

Document and Process Oriented Architectures

Creating enterprise-wide reusable software,
and reducing complexity

by *Alessandro Vernet, Orbeon Inc.*

November 6, 2003

Published on Enterprise Architect

Traditional software architectures could be called Data and Operation Oriented, as they focus on the data model (for instance database relational model, or object model) and the operations (APIs). In this article we describe the Document and Process Oriented Architecture, which enables enterprise-wide software reusability and reduces application complexity by leveraging XML application servers. During this exploration, an imaginary scenario involving expense reports at ACME Inc. provides us with concrete examples showing how the architecture can be applied, and what its benefits are.

A Simple Scenario

Let's consider how ACME Inc. deals with expense reports. Right now employees start by printing the PDF expense report form created by the HR department. They fill out the form and take it to their manager. Managers approve the form by adding their signature, and take the form to the HR department. In the HR department, the person in charge checks the form and calls the finance to have the expensed amount added to the employee's next paycheck.



Fig. 1 – Handling expense reports at ACME Inc.

ACME Inc. wishes to automate this process. They envision a Web site that would be used both by employees to create the expense reports, and managers to approve the expense reports. They also want the system to send approved expense reports to the finance department automatically, to offload the person in charge in HR department.

The Traditional Architecture: Data and Operation Oriented

A group of engineers at ACME Inc. comes up with a solution based on a traditional architecture. They decide to implement it on the Java platform. They start by designing the data model for expense reports. Since days could pass between the time an expense report is created by an employee, and the time it is approved or rejected by a manager, expense reports have to be kept in a persistent store. To that end, they decided to use a relational database. So they write a data model both at the database level (SQL tables with relationships and constraints) and at the application level (object model). Getting data from the database to the application, and vice versa, is a complex task with many implications on performance, security and data integrity. For this reason, they decide to use the Enterprise JavaBeans (EJB) Container Managed Persistence (CMP) technology that solves some of these problems.

Then they define operations on the expense report data model, like "create new expense report", "approve expense report" and "send expense report to finance department." They decide to use the Stateless Session Enterprise JavaBeans technology to implement these operations. They opt for the Struts and JavaServer Pages (JSP) technologies to create a Web interface. To communicate with the finance department, they plan to buy an integration product. This product will enable their Java application to talk with the finance application through the Java Connectors technology.

Issues of the Data and Operation Oriented Architecture

1. **The architecture is complex.** By focusing on the data model and the operations, the architecture soon becomes about technologies (database relational model, EJBs, connectors) and distances itself from the business requirements. The architecture describes the application in terms that only software engineers can understand. This creates a **communication gap between application architects and people who really know the business**, which results in frustration, delays and complex applications that often only poorly implement business requirements.

In the past, attempts have been made to fix this problem by adapting or creating new development methodologies. These methodologies try to enforce and structure communication between architects and people who understand the business. As one might have expected, trying to enable good communication between parties that talk a different language by enforcing methodologies only had limited success.

In the ACME Inc. scenario, a business person will most likely not be able to participate in the design of the application architecture as this would involve being comfortable with relational database schema design, EJB, JSP and connector technologies, which are all fairly complex. To describe the application, the business persons and architects will have to rely on high level documents (often Word documents). Those documents may be poorly written and maintained, as the architects at ACME Inc. want to start implementing the application as soon as possible and the business persons are too busy.

2. **Limited enterprise-wide reusability.** A good architecture enables reusability. If the data model and the operations are well designed, they can be reused in other projects. Object oriented architectures have gone a long way towards enabling reusability. Unfortunately, they only enable reusability with the same

programming language, and within the same application. But what if there is this great business logic written in C++ code deployed on the HR department server that could be reused in a finance application written in Java deployed on some other server? This proves to be a difficult task in most applications today.

In the past attempts have been made to fix this problem by leaving the architecture unchanged, and adding a layer on top of the application to expose the data and operations to other applications across the network, possibly written in other programming languages. This was in particular the promise of CORBA, which failed to materialize because of the complexity of adding enterprise-wide reusability on top of traditional data and operations oriented architectures. As we'll see in the rest of this article, Web services standards do not fall in the same trap because they put more emphasis on the exchanged documents and the processes.

Let's go back to ACME Inc. Their application will perform as designed, but what happens if they want to reuse some of the operations they developed, like creating an expense report, from another application deployed in some other server? It might be possible if the other application is also written in Java, but it will become more complex otherwise. What about accessing some other service from the expense report application, like sending an expense report to a financial application? With the traditional architecture, these scenarios fall under the magic umbrella of integration, which implies buying expensive products and consulting.

The Document and Process Oriented Architecture

AMCE Inc. was already handling expense reports before a new IT system was designed for this purpose. Their system was based on partners exchanging paper forms and following a clear business process (as illustrated in figure 1). This low-tech system worked for years, was very well understood by the business people, could be easily modified, and closely matched the business requirements. Its design was based on:

1. The expense report form, on paper – first filled by the employee, signed by the manager and reviewed by the HR department.
2. The process – starting with the employee creating the expense report, then transmitting it to his manager who either rejects or approves it before sending the form to HR department who finally communicates with the finance department.

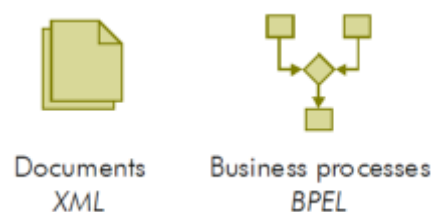


Fig. 2 – Focusing on Documents and Business processes

Similarly, with the Document and Process Architecture, the architect focuses on the design of:

1. **The schemas for the XML documents** involved in the system. A schema is a formal way to define the structure, content and semantic of an XML document.

For ACME Inc. this means creating a schema defining the expense report document, which will typically contain the same information as the paper form ACME Inc has been using for years.

Schemas are written in one of the existing XML schema languages, such as W3C XML Schema or OASIS RELAX NG. In some cases, schemas don't need to be designed from scratch, as standard schemas are available for many verticals (see for instance the OASIS Universal Business Language initiative).

- 2. **The business process.** The business process is formally described with the BPEL language, designed and pushed by IBM and Microsoft among others, and soon to become an OASIS standard.

Once the XML schemas and business process are defined, the process is deployed in a BPEL engine. The BPEL engine runs on a server and takes care of executing the business process.

Simple Web applications are created to enable end users to interact with the business process. These applications communicate with the BPEL engine by exchanging XML documents through Web services. In the ACME Inc. scenario, an application is created to enter new expense report data. Employees fill-out forms in their Web browser. .Once all the information about the expense report is entered, the Web application submits the expense report XML document to the business process through a Web service.

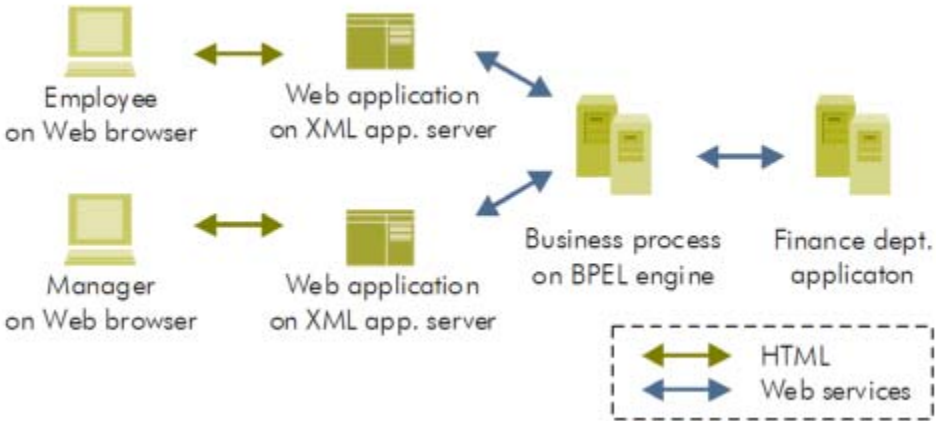


Fig. 3 – Document and Process Oriented Architecture

XML Application Servers

In the Document and Process Oriented Architecture, Web applications are on one hand generating an XML document to be sent to business process engine, and on the other hand generating HTML, or XHTML to create a user interface. Since Web applications deal mostly with XML documents, they are best implemented on top of an XML application server.

XML application servers implement standards created to deal specifically with XML documents, like XForms to create form-based user interfaces, XSLT to transform XML documents, Web services to allow XML documents to be exchanged between

applications, and schema languages to define the structure, content and semantic of XML documents.

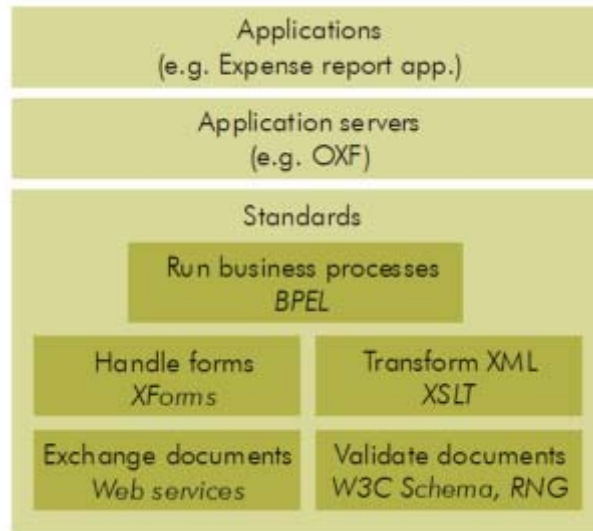


Fig. 4 – XML Application Servers

XML application servers embody a clear separation between standards (here defined by standard bodies, like W3C or OASIS), application servers (created by vendors that implement the standards) and applications (written against standards and running on top of application servers). Companies that decide to write applications on top of XML application servers benefit from reusability, thanks to specifications defined by standard bodies, and allow for knowledge reuse, as more and more people are familiar with XML technologies, which are generally more accessible than J2EE technologies.

BPEL engines can also change the way software is architected. For instance, the traditional architecture of the expense report application requires designing a relational database schema, since expense reports have to be persisted before they are approved by managers. With a Document and Process Oriented Architecture, once the Web application has created an XML expense report document, it sends it to the BPEL engine. BPEL engines are designed to execute long-running processes, and no particular step is required to ensure the temporary persistence of the expense report document: the task of making sure that no data is lost during the execution of the process is offloaded to the BPEL engine.

Benefits of the Document and Process Oriented Architecture

1. **Reduced complexity.** Companies traditionally work by exchanging documents and implementing business processes. This means that when designing an IT system, it is natural to concentrate on the documents exchanged and the business processes, as opposed to relational database models and the Enterprise JavaBeans.

This allows domain experts to participate in the design of the architecture, which increases the chances that the system will actually satisfy the business

requirements.

This also reduces complexity, not only because XML technologies are generally simpler and more accessible than the technologies used in the J2EE or .NET world, but also because more work is offloaded to the underlying XML application server.

2. **Enhanced reusability.** The architecture naturally breaks the system into pieces that communicate with Web services. This enables enterprise-wide reusability, and solves the "integration problem": integrating the expense report application with the finance server consists, at the end of the expense report business process, of calling a Web service provided by the finance department.

Conclusion

In this article we have seen that designing software architectures around documents and processes provides several benefits, including reduced complexity and enterprise-wide reusability. XML applications servers will play an increasing role as the platforms on top of which these applications are the most efficiently created.