

Component-Based Development With MQSeries Workflow

By Michael S. Pallos

Satisfying business needs while maintaining the flexibility to incorporate new requirements in a timely fashion is a constant technology challenge. The ability to increase speed-to-market while reducing (or at least controlling) costs is a chief concern for today's managers. This article explains how to extend the MQSeries infrastructure using component-based development with MQSeries Workflow, a solution that improves time-to-market while reducing costs.

What Are Components?

Components are executable programs. Yes, that includes beans, class files, and byte code. Components leverage existing resources. The use of components promotes reusability, reducing risk and cost. The greater the granularity, the stronger the ability to reuse and leverage.

So why would an organization choose to use components?

Grains of sand are analogous to lines of code. They're difficult to pile up into a cohesive entity, prone to blowing apart in a strong wind, and difficult to differentiate when piled into mounds. However, bricks made of sand are analogous to executable programs (here called components). They present a solid look and feel, carefully shield the user from examining internal structures, and do one thing well. Indeed, you can build almost any imaginable structure by placing them together in a set pattern. That pattern can be reusable. One would rather build with bricks than with sand.

Assembling Components

The challenge with component development is housing the process flow (i.e., execution of the set pattern).

Software architects must answer several questions:

- Where is the component state information kept?
- What is the interface to the component?
- Where is the process flow kept?
- How is the information from component A passed to component B?

Service-Oriented Architecture (SOA) addresses each of these questions, offering a viable solution for plugging components together.

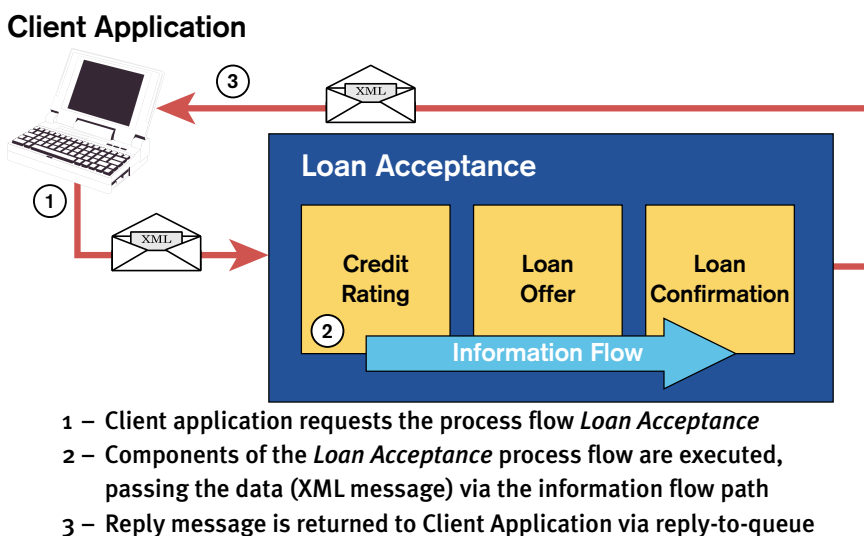


Figure 1: Elements of a Service-Oriented Architecture

Service-Oriented Architecture

SOA is the aggregation of components satisfying a business driver. It comprises services, components, and client applications (see Figure 1). A service is simply a grouping of components (executable programs) to get the job done. An example would be loan acceptance. Components are binaries that do specific tasks. These binaries usually have one goal in life: Do a job and do it well – whatever the job may be. In this example, the job may be “obtain credit rating” or “create loan offer.”

The client application is the program requesting the service. For example, request (loan worthy) and wait for the reply. The client gathers the applicant's information and requests the service.

The key focal point of an SOA is the business process. The grouping of components, allowing the application process pattern to more closely represent the business, satisfies the business process. This higher level of application development provides a strategic advantage, allowing one to focus more on the business requirement. The business driver is satisfying the requirements, as opposed to the technical mechanism of application development. A service is built using components. These components are executable programs. So cost of development, cost of ownership, and risk of implementation are all reduced.

Component-Based Development With MQSeries

One challenge with MQSeries, as with most application development, is the ability to link components together independently of each other without having to code for the interface. How can one link together independent components without the knowledge of the component path and the data required for the specific component? Previously, one would have had to ensure the data and component interface, thereby hard-coding the components together. This didn't allow for truly autonomous component development. We can now address this complication with XML and the message payload.

Another challenge is linking together process flows

(i.e., how to plug components together non-programmatically). Ideally, the solution should:

- Allow for a graphical representation of the process flow
- Allow for process flow changes without requiring re-linking or re-compilation of applications
- Monitor the environment and state information
- Allow for component state changes.

MQSeries Workflow is one option for connecting components via a process flow.

XML Payload

You can place information for the component in the payload, also known as the user data section of an MQSeries message. Use of eXtensible Markup Language (XML) makes this appealing. The message payload will contain all input required for every component in the process flow. Each component works only with the XML child element, or sub-record, that's relevant to its processing, keeping all other payload data intact.

Figure 2 contains the entire payload of the message prior to loan offer component processing. Let's assume we're processing a loan offer. The component would copy the entire input payload to the output payload, only updating the loan offer section of the XML message. This

lets the payload contain everything required for all components. So the amount of components in a process flow is irrelevant, as is the number of components or order of execution. Each component consumes only the portion of the XML message relevant to that component. This is

```
<?xml version="1.0"?>
<! --- Loan Application Processing Example --->
<! --- Developer: Michael S. Pallos - 'big iron rocks!' --->
<Complete Payload>
  <Applicant>
    <FirstName>Michael</FirstName>
    <LastName>Pallos</LastName>
    <SSNumber>123456789</SSNumber>
    <AmountRequested>50000</AmountRequested>
    <ErrorMsg> </ErrorMsg>
  </Applicant>
  <CreditRating>
    <RatingCode>550</RatingCode>
    <ErrorMsg> </ErrorMsg>
  </CreditRating>
</Complete Payload>
```

Figure 2: XML Message Containing the Entire Payload, Prior to Loan Offer Processing

simplified by having the XML child element, sub-class, and name consistent with the element name. Figure 3 displays the message payload after the loan offer component has completed.

```
<?xml version="1.0"?>
<! --- Loan Application Processing Example --->
<! --- Developer: Michael S. Pallos - 'big iron rocks!' --->
<Complete Payload>
  <Applicant>
    <FirstName>Michael</FirstName>
    <LastName>Pallos</LastName>
    <SSNumber>123456789</SSNumber>
    <AmountRequested>50000</AmountRequested>
    <ErrorMsg> </ErrorMsg>
  </Applicant>
  <CreditRating>
    <RatingCode>550</RatingCode>
    <ErrorMsg> </ErrorMsg>
  </CreditRating>
  <LoanOffer>
    <TypeOfLoan>Signature</TypeOfLoan>
    <Qualified>Yes</Qualified>
    <RiskScore>10</RiskScore>
    <ErrorMsg> </ErrorMsg>
  </LoanOffer>
</CompletePayload>
```

Figure 3: XML Message Containing the Entire Payload, After Loan Offer Processing

Clusters

MQSeries clusters simplify component availability and administration. Clustering permits multiple instances of a queue (with the same name) on different queue man-

agers. This facilitates workload distribution, fail-over, and balancing. Multiple instances of components can run on multiple platforms under different queue managers – all in the same cluster, leveraging the enterprise, without burdening the application developer.

Combining Components

Process flows contain steps, satisfying a business requirement. This maps nicely to an SOA, which contains services comprising components. A business workflow contains process flows comprising steps. MQSeries Workflow supports the use of manual processes, human intervention, and additional layers of complexity. However, for our sample SOA implementation, we're only incorporating the straight-through processing aspects. We're going to build workflow process flows comprising components that are executable programs to satisfy our business requirements – and we're going to build the process flows graphically.

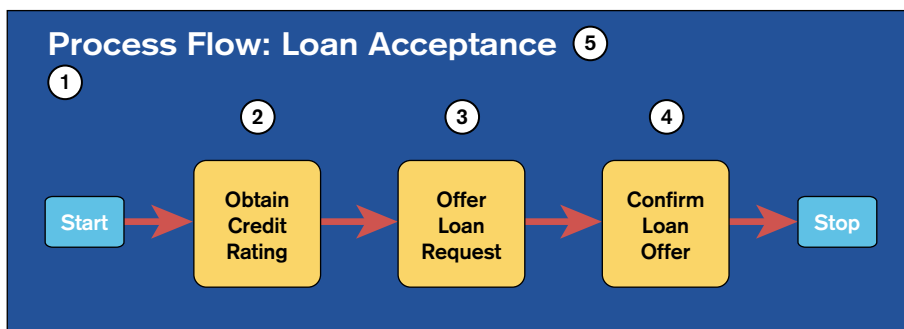
Build-Time and Run-Time

MQSeries Workflow consists of two parts: build-time and run-time. Build-time is used to model business processes, (i.e., create the process flow). Once the process flow is created, it's stored in the workflow UDB database. Figure 4 shows a process flow. Figure 5 shows the build-time process model.

Run-time is the workflow execution engine. The process flows created with build-time are imported into run-time. This process converts the process flows to process templates. This makes it easier to change and enhance the process models or flows while the process templates execute via run-time.

Components

Programs (components) to be executed with the process flow are registered with workflow. This can be done during the build-time phase. Just specify what component to execute at what step.



- 1 – Create new process flow
- 2 – Add Step 1: Obtain Credit Rating (register binary *Credit Rating*)
- 3 – Add Step 2: Offer Loan Request (register binary *Loan Offer*)
- 4 – Add Step 3: Confirm Loan Offer (register binary *Loan Confirmation*)
- 5 – Save process template as *Loan Acceptance*

Figure 4: Process Flow

During this setup phase, you can configure the component to execute automatically, which conserves system resources. The data inputs are irrelevant since all the information required is in the payload in XML format. Simply state that a variable XML string is arriving. The application is responsible for parsing the message.

Client Applications

The client application is the element requesting the process flow execution. There are two choices for client invocation: the Application Programming Interface (API) and XML message interface.

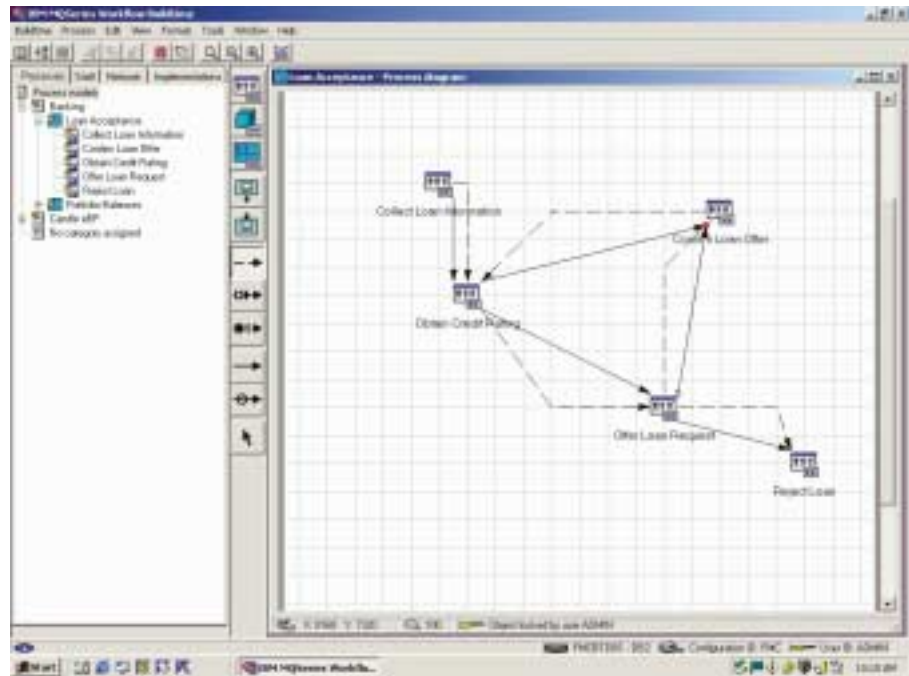


Figure 5: Process Flow Created With Build-Time

API

Via incorporation of the API, MQSeries Workflow API calls are placed into the client application. The downside of this approach is the intrusiveness into the client application. However, a benefit is that we can examine the state of the process flow. This is available to the API, since we established a session and requested a service, so our client application has a session identification (ID) to reference. This method is synchronously implemented.

XML Message Interface

Using the XML message interface, the XML message contains all the required process template and session information. The XML message is passed to the workflow engine. So, establishing a session isn't necessary and the request is processed asynchronously. The process flow can sync up with the requesting client via the MQSeries message descriptor correlation ID. Also returned to the client is a result object with return codes, allowing for error handling. The disadvantage of the XML message interface is the inability for the client to monitor, manage, or obtain information – from the process flow.

As shown in Figure 6, the MQSeries Workflow message-based interface requests services from MQSeries Workflow.

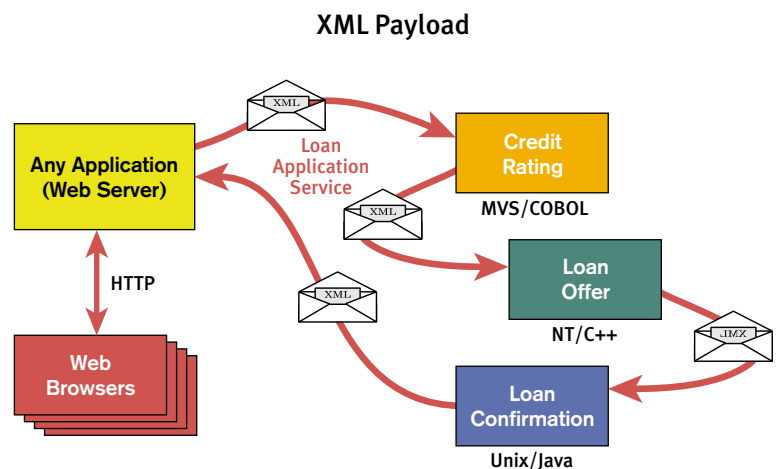


Figure 6: XML Message Interface Process With WebSphere Process Manager

Benefits

Leverage initial investment – SOA leverages the initial organization's investment in hardware, development languages, development resources, and architecture paradigm. Since a process flow is the aggregation of existing components, the developer isn't responsible for ensuring compatibility among the components.

Faster time-to-market – As organizational component libraries grow, the time-to-market reduces exponential-

ly. New initiatives support the rapid development of new business services via reuse of existing process flows or services (services can call other services) and components, when possible. Reusing existing components reduces the time needed to gather requirements, design, develop, and test. If a component works in its current state, we simply add the component to our service. It should continue to work. Only system testing on the complete process flow is required.

Reduced cost – As business demands evolve and new requirements emerge, the cost of creating technology to satisfy them is greatly reduced if portions of the process flows already exist as components. The new application merely builds a process flow that includes the components. This mitigates the development effort required to satisfy business needs.

Risk mitigation – We reduce risk by using components to develop process flows in an SOA. We do so by:

- Reducing the probability of introducing new failures in application development
- Closer mapping of business requirements to the technical implementation
- A simpler method for problem determination
- A reduced (if not eliminated) learning curve for application developers.

When we reuse an existing component, the probability that it will function properly is high. With component development, we don't get to "open up" the component and tweak it, so the probability of introducing a failure into an existing component is extremely low. JavaBeans has developed strong usage partly because you cannot break things you can't open.

The visual representation of the process flow provides a mechanism for simplified problem determination. The specifications of a business service give a clear picture of the components that comprise the process flow.

Risk reduction is also realized by allowing the organizational development staff to maintain its current Integrated Development Environment (IDE), program-

ming language, operating system, and databases. The learning curve for development is greatly reduced, enabling the staff to maintain its area of expertise.

Continuous improvement cycle for business process – An SOA provides a clear representation of business process flows identified by the order of the components used in a particular service. This provides business analysts an ideal environment for monitoring business operations. Process modeling is reflected in the process flow. Process manipulation is achieved by reorganizing the flow in a pattern. This allows for the modification of the process flow while monitoring the effects. This manipulation and monitoring of the business process supports continuous improvement.

Conclusion

Component-based development with MQSeries Workflow implementing an SOA leverages an organization's existing investment, taking advantage of current resources, including developers, software languages, hardware platforms, databases, and applications. Component development offers a practical, sensible approach to delivering business return, while leveraging many aspects of the existing organization. Participating in properly executed component-based development will reduce costs, reduce risk, and increase productivity. **IBM**



Michael S. Pallos is a senior solution architect for Candle Corp. He actively architects enterprise solutions for Candle's Consulting & Services clients. He has 15 years of experience in distributed systems and holds numerous copyrights for international applications he developed.

e-Mail: michael_pallos@candle.com
Website: <http://www.candle.com/>