effect on the way some businesses organize their interaction with customers and partners.
3. Decision support

As members of the huge software development community, we sometimes forget the real purpose of business software. We tend to admire IT solutions for their technical excellence rather than for the way they support the business. It's sometimes taken for granted that the software does its job.

What Business Managers Say

A fairly recent study about business managers' top priorities for future IT investments, made by IDC , indicates that it's not always so. The survey was reported in *Computer Sweden*—the leading Swedish newspaper dedicated to IT—early in 2005. The article was written by IDC Nordic employee Per Andersen.

According to the survey, the highest priority (36.2%) was for "programs and services to be better integrated with business processes." The second highest priority (35.2%) was for "better access to relevant information." Priorities such as "lower system cost" and "improved IT security" wound up lower on the list, with 26.1 percent and 23.4 percent, respectively.

This indicates that we in IT don't succeed particularly well in meeting business managers' needs. If we did, lower system cost and improved IT security would be a higher priority for business managers than better integration with business processes and better access to relevant information.

A Gap to Bridge

The figures also indicate that there's a gap between what concerns business managers and IT people. Business managers want effective business processes to help them achieve high-order business goals. They want access to information that helps them know whether the processes are as effective as they should be, and whether they are on the way toward achieving their business goals. Apparently, as seen from the IDC survey, these seem not to be equally high on the IT person's priority list.

The Agile Business Thrives

For business managers it's important that the structure of software solutions is based on the strategy and structure of the business. Otherwise, the business can't change swiftly enough when new

business conditions require change. Being able to change quickly is an important characteristic for any modern business.

In contrast, if the business software is structured the same way as the business is, it becomes easier and less risky to change the software concurrent with the changes in business. When this is the case, the business becomes agile. The agile business always beats the less-agile competitor in the long run.

However, if the business itself isn't well structured, or if the structure isn't well known and understood, it's impossible to create a software structure based on the structure of the business. And if the software architect is much more interested and competent in technical issues than in business matters, which is often the case, it becomes really difficult to design software that is well aligned with the business.

Software Architects Need Business Savvy

Whenever you meet someone who is thought of as a software architect, chances are that this person is highly competent in technical IT matters—especially those that concern software—and has a high level of interest in these matters. It's less common that such a person is equally competent in business matters.

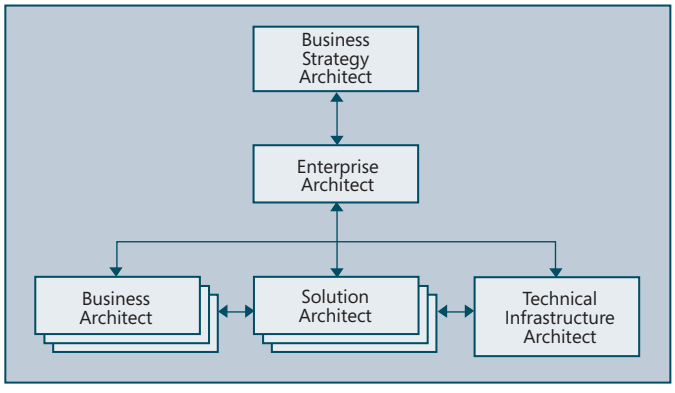
This might be a problem! For someone to be able to create a structure that supports the business well, that someone must first understand the business, its structure, its goals, and its problems. How else would it be possible to find the "right" technical solution?

We firmly believe that this should be one of a software architect's most important qualifications: to understand business issues well enough to be able to define IT solutions that help solve business problems, help reach business goals, and are structured in compliance with the structure of the business.

Businesspeople Don't Understand IT Well Enough

Now let's assume that the IT community sometime in the future arrives at a point where software architects indeed do understand business well enough to design effective business software. Would we, at that time, have solved the problem of software not quite being based on the way business is performed, not quite helping solve business problems, and not quite being able to help reach business goals? Not necessarily. There's another side of the coin, too. Businesspeople don't understand IT well enough to specify their requirements for business software, or to understand how innovative software could change the basics of their businesses.

Figure 1 - Architect Roles in Business Software Development - Modified



This leads to an interesting question: Could it be that we need other kinds of architect roles to produce business software than those we typically think of today? And could it be that these roles differ more from each other than most of us think? We suggest that this is the case.

Five Architect Roles

We believe that Figure 1 gives a good view of the architect roles needed for the development of good business software. The rest of this article explains the responsibilities of each role in some detail.

Roles—not People

Remember that each rectangle in Figure 1 represents a role people can play, rather than a specific person. In reality, some roles will be played by a single person, while others will be played by multiple persons. More importantly, some persons will play multiple roles in relation to Figure 1, as well as in relation to roles not present in Figure 1.

Business Strategy Architect

We have never before thought of this role as an architect role, even if over the years we have talked a lot about the importance for software architecture of its outputs. It was John deVadoss, director of architecture strategy at Microsoft, who told us that he tends to talk about a strategic architect role, the purpose of which is to change business focus and define the enterprise's to-be status. This role, he says, is about the long view and about forecasting.

Notice, though, that this role is about business strategy rather than IT strategy. The strategic architect works closely with top management, which is the body responsible for strategic decisions. He or she might even be part of top management.

Years ago Peter Drucker told us all that every organization—be it a commercial business or a public organization—has a purpose and a mission. The purpose describes why the organization has been started and why it still exists. In the case of a commercial business, Drucker says, the purpose is always to create economic values for its owners. The mission describes what the organization is supposed to do for whom; it should be expressed in terms of customer values rather than in concrete products to provide.

Future Product and Market Scope

A strategic plan should tell a business what markets it should get into and what kinds of products it should offer those markets. Such a specification of the future product and market scope then drives other strategic decisions. For example, it drives decisions about which key capabilities are required to support the future product and market scope, decisions about size/growth and return/profit guidelines, business unit missions, business processes and activities needed, and more. The strategic plan should also tell what the business must do to get out of some activities to make room for the new, and when all that work must take place.

Based on the strategic plan, key people and resources must be allocated to do the necessary work, and goals must be set to direct that work. Every business process and every business activity must have a purpose, and that purpose must be to produce business values, contributing to the value streams.

To define, negotiate, and document all of this is the business strategy architect's job. It's also his or her job to communicate it to top management for approval.

Business Architect

As already noted, the mission of business architects is to improve the functionality of the business. Their job isn't to architect software but to architect the business itself and the way it is run. However, the structures they provide should be used by software architects as a foundation for the design of business software solutions.

Definitions

The Business Architect Association defines the term business architect as follows:

A Business Architect is anyone who takes a step back, looks at the way work is being directed and accomplished, and identifies, designs and oversees the implementation of improvements that are harmonious with the nature and strategy of the organization.

The Open Process Framework defines business architect as "the specialized architect role played when a person develops the architecture for a customer organization's business enterprise." The framework also mentions a number of areas in which a business architect needs expertise. Among them you'll find such areas as:

- Deep knowledge of business and strategic marketing
- Strong in strategic and analytical thinking
- Excellent communication and presentation skills
- Solid experience in areas such as business, marketing, and business process reengineering

Notice that no IT-related areas of expertise are mentioned. This strengthens the argument about business architecture being primarily about business improvement and only secondarily about creating a foundation for software development.

Existing Business Architecture Methodologies

Formal methodologies for business architecture exist. If you're interested in finding out which some of them are, you could take a look at an extended version of this article, published on http://www.2xsundblad.com/en/articles/business_improvement_through_better_architected_software

Multiple Levels of Abstraction

Business architecture is work that spans multiple levels of abstraction. It's not enough to map the activities of business processes; the external profile of each process and activity must also be analyzed and documented. The business architect must understand how even low-level activities contribute to the business's value streams, and he or she must be able to evaluate each activity in terms of how important it is to the organization's results, how well it needs to perform versus how well it actually does perform, and whether it helps differentiate the enterprise from its competitors.

Software Architecture in General

When moving the focus from business architecture to software architecture, it's easy to see how different they are. Business architecture is about architecting the business; software architecture is about architecting software meant to support, automate, or even totally change the business and the business architecture.

They have much to do with one another. Software architecture should be based on business architecture. How else could IT systems effectively support the business? Inversely, business architecture should be designed to take as much and as effective advantage of information technology as possible. The two go hand in hand, but they are different.

In our framework—see Figure 1—we have included one software architect role only. It's the solution architect role.

Solution Architect

Solution architect is a relatively new term, and it should refer to an equally new concept as well. Sometimes it doesn't, though; it tends to be used as a synonym for application architect.

Application Architecture

If we talk about the service-oriented world as the new world, and everything that came before SOA as the old world, it could be argued that the concept of "an end-to-end application" belongs to the old world. We believe that the most common conception of an application is a computer program that contains everything needed to help solve a business problem. User interface, business logic, and data access logic are typically included in an application. Sometimes the database is also included.

In an application-centric world, each application is created to solve a specific business problem, or a specific set of business problems. All parts of the application are tightly knit together, each integral part being owned by the application. An application architect designs the structure of the entire application, a job that's rather technical in nature. Typically, the application architect doesn't create, or even help create, the application requirements; other people, often called business analysts, less technically and more business-oriented than the typical application architect, do that.

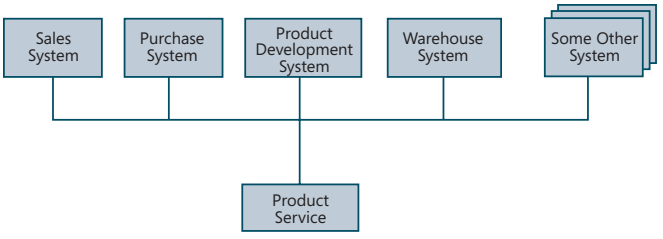
Service-Oriented Solutions Are Different

In SOA, it's different. IT solutions to business problems tend to be solved by the use of multiple autonomous services, where none of the services is an integral part of or owned by the solution. Instead, the services are only used by the solution, just as they might be used by other solutions.

For example, as seen in Figure 2, a product service might be used by a sales system, by a purchase system, by a product-development system, by a warehouse system, and probably by other systems as well. The service will not be owned by any of these systems; it will be a separate resource used by each of them. It will contain some functionality that is developed for the sales system, some that's for the purchase system, some for the warehouse system, and perhaps some for general use.

One of the four tenets of service orientation is for services to have

Figure 2 - Reuse of Product Service



clear and respected borders. The internals of a service should be used by that service only. According to this tenet, the only way for a service consumer to enjoy results produced by the service is to send a message to it, asking it to perform some operation. There's no way the consumer will be let inside the service's outer border.

It's obvious then that the service must expose the functionality it offers through service interfaces. The typical service interface today seems to be a Web service. Consider a service such as the product service mentioned above. It will serve different systems with different needs. Doesn't it seem reasonable that each need will be taken care of by a separate service interface, as in Figure 3? One for the needs of the sales system, another for the needs of the purchase system, and so on.

It follows that consumers of such a service can experience the service only through its service interface. In effect, for a consumer, the service interface exposed to that consumer is the service.

Business-Driven Solution Architecture

For a service to be successful, it must meet its consumers' needs. From this it follows that consumption needs should drive the specification of the service's interface (or interfaces). For each detail of the service's set of interfaces, an identifiable business need should exist.

It follows from the job title that a solution architect should architect technically oriented solutions to business problems. If this means—in a service-oriented environment—to design service interfaces, then it's interesting to note that the job isn't terribly technical. It requires technical understanding, but to an equally high degree it requires understanding of business issues and business needs. This is probably not a profile that very well matches the profile of most of today's application architects.

Context-Aware Solution Architecture: Creating the Physical Order of a Business

Christopher Alexander, the man behind the design pattern movement, says another interesting thing about architecture in general: "The activity we call building creates the physical order of the world, constantly, unendingly, day after day … Our world is dominated by the order we create." As a consequence, whatever we build is always part of a greater whole, and as architects we can't just focus on what we build but also on how well it harmonizes with its present, or at least its future, environment. Alexander is a "traditional" architect; his environment consists of "towns, buildings, and constructions," but what he says is equally valid for the architecture and building of business systems.

This quality of a design fitting well with its present and future environments is crucial for service-oriented architectures. It's our absolute conviction that the most important purpose for service-oriented architectures is the business agility they're capable of bringing to the enterprise. And being agile is a required—even if hard to achieve—attribute of any modern business. It's much more important for a business as such to be agile than for the software it uses to be produced by agile development methods. There's not necessarily any conflict between the two, but business agility is by far the most important one.

In the last part of this article we'll talk about the enterprise architect and his or her vision of the electronic or serviced enterprise. Such a vision is realized step by step over many years by architecting and implementing increasingly effective business processes and effective technical solutions to support them. If the solution, business, and enterprise architects work well together, they can make sure that every single solution fits well into the enterprise architect's vision of the enterprise's future business process and system portfolio.

Thus, each new or improved business process, and each new or improved IT solution, becomes an important part of the greater whole. This is the way the physical order of the enterprise is constantly and coherently re-created and improved.

If, in contrast, the solution architect is allowed to—or even forced to—suboptimize solutions, the business will never be agile. It won't be able to change with the same speed as its more-agile competitors.

One might ask how a solution architect can be forced to suboptimize a solution. Unfortunately, this happens all the time. Building the serviced enterprise is a process that takes time and money, and every project involved becomes more expensive and takes more time to plan and implement because of that. This is especially true for the first few projects during that process. Extra money and extra time must be allocated to every project to avoid suboptimization. When no extras are allocated, the solution architect is in effect forced to suboptimize the solution. It might seem hard to swallow, but in the long run suboptimization costs much more than the extra cost needed to avoid it.

An Infrastructure of Information

Notice that we're not advocating top-down architecture or top-down development of either business nor business software. To the contrary, what we do recommend is a top-down effort to make sure that business and software architecture support business strategy and strategic plans, and also to make plans for an infrastructure of at least such information that should be shared within the enterprise across organizational borders.

Such an infrastructure could be compared to the infrastructure of electricity and the infrastructure of water, which are available to make life easier for the builder of houses in a typical city. To be shareable across organizational and functional borders, the infrastructure must be planned in a top-down fashion. But design, implementation, deployment, and enhancements over time will all be performed bottom-up.

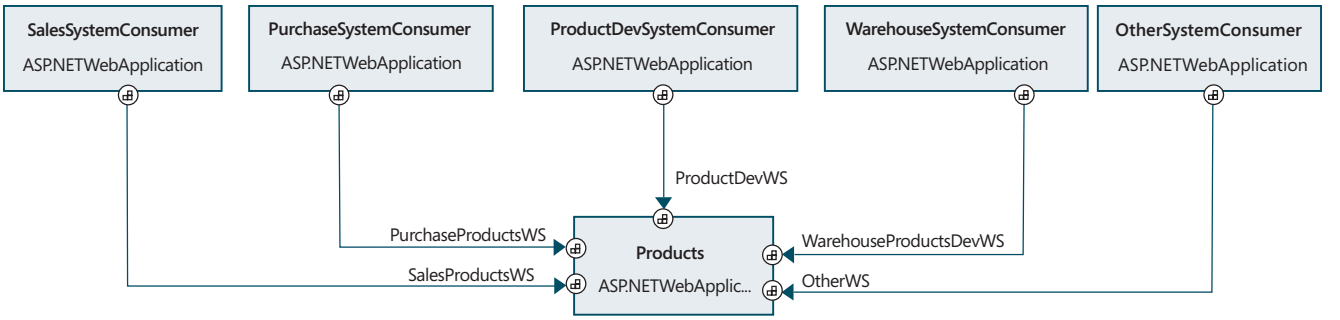
You can envision the whole from a top-down perspective, but it will grow from its parts! The "whole" grows organically from the needs of real people rather than from a grand master plan, even though it's conceptualized from a top-down perspective. To paraphrase Alexander, each business process we design or modify, each service or user application we define, develop, deploy, modify, and enhance will all help create the physical order of our business, constantly, unendingly, day after day.

Technical Infrastructure Architect

This role isn't directly involved in software development, so we're not talking much about it in this article. The technical infrastructure exists for the solution architect to deploy his or her solutions, which means that the solution architect and the technical infrastructure architect should work together to ensure safe and productive deployment and operation of the system.

Notice that our name for the role—just slightly different from

Figure 3 - Separate Web service interface for each conversation



what's common—emphasizes that it's about technical infrastructure. This involves things such as hardware, network, operating system, and system software. The technical infrastructure does not involve any business software. Having said that, it could involve the so-called infrastructure services, such as a security service, a logging service, or an error-management service.

Enterprise Architect

We'll discuss the enterprise architect role last. This is for a good reason. The enterprise architect role collects all of the other architect roles—with the business strategy architect role as the only exception—within it. You could argue that enterprise architecture consists of business architecture, solution architecture, and technical infrastructure architecture. You could also argue that additional architecture roles could be included; security architecture and organization architecture are two good examples.

It all depends on how you define enterprise architecture. Let's see how some sources define it and then talk more about the role and its responsibilities as we see them.

Definitions

Wikipedia defines enterprise architecture as follows:

Enterprise Architecture is the practice of applying a comprehensive and rigorous method for describing a current and/or future structure and behavior for an organization's processes, information systems, personnel and organizational subunits, so that they align with the organization's core goals and strategic direction. Although often associated strictly with information technology, it relates more broadly to the practice of business optimization in that it addresses business architecture, performance management, and process architecture as well.

The U.S. Department of Health and Human Services, in its Centers for Medicare & Medicaid Services, refers to the Clinger-Cohen Act, which requires that every U.S. federal agency develop an enterprise architecture. Enterprise architecture is defined as "a management engineering discipline presenting a comprehensive view of the enterprise, including strategic planning, organizational development, relationship management, business process improvement, information and knowledge management, and operations." Enterprise architecture is also described as consisting of "models, diagrams, tables, and narrative, which together translate the complexities of the agency into simplified yet meaningful representations of how the agency operates (and intends to operate)."

What Do We Need Enterprise Architects for?

The enterprise architect should concentrate on the big picture rather than on low-level issues. For example, he or she should know that there is a purchasing business process, what this process is for, and how it relates to business's value streams, but he or she doesn't necessarily have to know which business activities that process consists of.

The enterprise architect should also know about the possible existence of a purchasing process service supporting the purchasing process. Even more important, he or she should know about the existence of all entity services, and all business process services, and also what the mission and responsibilities are for each service. But the architect shouldn't be too much concerned about what interfaces each service exposes, or about its internal contents or structure. These details should be left to the solution architect and software engineers, and to the owner of the service.

Many large companies do employ enterprise architects. Some are called enterprise architects, while others have different titles, such as chief information officer (CIO). In the latter cases, the CIO takes on enterprise architect tasks in addition to other CIO tasks.

As you can see from Figure 1, the enterprise architect's place is between the business strategy architect and the other architects. He or she is controlled and led by business strategy, and should control and lead business, solution, and technical infrastructure architecture efforts. And he or she should own and care for the "city plan," which is a large-scale map of the business architecture and the IT resources used to support the business. Chiefly, such a map would consist of overview models, lists of items, tables, and narratives. Here are a few examples of lists and tables that might be included in a city plan:

- List of business processes
- List of strategic information areas
- Cross-referencing table of interactions between business processes and information areas
- List of available applications and services
- Cross-references of applications and services to business processes and information areas
- Cross-references between applications and services

Each list and table item should be populated with or supported by quality information. For example, how well do we need this application, service, or business process to perform, and how well does it actually perform? An item with mediocre performance is a problem if we need it

to perform well or excellently, but if mediocre is good enough, its actual performance is not a problem.

Business Improvement and Software Development Efforts

In principle, all development of business software should be based on the enterprise architect's strategic road map leading toward a future desired state. It's the enterprise architect's job to know where development efforts can yield the highest payoff for the efforts needed. It's also the enterprise architect's job to know where the fruit is hanging low and will be easy to pick.

Notice that new business software should never be the goal for business improvement efforts. Instead, the goal should always be improved business value streams in terms of things such as higher value for internal or external customers, lower costs, or faster throughput. The business software is always just a means to that end.

Therefore, business software development should always start with a detailed business analysis performed by a business architect. Some combination of the enterprise architect, the strategy architect, and top management sets the scope for such analysis, typically tied to a business process.

The business architect is constrained by this scope and by an overview model of business processes that has been created from a top-down perspective. Within these constraints, he or she performs a detailed analysis of the business process at hand, from a bottom-up perspective. Together with a solution architect, he or she designs a model of the future state of that part of the business. After approval, this model is handed over to the solution architect for design of a technical solution.

Notice again how top-down and bottom-up perspectives go hand in hand to create the best results in a minimum amount of time. Also notice how this process highlights the position of the enterprise architect between the business strategy architect and the business and solution architects.

This position, and the fact that the enterprise architect owns all the business and technology models, makes it possible for the enterprise architect to give strategic directions, as well as business and technology context, to each business improvement and software development effort.

Conclusions

Business managers want IT systems that are better integrated with their business processes and that give them better access to relevant business information than the IT systems they have traditionally been given. Managers set higher priorities for these qualities than for lower system cost or improved IT security. This should tell us that IT doesn't satisfy business managers the way business managers need to be satisfied. It should also tell us that there are great opportunities for increased business agility, increased competitiveness, and improved business functionality if we could solve this problem and satisfy business managers better.

The main reason for these difficulties seems to be the gap in knowledge, understanding, and interest between businesspeople and IT people. This gap results in too-large differences between the architecture of the business and the architecture of its software. When these architectures are not well aligned, it's difficult to make the software revisions needed to change and improve the business processes.

If an enterprise could bridge that gap, or even close it, and if it could better align its software architecture with its business architecture, that enterprise would become more agile; it would be better able to change when external pressure makes change desirable for the business. Today more than ever, an agile business thrives and is better able to compete in the marketplace than a less-agile business.

Business architecture should be established and managed by a new breed of business architects rather than by the old business analysts. Software architecture should be established and managed by solution architects with a more business-oriented and a less-technical background than yesterday's application architects. Business and solution architects should work together to design and suggest improved business processes that take better advantage of technical opportunities than the original ones did.

Extended version of this article

If you enjoyed reading this article, you might also enjoy reading the extended version, which we have published on http://www.2xsundblad.com/en/articles/business_improvement_through_better_architected_software. It's more than twice the size of this one, and it covers some other interesting aspects of development roles. For example, it suggests a very clear border between solution architecture and software engineering.

References

IDC presents itself as "the premier global provider of market intelligence, advisory services, and events for the information technology, telecommunications, and consumer technology markets". You'll find it at http://idc.com.

For example, Peter Drucker's Management: Tasks, Responsibilities, Practices. Collins (reprint edition April 1993). ISBN 0887306152. Several other Drucker titles exist, all of them very interesting. Five people reviewed this title on Amazon.com; all five gave five stars to the book. You may also want to check out Christopher Alexander's The Phenomenon of Life: The Nature of Order, Book 1 (Prologue). Center for Environmental Structure. ISBN 0972652914. The following web sites are also good references:

http://www.businessarchitects.org/index.php?page=what_is_baa

http://www.opfro.org/index.html?Components/Producers/Roles/BusinessArchitect.html

http://en.wikipedia.org/wiki/Enterprise_architecture

http://www.cms.hhs.gov/EnterpriseArchitecture/

http://www.ed.gov/policy/gen/leg/cca.html

About the Authors

Sten Sundblad is co-founder and chief solution architect of Sundblad & Sundblad, a company he runs with his son, Per Sundblad. The main business of Sundblad & Sundblad is to develop and use training material to train software architects. Sten is a frequent speaker on architect subjects. Sten is also a Microsoft regional director and was selected in 1998 as the first non-U.S. Microsoft Regional Director of the Year. He was named a Microsoft MVP Solutions Architect in 2005 and 2006. Sten lives in Uppsala, Sweden.

Per Sundblad is co-founder and managing director as well as solution architect and software engineer of Sundblad & Sundblad, a company he runs with his father, Sten Sundblad. The main business of Sundblad & Sundblad is to develop and use training material to train software architects. Per has also been a Microsoft regional director since 1997. He was named a Microsoft MVP Solutions Architect in 2005 and 2006. Per lives in Uppsala, Sweden.

ARC

Microsoft®

THE ARCHITECTURE JOURNAL™
Input for Better Outcomes