A PRAGMATIC APPROACH TO IMPLEMENTING A SERVICE-ORIENTED ARCHITECTURE WITH SUN JAVA™ COMPOSITE APPLICATION PLATFORM SUITE

White Paper
February 2007

Table of Contents

Chapter 1

Introduction

The term “service-oriented architecture” has been used quite frequently in the news and by many information technology (IT) vendors recently. While a large number of service-oriented architecture (SOA) projects have been undertaken in the last few years, not all have achieved the benefits expected. Often, an SOA project has been turned into an IT-only initiative that just delivers more IT-centric solutions. The result of an IT-led SOA implementation can easily be a mass of complex technical services that are very loosely aligned to the goals of the business and may provide very few reusable business services. These resulting technical services usually require complex point-to-point interconnections that only the most skilled IT staff are likely to understand.

In some cases, well-meaning project teams have had to drop their plans to adopt an SOA because the business was demanding fast project delivery of a tactical solution. This means that team members have little or no time to get educated on the new development approach required in order to benefit from an SOA, and SOA investments are then delayed until a later project. The key challenge is breaking this cycle. How can you implement your next project using SOA-based principles without the burden of implementing a completely new architecture from scratch?

Incremental delivery is a key goal for any SOA project, but it is important to keep sight of the business goals of the SOA as well. Without a clear understanding of what’s critical, what has to be done, and when, real SOA benefits may never be reached.

The main focus of this white paper is to provide practical, real-world guidance on how to start your first SOA project and how the Sun Java™ Composite Application Platform Suite (Java CAPS) can help you successfully implement SOA-based solutions. This paper describes a pragmatic approach for SOA development that adheres to the principles of SOA. It will also outline a four-layer model for SOA implementation that will:

- Help IT and business cooperate in project delivery
- Show how to provide real business solutions in practical time frames using Java CAPS
- Deliver business solutions on a project-by-project basis

Not all aspects of an SOA implementation will be discussed; the main focus for this document is to describe the steps involved in taking business processes from design through to the physical implementation using a pragmatic approach and the Java CAPS platform.

Chapter 2

Defining SOA Projects

First, what exactly is an SOA and how is it different from traditional software development approaches? SOA can be defined at the high level as:

An architecture where services are defined and orchestrated using open standards, allowing for a pluggable service infrastructure that removes single vendor lock-in and provides an agile infrastructure where services range from business definition through to technical implementation.

It is important to not confuse your existing IT infrastructure and approach in SOA terms. After all, you could argue that everything developed in your IT department for the past 20 to 30 years is service-based, and therefore is an SOA. You could also argue that Customer Information Control Systems (CICS), Corba, Enterprise JavaBeans™ (EJB™) technology, object orientation, and Web services are all SOA, but that would miss one of the more fundamental points about SOA benefits: delivering increasing value to the business from IT investments through the adoption of an SOA.

Another problem to avoid when defining an SOA project is devising solutions that are IT-centric. In other words, the term “business service” has to be used to refer to a component that the business can understand in business terms. This is a critical point that will impact the business’s involvement in and joint ownership of an SOA project. If the business does not see the alignment to business services, the solution will in all likelihood end up as just another technical solution. This may also result in a technically complex set of service interfaces that do not help to deliver the agile infrastructure the enterprise needs.

It is important to understand the definition of an SOA and the expected benefits when embarking on an SOA project.

Chapter 3

The Benefits of Adopting an SOA

Who specifically benefits from an SOA? The short answer is the business, for example, through cost savings, revenue growth, and increasing customer satisfaction. The longer answer is that if done right, everyone across the entire organization will benefit, even your partners and customers.

It's important to define the key benefits you expect from an SOA project, then use those expectations as a guide when you document in more detail the practical implementation aspects of your SOA project.

Examples of key SOA benefits include:

- Decoupled services providing better management of complex systems as well as increased business and technical agility
- Services reuse driving speed of implementation and lowering costs
- Open, pluggable infrastructure allowing more choice of components and technologies, reducing vendor lock-in and risk
- Service-based approach aligning business objectives and IT capabilities, helping IT deliver real business value

Different functional areas within an enterprise will benefit from an SOA infrastructure in a variety of ways, as illustrated in Figure 1.

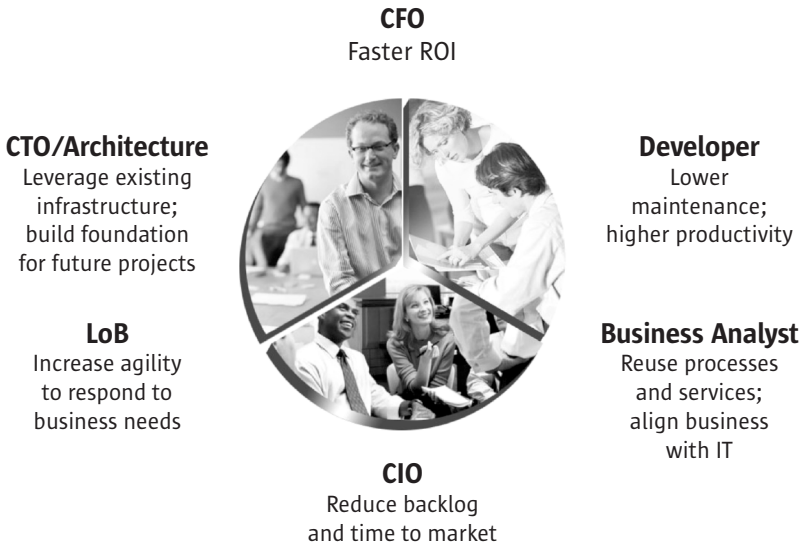


Figure 1: The benefits of adopting SOA

In today's business environment, however, an SOA infrastructure is also a competitive advantage with partners and customers. With growing expectations for efficiency, operational transparency, and speed, many organizations are looking at how to effectively implement an SOA to meet those demands.

Chapter 4

The Toolset for SOA is Important

One of the factors to consider when embarking on an SOA-based business integration solution is that it is actually an "integration architecture" solution. And as such, it needs to cover a wide range of different integration styles required by the enterprise, including:

- Application-to-application integration (A2A)
- Business-to-business integration (B2B)
- Business process management (BPM)
- Human workflow integration (Workflow)
- Business activity monitoring (BAM)
- Extract, transform, and load (ETL) for data warehousing
- Single customer view (SCV)
- Composite applications

And all this should be within a single architecture.

One approach often taken by developers to cover these different integration styles is to consider each independently and select a suitable product/toolset for each style from one or more vendors. This approach adds dramatically to your people costs, reduces reuse, and adds additional coding and complexity. Additionally, this disjointed approach can break your original "architecture" and result in a much higher total cost of ownership (TCO).

Many software product vendors today have made their products SOA-capable by adding a layer of software that effectively provides a Web services wrapper. This additional layer sits on top of the key components that you build. In the more extreme cases, the creation of this Web services wrapping is left up to you. Both of these approaches lead to complex tooling, increased developer training, duplicated development effort, and less flexibility going forward.

Most SOA software vendors fit into one of two areas:

"We can do it all with our mix of disparate products."

"You need to partner with other vendors for the full solution toolset."

Alternatively, using a single product suite to implement an SOA-based enterprise integration solution can avoid these costly issues, significantly lowering TCO. A single product suite provides a level of integration that enables an enterprise to quickly meet business challenges — without the need to integrate and maintain numerous point products.

Chapter 5

Sun's SOA Solution

Sun offers a single product suite with the Sun Java Composite Application Platform Suite (Java CAPS). Java CAPS is the first Java Platform, Enterprise Edition (Java EE) certified integration suite for SOA. Java CAPS was built using open standards throughout, so every component created is natively a reusable service. This dramatically reduces your implementation efforts without adding unnecessary complexity to the solution.

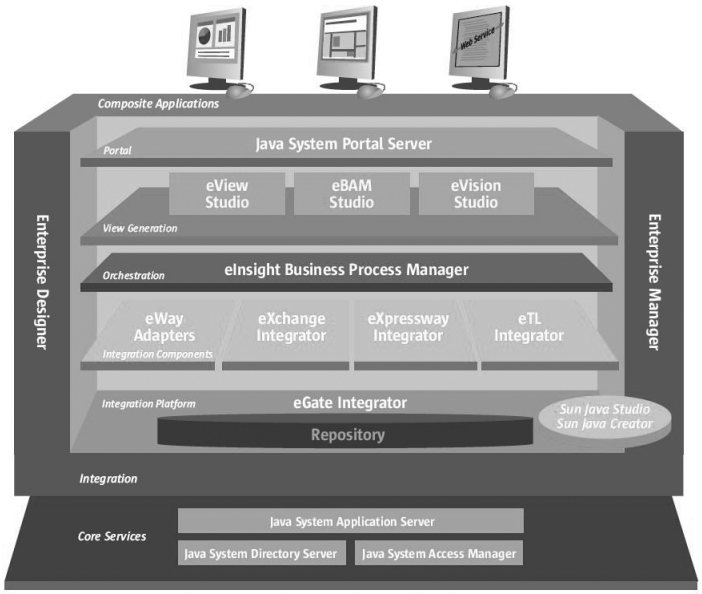


Figure 2: Java CAPS functional illustration

Java CAPS contains everything you need to develop and deploy an SOA solution. With Java CAPS, you can reuse existing applications, deliver new services, and integrate legacy and packaged applications within an existing infrastructure without extensive coding. In addition, Java CAPS combines business process modeling and orchestration, convenient interface design, metadata management, and strong development tools — all in a comprehensive platform built on open standards.

As stated in the introduction, this paper provides a detailed look at implementing SOA-based solutions with Java CAPS.

Chapter 6

The Layered Approach to SOA

Another key concept in SOA is the layering or classification of service types. If you consider how a "top-down" implementation approach would be tackled, you can get a better understanding of these layers.

Step 1: Business challenges and processes

The first step is to document the business challenge, then define the business processes needed to meet that challenge. For example, if the business challenge is to automate the sales fulfillment process to increase speed, reduce costs, and improve customer satisfaction, document the high-level, end-to-end business processes involved.

Step 2: Business services

This leads to defining the business services needed to support those business processes. For example, three business services needed within the above processes might be "Update-CRM," "Place-Order-On-Supplier," and "Check-Credit."

Step 3: Technical services

Next, the business services need to be implemented utilizing technical resources (or services). Assume we have a data center where we have a Siebel CRM version 7.5 implementation with its own customized data structures. We would then need to implement the Update_CRM business service by mapping the business service interface to Siebel's proprietary interface, namely the UAN/XML data structure specific to the data center instance and the Siebel EAI-Post transactional interface. This application integration is done through building technical services based on integration adapters and transformation mapping from application centric data structure to the business service interface representation.

This top-down implementation approach is summarized as follows:

The Business Challenge	Supported by	Business Processes
Business Processes	Supported by	Business Services
Business Services	Supported by	Technical Services

Step 4: Presentation services

For presentation services, consider how a user interacts with business processes. For example, the sales manager needs to authorize orders when the credit check fails for an existing customer placing a new order. If this user form does not exist in any current application, then you need to be able to design a new user interface screen. If your computer users can continue to service all of their business obligations from the existing applications' user interface screens, then there is no need for this presentation layer. However, if you need new user interfaces, this type of solution is what we refer to as a "composite application."

A final consideration is if you want a business service to be understandable by business people, such as business analysts, business department managers, and project managers, you need to make the interface understandable to them. This requires business-centric common-object models or canonical data formats. This will be discussed in more detail in Chapter 9 under *Common message formats*.

Chapter 7

The Agile Infrastructure

Businesses must be able to change quickly to take advantage of new market opportunities and stay competitive.

For example, consider an instance in which the flow of a business process needs to change (Figure 3). With an SOA, it should be possible to implement this simply by reorchestrating business services into the new sequence without any new development effort. This is achieved with a simple drag and drop of the relevant activities described in the business process flow at design time and automatically reimplemented as part of the runtime solution.

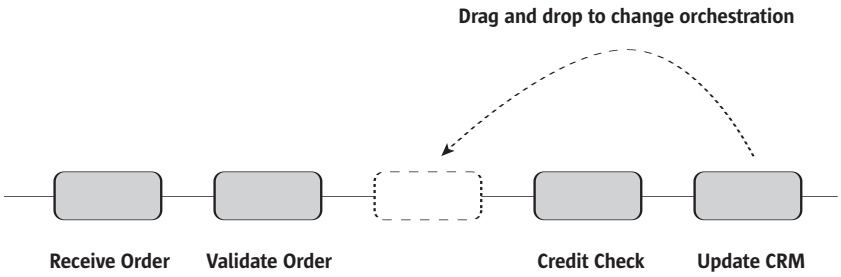


Figure 3: GUI rewiring at the business service interface; no development needed

In some cases, new business services will also be required. These can be developed in isolation and then orchestrated at the right time together with the other services that support the business process.

This approach allows maximum reuse of existing business services regardless of where they were built or where they are currently being used. This also requires little or no applications domain knowledge to reorchestrate the solution. The technical implementation team need only to get involved in new development when new or modified business services are required.

Business processes are inherently linked to technical implementation, but a business process designer should be able to redefine business process flows without having to rely on IT technical staff to implement it. This is the value of an SOA at the business level: providing increased flexibility over what is normally a more complex set of technical components.

Consider another issue. Assume users want to see a change in the user interface. A user-driven application is defined by the data entry and inquiry forms that the user sees in addition to the screen flow sequence that guides them through the tasks that they perform.

These user applications tend to get the user forms mixed in with business logic, removing any chance of reuse and making maintenance difficult. In an SOA solution, the presentation should be isolated from the business logic. To implement this, you need a layered approach for services, such as the four-layer SOA model.

Chapter 8

The Four-Layer SOA Model

A layered approach to SOA is essential, but most SOA implementations are paying little attention to this today and are therefore not realizing the benefits expected. This four-layer SOA model helps you maintain focus on both the business and technical IT components of the solution. The four layers are:

- Presentation services (e.g., Web pages)
- Business processes (orchestration of business services)
- Business services
- Technical services

In a true SOA implementation, every component you build (and therefore every service you build) should fit discreetly into one of these layers. If the layers are ignored, everything gets named as "business services" in a maze of technical interconnections. The result will be that you will still need technical experts to implement any type of change, which is certainly not the business agility that SOA is expected to deliver.

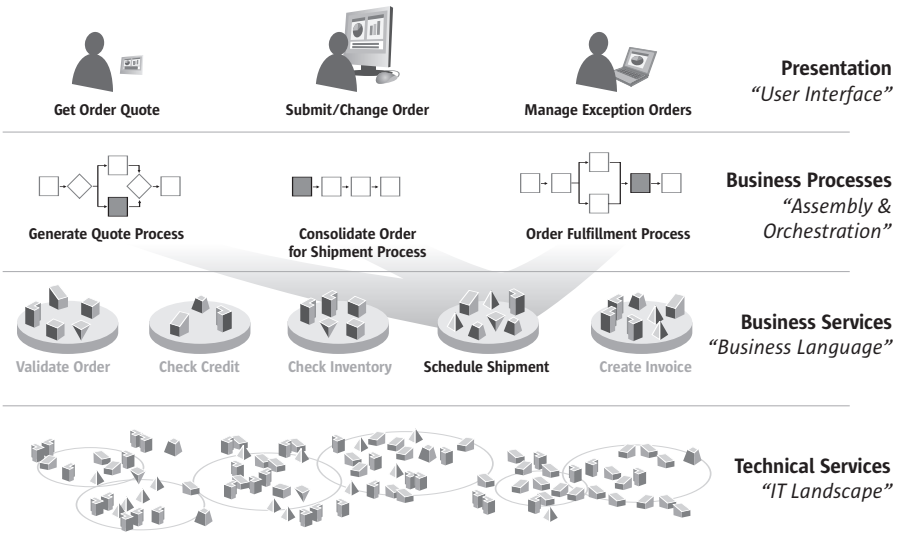


Figure 4: Four-layer SOA model

To be able to classify any particular service artifact into one of these four layers, it is important to clearly define the differences between the layers.

Presentation services

The presentation layer involves anything to do with screen layouts, (e.g., HTML and JavaServer Pages™ (JSP™) technology, images, style sheets, etc.). This layer is owned by business users, but that does not mean that business users do the actual build; rather, the layout of the screen is defined to support the requirements of business users and built to their satisfaction.

In Figure 5, notice the four-layer SOA model in the left-hand panel "DevLife_OrderProcess" project. The selected Web page is "pgOrderUpdate" and is contained in the presentation layer.

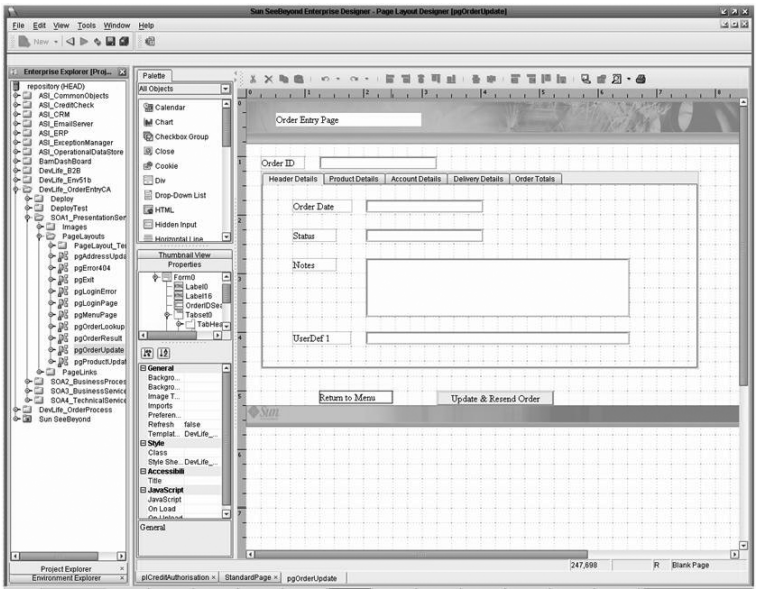


Figure 5: Java CAPS Enterprise Designer — editing a user form

The left-hand panel in the Java CAPS Enterprise Designer is used to create, edit, and move project folders. This is where you can build any type of components needed in your solution. The first step in a new project would be to create project folders in an SOA hierarchy (root project folder and then subproject folders) to suit your new development. To create a root project folder, click on the top-level item in the project panel — the repository. Right click, select "new project," give it a name, and the project folder is created. To create a subproject folder, click on the root project folder and repeat as above. Any level of project folder structure can be created.

In the Java CAPS Enterprise Designer, you can create any type of component simply by selecting the project folder where you want it to reside, right clicking, and then selecting to object type.

Business processes

Business processes are the processes as defined by a process designer (a business analyst or lead consultant) using business terms. Each individual step (activity) within the process is a business service.

It is vital that the business processes being designed can be implemented. The language of the business process must stay business focused and not become complicated with technical detail. Technical implementation details should be pushed down into the technical layer and only be accessed by business processes through a business service interface.

Figure 6 shows a process model built using drag and drop. To build a business process, select the project folder in the projects panel (e.g., “Devlife_OrderProcess/SOA2_BusinessProcesses”). Then right click and select new “business process.” This gives you a blank process model as your starting point. You can now drag and drop on the activities you want, wire them together, and test the model, all within the graphical build panel. Business Process Execution Language (BPEL) code is automatically generated in the background.

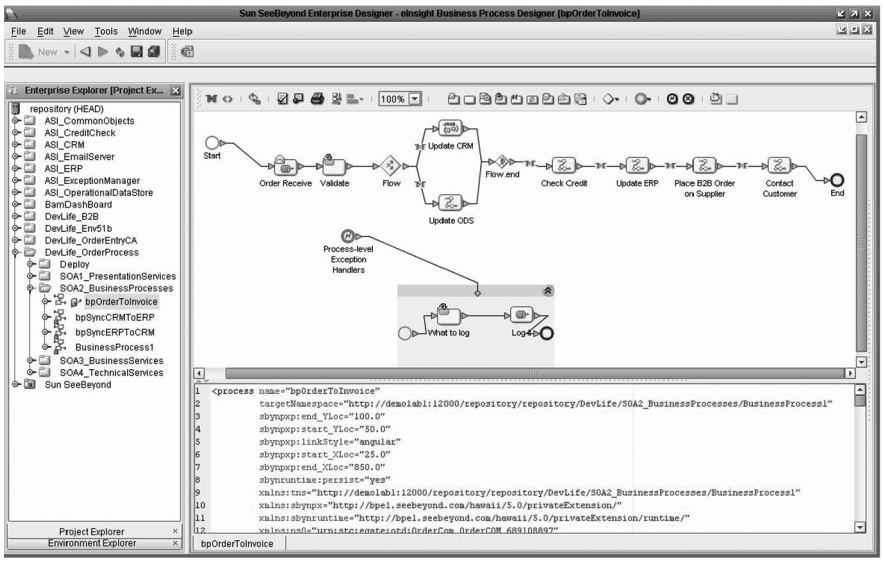


Figure 6: Java Caps Enterprise Designer — designing a BPEL process flow

Business services

Business services are the activities defined in the business process. They should be in business terms with no reference to the physical systems or locations where the systems run (because that information is technical and will change when the technical implementations change). Business services should be fully decoupled from technical implementation.

The business process names the business services that IT will need to implement. A well-defined business service will have its service interface defined at this level. The business service interface, along with its name, defines what needs to be implemented by IT. It is the unit of work that can be implemented in isolation.

For example, a business service to create an order in the customer relationship management (CRM) system could be defined as CreateOrderInCRM(OrderCom, OrderCom, ErrorCom), where OrderCom is the common object model for an order and is used as the input and output data structures for the service call and ErrorCom is the common object model for returning error information. We will see how these common objects are derived shortly.

Figure 7 shows a business process model with business services dropped on the activities.

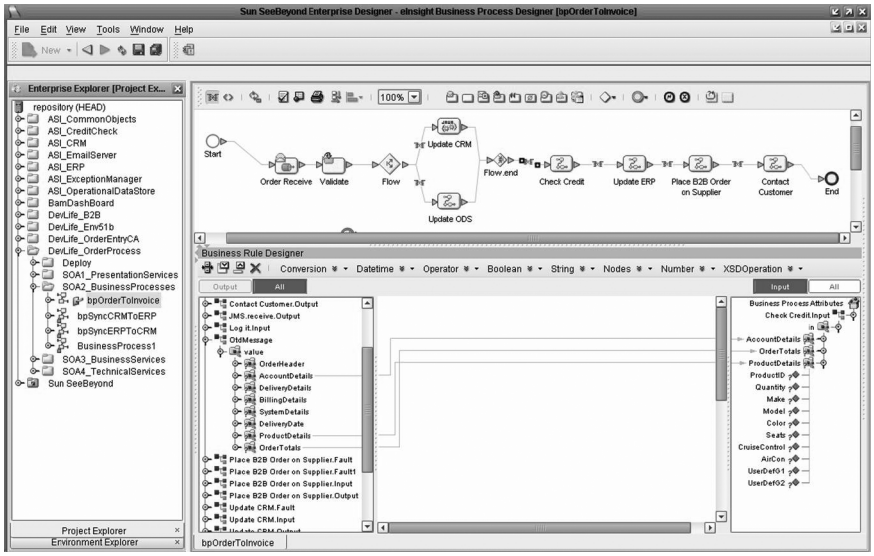


Figure 7: Java CAPS Enterprise Designer — mapping business service interfaces in a process

The business rules designer section of the Sun SeeBeyond eInsight™ model in the Java CAPS Designer shows mapping an OrderCom object into a specific business service that has the same data interface. Note that there are 150+ fields in OrderCom that can be mapped from one service to another by simply mapping the top-most node, or it could be mapping individual fields within the OrderCom object and even transforming the data as it is mapped.

Technical services

The likelihood is that most of your current IT artifacts (applications, code segments, reference tables, etc.) fit into the technical services level, but these are only true technical services when you service-enable them (i.e., expose a service interface to make it easily reused). The business analyst has no need to be involved in the technical services layer.

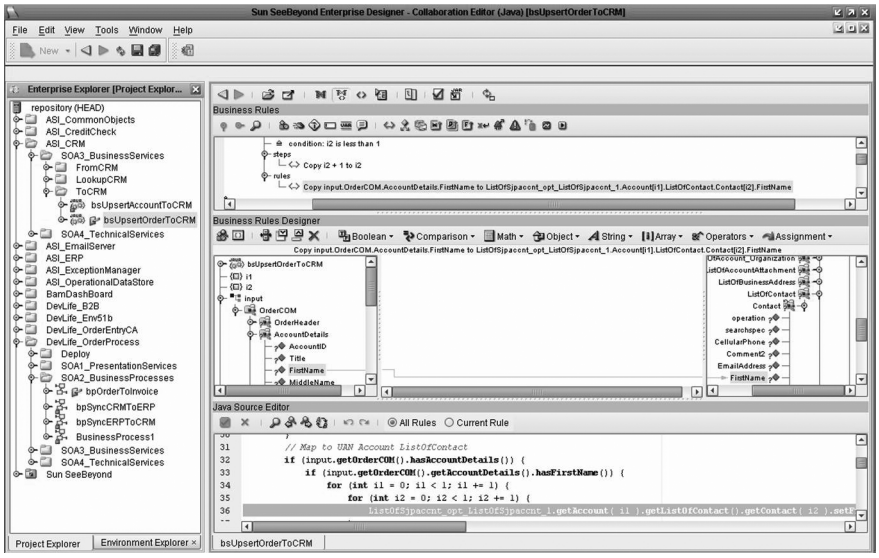


Figure 8: Java CAPS Enterprise Designer — implementing a transformation in the Java platform

Any type of object can be built by selecting a project folder in the projects panel, right clicking, and selecting the type of object. This creates the object and gives you the associated object editor in the right-hand panel.

The benefit of this four-layer approach is that each of the layers are decoupled from the neighboring layers by their service interfaces. By keeping the interfaces stable, you can change any service implementation within a particular layer without impacting anything in any other layer, giving your organization the ability and agility to quickly make changes as market forces or new strategies require.

Chapter 9

SOA Key Components

To get a better understanding of how this all works, let's explore the components on which an SOA relies:

- Common message formats
- Business-level service interfaces
- Technical-level service interfaces
- Service enablement (and service creation)
- Service orchestration
- Repository organization

Common message formats

Business analysts, process designers, project managers, and business department managers should all be able to freely communicate on business processes and business services without reference to any specific application or technical knowledge. Similarly, the technical staff should be able to converse at the same level and easily translate business language into technical challenges. Key to this ability is the use of business-level common object models (also known as common message formats, canonical message formats, common data models, or just common objects), which allow all those involved to use the same data model and therefore the same language, removing potential misinterpretations between IT and business.

The common object models link the services between the SOA layers. These are a critical component in an SOA framework and a cornerstone of how Java CAPS supports delivery of an SOA framework. You should be free to choose the best common object models to suit your business, and you should not be forced by the SOA technology to base this on any particular standard, for example, the eXtensible Markup Language (XML).

Three key sources for common object models are:

1. **Open standards and industry standards bodies**

   For example: SWIFT and FIX for finance, OASIS BODs or Rosettanet for retail, OSS/J for telco, and HL7 for health care. These may provide the best "open" model, but may also overcomplicate the common object model, making it difficult for all parties to understand and use.

2. **Application vendor-led standards**

   For example: SAP IDOCs and Siebel UAN. These are more pragmatic standards that may work if your application space is predominantly driven by one of these applications and if this is a good representation of your business as a whole. However, these standards tend to lock you to that vendor, reducing the "no vendor lock-in" aim of an SOA.

3. **Build your own**

   This is the most pragmatic approach of all. Look to your current systems, look to the business understanding, and then define your own common objects. This does not have to be extensive. You may only define 10 to 20 types of objects, and this could be the easiest to understand for both business and IT and the fastest approach to implement.

Whichever approach you take, it is important that this model is straw man tested against a number of scenarios to ensure that it is stable across business and IT changes. The SOA platform should allow you to make simple changes, such as adding fields or substructures; however, you don't want to make extensive structural changes such as restructuring multilayer data nodes.

For example, consider the objects associated with an order. You would need to handle customer details such as address, contact details, credit status, delivery details, product details, order values, etc. Each of these objects has a number of fields, for example, delivery address fields could include street name, town, city, and postcode.

Let's look at the following example. To determine the best representation for each of these, we can look to our existing systems and then define them in XML.

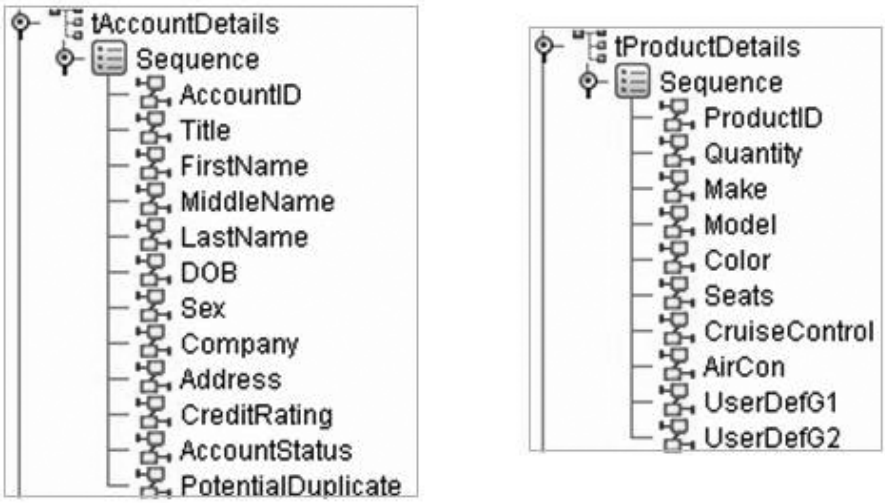


Figure 9: Two common objects

We can now combine them into a single OrderCom composite object that can be used as the standard interface for any business service that deals with orders.

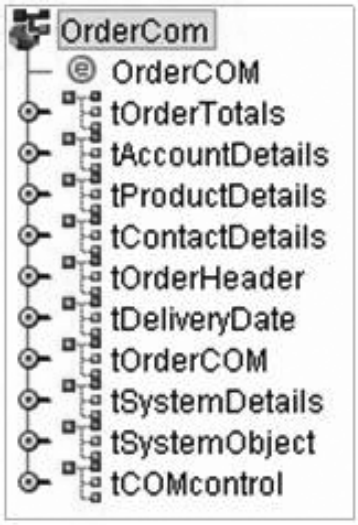


Figure 10: OrderCom — a composite order common object built in Java CAPS Enterprise Designer

Some business services only need a subset of these common objects, while others need the full structure. For example, the credit check service only needs "account" and "order total" common objects, while the delivery service only needs the "product details" and the "delivery address." You would need to create two composite service interfaces for these two services.

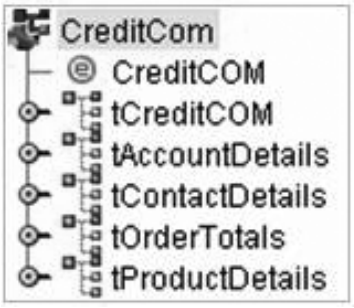


Figure 11: CreditCom — combining account and order totals objects

Within Java CAPS, you can either create your common objects directly through a forms-based object type definition (OTD) editor or import metadata, such as XML Schema Definition (XSD). Both mechanisms require no coding.

Business-level service interfaces

It is vital to define the business service interface in business-specific terms, not technical terms. This is the most critical layer because it links the business requirements to the IT solution and ensures that business and IT have a common understanding of the proposed solution. There are two key parts: the naming convention and the data definitions (i.e., the common object model).

The business services are the key units of work that IT needs to meet the project goals. How these business services are orchestrated into business processes is what delivers the business objective in a flexible and agile business solution. For example, consider:

- UpdateCRMAccount(AccountCom, AccountCom, ErrorCom) service. This would indicate that it is a business service because its operation name uses business terms and is not specific to any physical application implementation.
- Update UpdateSAP4.6cAccount(AccountCom, AccountCom, ErrorCom) service. Even though this might have the same common objects, this would be a technical service because its operation name references a specific application. You would be forced to change this service interface if the target application was replaced with another application, or even if it was just a simple application version change. Any service interface change then forces every service orchestration to be changed and most likely requires a full implementation release, not just a minor update. Hence, it is not a very robust or agile solution.

A business service interface also defines the structure of the data to be passed into and out of the service. There should be common object models for complex services (such as AccountCom for account) and simple parameters for less complex services interfaces.

If you use application-specific data structures in the service interface, you will need an application domain expert — not just to build the service, but also to use the service (wire it into processes). This would therefore be a technical service and not a business service.

One potential issue to avoid is allowing technical services to be confused with business services. Basically, business services are defined by the business analyst. Any components that the implementor needs to build to implement the business service is a technical service. Mixing business and technical services will pollute the business domain with technical detail and turn the solution into just a technical solution, breaking the business link.

Business services should be defined in Web Services Description Language (WSDL) and published in a Universal Description, Discovery, and Integration (UDDI) registry, runable on an enterprise service bus (ESB) or in a hub, accessible as a Web service or at a lower level (for performance), and transparent to the way you build the services. Java CAPS provides this functionality out of the box.

A business service will “internally” call technical services and optionally other business services, but the business service interface should not change for any application-centric changes and should rarely need changing — even for business changes.

Figure 12 illustrates how easy it is to map business service interfaces, no matter how complex the target systems. Even the most nontechnical business analyst will be able to understand the impact of changing processes.

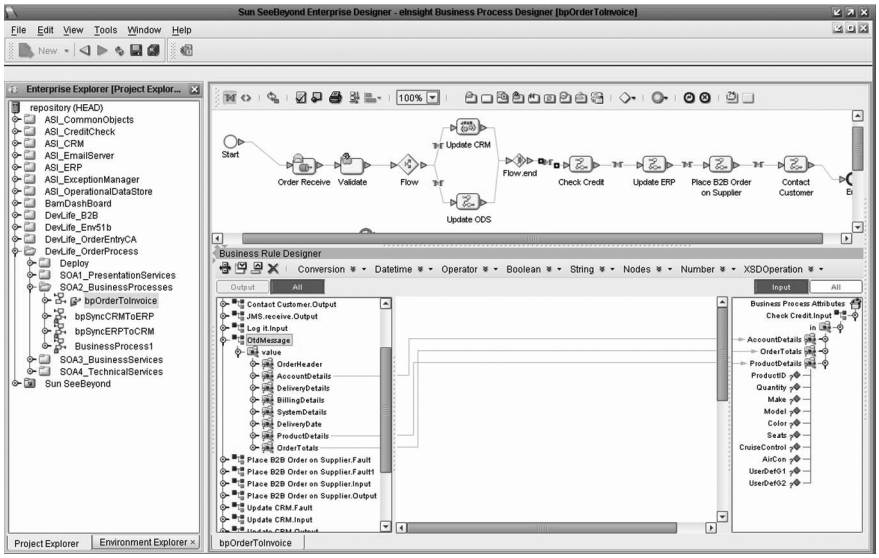


Figure 12: Java CAPS BPEL for mapping business service interfaces

Clicking on a line between two business services (yellow boxes) in the business process brings up the service interface mapper, as shown in Figure 12, bottom right-hand panel. From there, you can map common object data fields from any output service into any input service using a set of transformation functions and data enrichment options. This is a simple drag-and-drop interface: the function is known as service interface mapping.

Technical-level service interfaces

Technical-level service interfaces should also be defined in WSDL and published in a UDDI repository, runable on an ESB or in a hub, accessible as a Web service or at a lower level (for performance), and transparent to the way you build a service.

What makes a technical service different from a business service is that it tends to be application-specific. Its interface can be the business-level common object model if it is suitable, or the lower-level application-specific interface. Regardless, this is where the complex implementation work is done.

Service enablement

The single most time-consuming and complex task in implementing an SOA is service-enabling the valuable assets that you already have. Examples include the call center operation through CRM, order management in enterprise resource planning (ERP), or logistics through third parties. Any new business challenge will be easier, faster, and less expensive to address if you can reuse much of what you already have. Once you have service-enabled these assets they become even easier to reuse.

Service enablement is an SOA term for enterprise application integration (EAI). It encompasses all of the same requirements that any integration project requires, but delivers them in a more open and service-oriented way. To service-enable an existing application, you need to do the following:

1. Define a business-level service interface in WSDL that incorporates a common objectmodel. This is a task done by the business analyst and the lead consultant or architect.

2. The technical implementation, which is logically “inside” the business service, needs to:
   - Map this business-level common object model to and from the proprietary application message structure (such as IDOC for SAP, UAN XSD for Siebel, etc.)
   - Connect to the application using an application adapter
   - Manage the connection, transaction, and exception logic

The technical implementation could be done as a single unit of work inside the business service (where reuse is only at the business-service level), or it could be an orchestration of a number of individual technical services that could each be reused in other business service implementations.

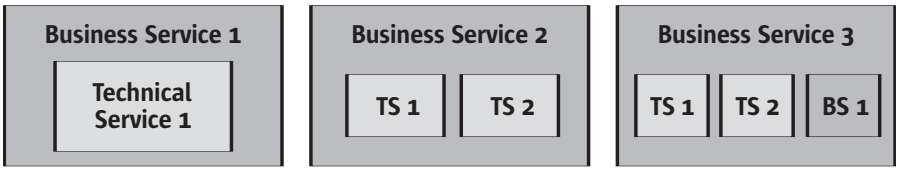
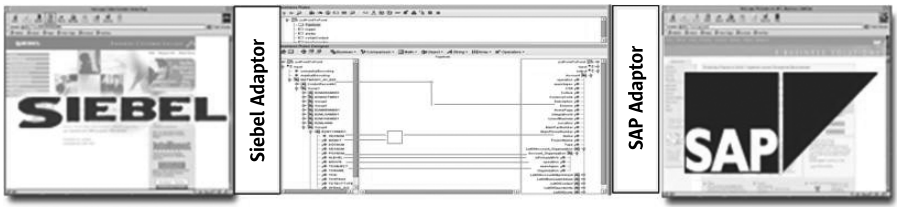


Figure 13: Technical services inside the business service interface

For example, consider the difference between a conventional application-to-application integration and an SOA-based service enablement of existing applications (both of which are possible in Java CAPS). Assume the goal is to synchronize a customer record change being made by a user of the CRM application with an automatic update of the same customer record in the ERP system.

Case 1: Conventional application-to-application integration



The key components in this scenario are the adapters (Sun SeeBeyond eWay™ Adapters) for each application, the message structure representation (object type definitions or OTDs) for each application, and the transformation mapping (Java or eXtensible Stylesheet Language Transformations (XSLT) collaboration editor) from one message structure to the other. The entire construct between the two applications is not a service because it only does one thing: pick up an account change in Siebel and convert it to an SAP account change. It most likely only works for these exact instances of the applications since other Siebel and SAP application instances could have different underlying application data structures.

Also, it cannot be reused because the adapters and message transformation components are hardwired together. There is not actually a usable service interface. Further, if either application should change for any reason, this transformation will need to be changed. That means it is twice as fragile as either application and will require domain expertise of both applications should either application force a change.

Case 2: SOA integration (service enablement) using Java CAPS

With SOA, you service-enable both applications independently. Notice that the transformation is from the application to the service interface (common object based). This service is now an open service, defined in WSDL and published in a UDDI, that can be run on an ESB and can be accessed by any application that wants to publish or subscribe to that service event.

To better represent this in a service bus diagram, we will turn the construct through 90 degrees.

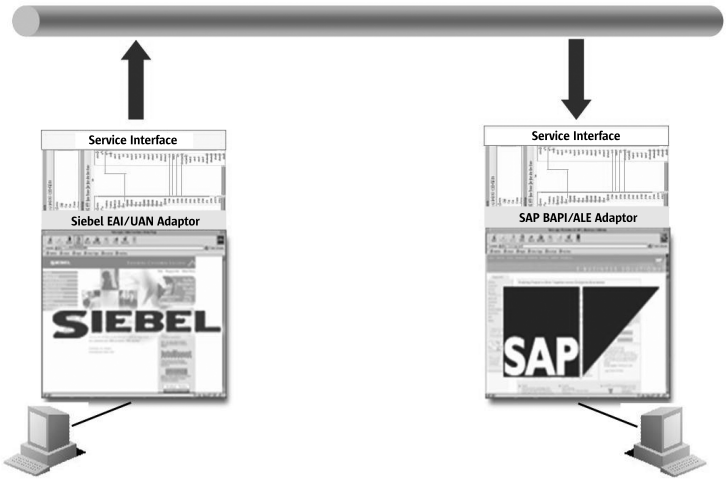


Figure 14: A classic service representation

To implement synchronization of CRM account changes into SAP in an SOA implementation, we need to orchestrate the Siebel publish service with the SAP subscribe service. We could do this by using a publish-to-JMS (Java Messaging Service) service. This is done by simply dropping the JMS service on the newly created Siebel and SAP services onto what is known as a connectivity map. Then a single button click will build the solution and another deploys it to your message hub or ESB.

Service orchestration

Service orchestration is where services are organized into the tasks the business needs done. From the business perspective, this is the business process management layer, where business services are orchestrated into business processes. It is at this level where business analysts and business managers agree on the physical business processes with the process designer. These processes can be defined, reorganized, and augmented without the need to delve into costly IT development work. This is the business level of service reuse.

Figure 15 shows a simple two-step service orchestration of the Siebel-to-SAP account synchronization using BPEL in Java CAPS. The first service publishes the change while the second subscribes to it. These would be the same services detailed in Figure 14, but this time BPEL is managing the orchestration instead of just using JMS to orchestrate it.

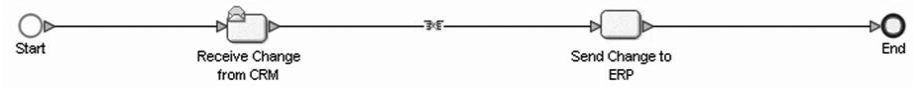


Figure 15: Service orchestration with a Java CAPS BPEL engine

If you look at it from the view of one application (e.g., Siebel CRM), you can see that for business objects you want synchronized across all business systems, you would implement two services: one to publish changes made in the Siebel application and one to subscribe to changes made from any other system to be inserted into Siebel. If you need logic to dynamically manage the distribution of these updates, then the process level is the perfect location for this.

Repository organization

With numbers of services that combine to make a solution, you need a single repository in which to store them. The repository needs to be organized so that the services can be easily searched, built, and enhanced in isolation; version controlled; and deployed in a controlled manner.

Figure 16 shows a standard four-layer SOA project structure in Java CAPS that provides a location for every type of artifact during the build. Note that in Java CAPS, every artifact is actually a first-class service with a WSDL definition, although you get to choose how you want to expose them at runtime.

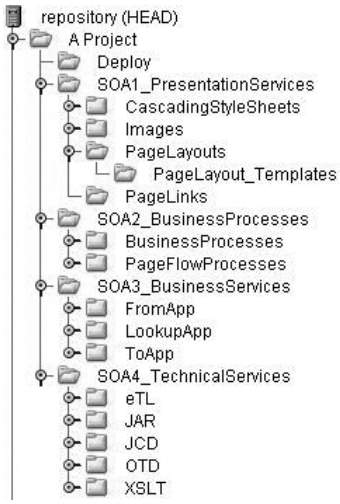


Figure 16: An SOA project structure in Java CAPS repository

The project panel in the Java CAPS Enterprise Designer is the access mechanism into the repository where every type of object is stored. Since we want a layered SOA model to help tie business and technical implementation together, we implement the four SOA layers into the project structures in the repository.

In Figure 17, you can see a root project called “ASI_CRM.” This is an application service interface project for the specific application, the Siebel CRM in this example. These CRM services are then completely decoupled from the rest of the solution, making them easy to implement and manage in isolation.

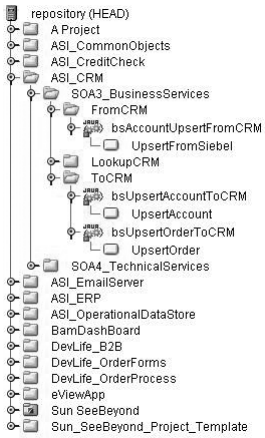


Figure 17: The CRM ASI project structure in a Java CAPS repository

In the “business services” layer, there is a third type of service known as a “lookup,” where the application is providing a lookup service to other systems. Not all of the four SOA layers are needed in every root-level project. Notice that in Figure 17, the CRM project does not have any presentation services or business processes, so these two levels are missing.

In Figure 18, there is a four-layer SOA model inside each of the root projects. A root project represents one of the following:

- A business process or process group implementation (e.g., order processing)
- A composite application implementation (e.g., quote application)
- Service enablement of specific applications (e.g., ASI_CRM)
- A grouping of common services (e.g., shared common objects)

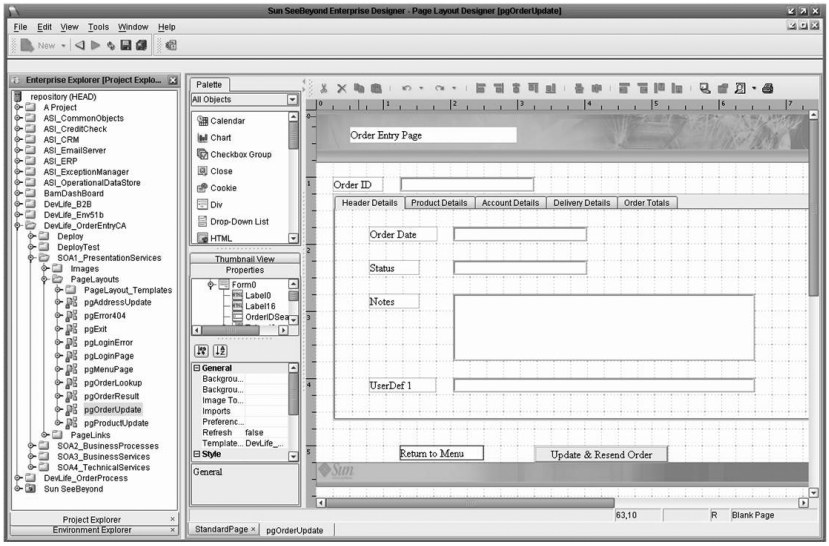


Figure 18: The Order Entry composite application project in Java CAPS

Notice also that the “Quote” project has all four layers. For example, the business services level will have any new business services that have been built which are only required to support the quote composite application.

Chapter 11

Integration Styles and SOA

As mentioned earlier in this paper, there are many integration styles. The following will address how each of these styles fits into an SOA infrastructure.

Business process management in an SOA

So where does business process management fit in? As long as each activity in a service orchestration is a business service, that service orchestration is a business process. If any activity is a technical service, the service orchestration is a technical service flow. For example, Figure 19 shows a business process to synchronize account changes with multiple other services.

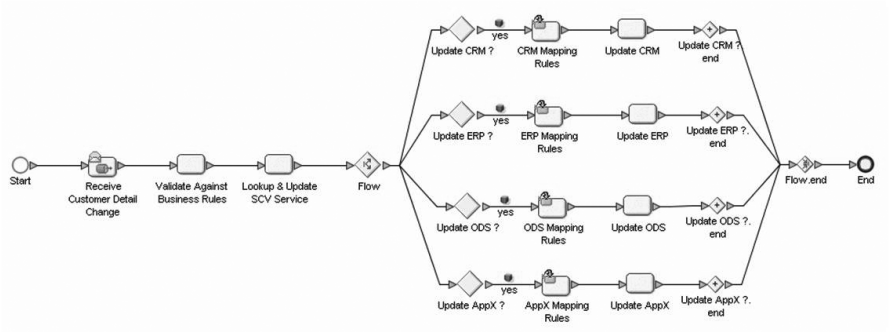


Figure 19: Synchronization of account changes across multiple systems in eView

Note that the “lookup” business service returns the account IDs in each of the disparate systems. This service is supplied by the Sun SeeBeyond eView™ Studio module in Java CAPS.

Business processes defined by business analysts should be the exact process that controls the solution at runtime. Business analysts should be able to monitor their processes through a real-time graphical representation of exactly the process model they defined at the start of the project. This is provided out of the box with Java CAPS.

Enterprise application integration for service enablement

EAI is a critical aspect of any SOA project, yet pure-play ESB vendors tend to be weak here. Often, vendors make the assumption that every existing application is already service-enabled with a simple XML interface that is accessible as a Web service. If this were the case, you could pick any ESB product. The reality is that most legacy systems are not service-enabled, and any that are tend to have complex technical XML interfaces, which makes them unsuitable for business service interfaces.

The Java Collaboration Editor is an object editor inside the Java CAPS Enterprise Designer. It provides a graphical drag-and-drop interface that generates pure EJB code (see code representation in Figure 20). You can even type in the Java code directly and watch the GUI representation being built or do the reverse, dragging and dropping a transformation rule and watching the Java code being generated. You can also edit the Java code in a third-party source code editor and then import it back into the Java Collaboration Editor (known as round-tripping).

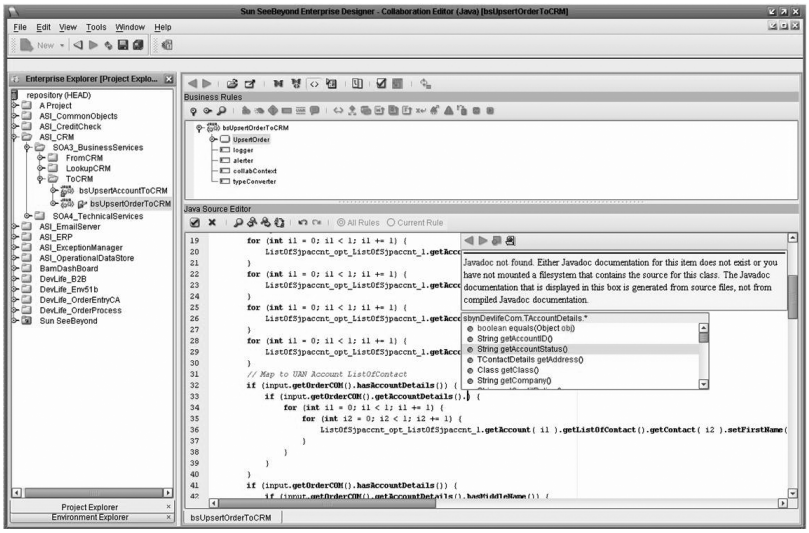


Figure 20: Java CAPS Enterprise Designer — Java transformations

The inclusion of the Java Collaboration Editor means that you have the full power of the Java language to enable you to build any level of transformation capability into your technical services.

Adapters for service enablement

Another key component to service-enabling your existing systems is adapter technology. Adapters allow you to connect to, request information from, and update the target system — all within a manageable transaction dialogue. Adapters are key components to service enablement used within each business service that needs to connect to systems.

Java CAPS has more than 80 highly functional eWay Adapters that cover the full range of connectivity styles, such as:

- Communications eWay Adapters: Connecting at protocol levels such as HTTP/S, File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), System Network Architecture (SNA), TCP/IP, etc.
- API-level eWay Adapters: Microsoft Com/DCom, CICS, application servers, Web servers, JMS, IBM MQ, Corba, etc.
- Database eWay Adapters: Oracle, Microsoft SQL Server, Sybase, Informix, DB2, Adabas, etc.
- Application eWay Adapters: Siebel, Oracle, SAP, Retec, etc.

For the occasional system where even the communications eWay Adapters won't fit, a new adapter can be implemented using the eWay development kit.

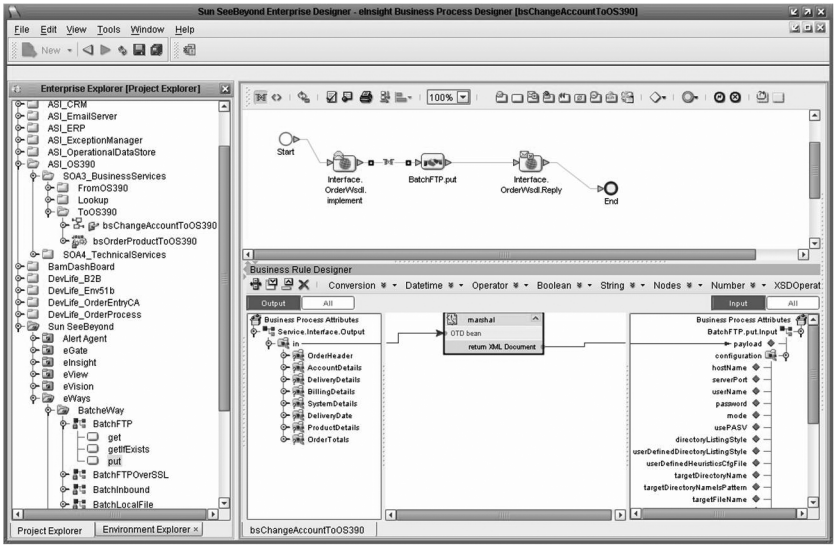


Figure 21: Java CAPS Enterprise Designer — mapping to a batch eWay with eInsight

Java CAPS is a fully open solution, so you can use third-party Java Connector Architecture (JCA) adapters in your solution.

Workflow in an SOA

Workflow can be characterized as the human being in the process. Processes that require human intervention also require defining who, in what role/what group as well as hierarchies for escalation/delegation, and time limits for alerts and escalation. Workflow is a subcomponent of BPM, although not all BPM products support workflow functionality.

Most business processes need workflow. For example, consider when an order is placed. Order-to-cash may be an automated process, but in some instances, a credit risk issue may require a manager to authorize the order. This requires workflow.

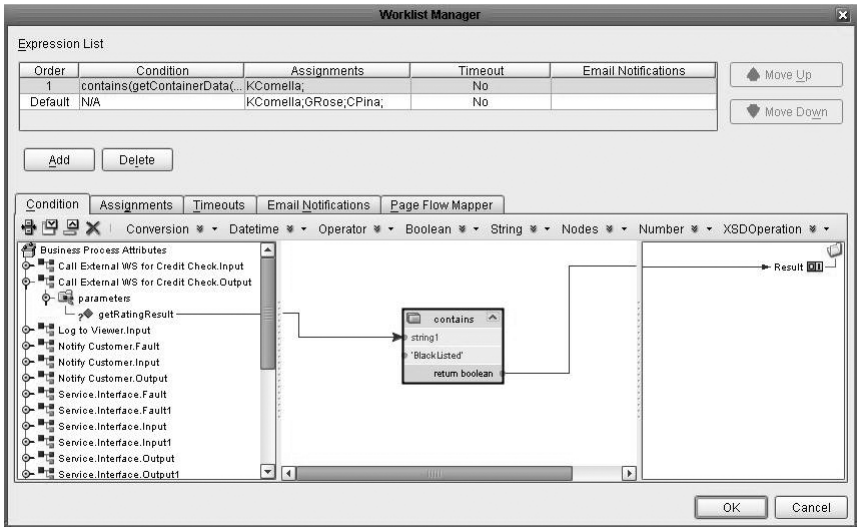


Figure 22: Java CAPS Enterprise Designer — configuring escalation rules

Within a Java CAPS business process, workflow is just another service type known as a user activity. For direct user interaction with a process (i.e., workflow), you would drag and drop a user activity onto the process model, then define the user, role, or group, the escalation details, and the forms the user should be presented with. This is all done within the eInsight process model in the Java CAPS Enterprise Designer.

Composite applications in an SOA

If you need new user interfaces that don't exist in your current applications (e.g., a new quote form that uniquely supports your business), you may require a composite application.

Composite applications are effectively business processes that are driven by users. For example, a quote application could be built that presents numbers of screens to the user, allowing them to construct a quotation for a sale. This application could be calling many services from many service-enabled existing applications and new services built specifically to support this new application. This is what is known as a composite application.

Figure 23 shows the first screen in a quotations composite application, where most of the functionality for this application actually exists in legacy systems. The pages are built by selecting the “page layout” object type in the Java CAPS Enterprise Designer, which then displays the page layout editor as shown in Figure 18. You can drag and drop the forms objects you want. To implement the “process” behind the forms, you simply design the flows with the eInsight process designer, linking pages with the services that you want to call.

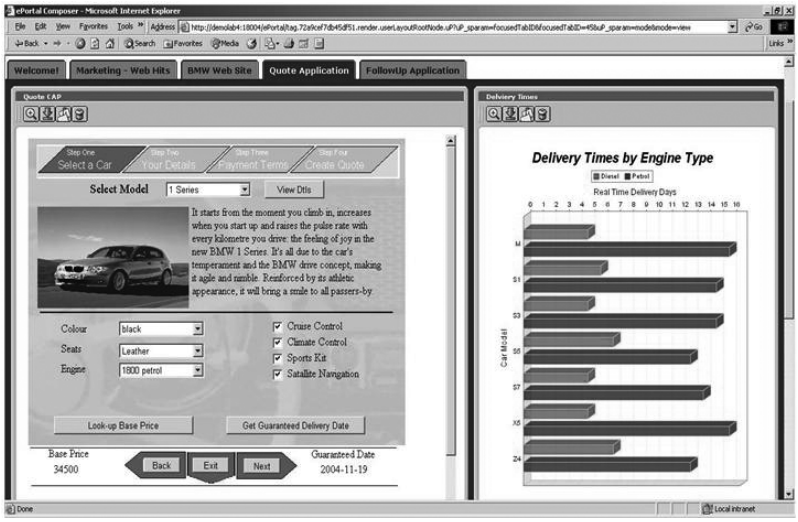


Figure 23: Quote composite application built with Java CAPS

Portals in an SOA

A powerful portal is an important component to successful enterprise integration and SOA projects because it provides end-user configurable access to the new business processes and composite applications.

With an open and standards-based architecture, the Sun Java System Portal Server (included with Java CAPS) provides a flexible framework for designing the service view of an SOA.

- Bidirectional Web Services for Remote Portlets (WSRP) support enables the portal to simultaneously produce and consume remote Web services
- An interportlet communication API lets portlets share information with each other to create a greater degree of dynamism and adaptive behavior
- Java and Web services standards support facilitates integration: UDDI, WSDL, SOAP, Java Specification Request (JSR) 168, and Java API for XML (JAX)

Portals provide a new level of enterprise productivity, enabling users and groups to work together easily and securely within the requirements of a dynamic organizational structure. Advantages of a well-designed portal include:

- Ensuring more rapid and cost-effective deployment of successful enterprise solutions by delivering identity-based content to employees and partners
- Providing end users with easy-to-use services that enable them to design customized community portals
- Adding compelling content and services to portal pages via custom development, with simple-to-use developer tools accessing Web services or data repositories through a content management system, workflow management, and enterprise integration

Business activity monitoring in an SOA

Among the challenges of automating business processes is that staff can lose visibility of what's happening. If you want real-time information rather than just weekly/monthly reports, business activity monitoring (BAM) is the answer. BAM relies on picking up events as they occur on the service bus; collating, aggregating, and checking against key performance indicators (KPI); and then providing information to dashboards or through alert channels.

There are three key components to BAM: collection of data, processing of data, and delivery to the user. Collection can be done at the business process level or via eWay Adapters. Processing is done via the Java CAPS BAM engine (Sun SeeBeyond eBAM™ Studio), and delivery to the user can be via any channel. The Java CAPS BAM engine is fully configured and managed within Java CAPS Enterprise Designer.

To build a BAM solution, select the project folder where you want to create the BAM components, right click, and select "new BAM object." Java CAPS will walk you through the build of the BAM solution, all from within the Java CAPS Enterprise Designer.

Figure 24 shows an executive dashboard where real-time data is being provided by Java CAPS and the dashboard itself is built using the Sun SeeBeyond eVision™ module within Java CAPS.

Figure 24: A BAM Dashboard built with Java CAPS

Single entity views in an SOA

Classic examples of a single entity view are customers, patients, companies, citizens, and products. Single customer view (SCV) will be referred to here, but the logic is the same for any type of entity.

It's a challenge to have a single customer view when customer information is gathered from many disparate sources. For example, a finance company might have a CRM system, ERP system, household insurance system, life insurance system, or trading system – and each system may have its own unique ID for a customer.

The simple answer to this challenge is to add a master database, and every time a new customer is added into any system, check the master database first to see if the customer already exists in another system. This has been the CRM solution for many years. Clearly this is not practical in most instances because it is not practical to change every other system to do a lookup on the CRM system first. It's just too expensive to change all of your existing systems.

A single entity view service needs to manage cross references between systems, provide real-time probabilistic matching, duplicate management, and full auditing. In an SOA, this is a critical service on the service bus.

To build a SCV service, select the project folder where you want it to reside, right click, and select the “New eView Object.” You will be presented with editors to define the entity model, define the probabilistic matching rules, automatically generate the application, and deploy it, all through the Java CAPS Enterprise Designer.

Figure 25 shows one of the automatically generated forms for use in managing potential duplicates in the system.

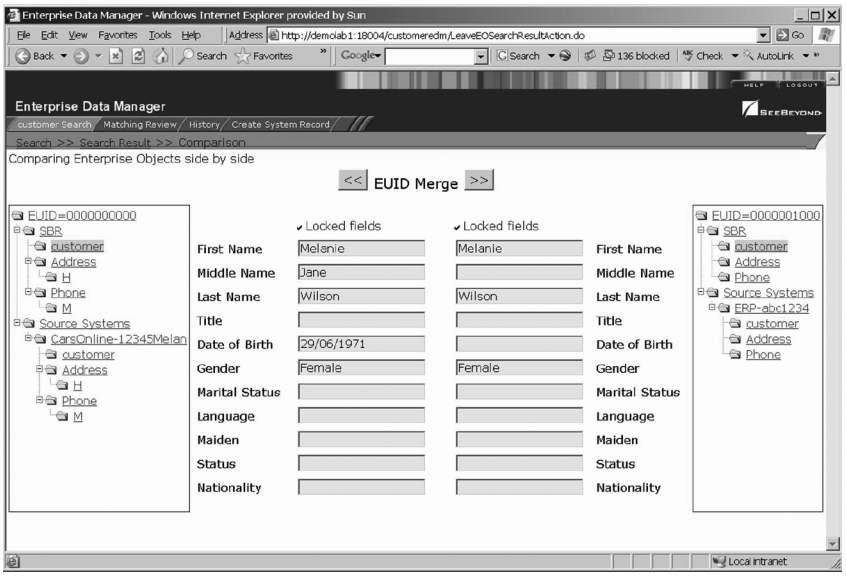


Figure 25: eView EDM — managing potential duplicates

Chapter 12

Summary

It is possible to deliver on the SOA promise. However, you need to implement it in a pragmatic way. You need to deliver results fast, yet on a project-by-project basis. Also, you need the SOA overhead to be minimal.

To meet the objectives of an SOA, a layered approach to services is necessary. Without a layered approach, all services can become complicated with technical domain detail, making the SOA solely a technical tool and keeping it at the domain of technical experts. Unless the SOA project remains on business objectives, it will not succeed.

Many software product vendors today offer either a disparate range of products or require customers to partner with additional vendors to get a full SOA solution toolset. Both of these approaches lead to complex tooling, increased developer training, duplicated development effort, and far less flexibility for your SOA infrastructure in the future.

Throughout this paper, we have covered the key aspects of an SOA implementation and shown how Java CAPS is specifically designed to enable an efficient and effective SOA implementation. In summary, key points to keep in mind include:

- Pick a business project – don't look to put in an SOA infrastructure as an IT project in its own right.
- Define the business processes from the business challenge using standard analysis and design methods. This should be owned by the business analyst.
- Agree on the names of the business services (the activities on the business processes) and draw the physical process model in Java CAPS.
- Define the business-level interface data structures needed for these business services. These will be the common objects, again implementing in Java CAPS.
- Implement all business services interface in Java CAPS (no coding needed) and drop it on to the relevant activity in the business process model.
- Leave the technical implementation to the domain experts to implement using Java CAPS. Each business service can be implemented in isolation.
- Use standard test and deploy techniques, fully supported within Java CAPS.

Learn More

For more information on Sun SOA solutions with Java CAPS, go to sun.com/soa. For additional information on Java CAPS and Java CAPS for developers, go to sun.com/software/javaenterprisesystem/javacaps and developers.sun.com/prodtech/javacaps

About Sun

A singular vision, The Network is the Computer™, drives Sun in delivering industry-leading technologies that focus on the whole system — where computers, software, storage, and services combine. With a proven history of sharing, building communities, and innovation, Sun solutions create opportunities, both social and economic, around the world.

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN (9786) **Web** sun.com

SunWIN# 497805 02/07