Tuxedo Control Migration

BEA WebLogic Workshop 8.1 included a component called the Tuxedo Control. This control allowed easy development of applications that included WebLogic Workshop features and Tuxedo services. Applications of this type are often called composite applications.
The Tuxedo Control allowed easy development of applications that included WebLogic Workshop features and Tuxedo services. Applications of this type are often called composite applications.

In BEA WebLogic Workshop 9.2 the Tuxedo Control is not included. As a result, composite applications utilizing the Tuxedo Control will need to use another mechanism to access the required Tuxedo services. This whitepaper describes three alternatives that can be utilized to access the Tuxedo services that the Tuxedo Control was being used to access.

The Tuxedo Control performed the following functions:

1) Created a WebLogic Tuxedo Connector (WTC) connection.
2) Mapped Java data types into a Tuxedo buffer.
3) Performed a `tpcall` or `tpenqueue` to send the buffer to Tuxedo.
4) Mapped the returned buffer into a Java data type.
5) Mapped any returned errors into a Java exception.

The mapping performed was done by matching parameter and property or field names to Tuxedo field names for Tuxedo fielded buffers. This matching applied to nested properties or fields as well. This mapping represented the bulk of the work performed by the control and was dependent upon the method signature defined in the Tuxedo Control extension and the types of the Tuxedo buffer fields.

Alternatives to the Tuxedo Control

The next three sections describe three alternatives to using the Tuxedo Control.

Custom Java Control

One approach to replace the Tuxedo Control is to develop a custom Java control. The complexity of this approach depends mostly on the complexity of the Tuxedo buffer that the service requires and the complexity of the Tuxedo buffer the service returns. For simple non-fielded buffer types such as STRING, MBSTRING, and CARRAY, the current Tuxedo Control extension should only have a single parameter and return a simple type. This makes the mapping between Java and the Tuxedo buffer very straight forward. For fielded buffers the process is a bit more complex as each field in the buffer needs to be populated from the Java parameters the Tuxedo Control extension expected. A brief summary of what must be done follows.

The Java control would need to first create a WebLogic Tuxedo Connector connection. This code might look like:

```
  Context ctx;
  TuxedoConnectionFactory tcf;
  TuxedoConnection myTux;
  ctx = new InitialContext();
  tcf = (TuxedoConnectionFactory)
ctx.lookup("tuxedo.services.TuxedoConnection");
  myTux = tcf.getTuxedoConnection();
```

This connection then is used for all WTC interactions.

For non-fielded buffer types, the Java control then creates the appropriate buffer type using something similar to:

```
    TypedString myData;
    myData = new TypedString(toConvert);
```

Where toConvert is the parameter that was passed to the Java control. At this point the buffer is ready to be sent to the Tuxedo service via something like:

```
    Reply myRtn;
    myRtn = myTux.tpcall("TOUPPER", myData, 0);
```

Where TOUPPER is the name of the Tuxedo service being called and myData is the Tuxedo buffer being sent to the Tuxedo service. The returned buffer from the Tuxedo call can then be accessed using something similar to:

```
    myData = (TypedString) myRtn.getReplyBuffer();
    return (myData.toString());
```

For fielded buffers the setup and process is a bit more involved. For VIEW and VIEW32 buffers, the view description file needs to be converted to a class. This should have already been done to use the Tuxedo Control by using the viewj or viewj32 utilities. Likewise for FML and FML32 buffers, the field table files must have been converted to field table classes. Again this should have already been done for use by the Tuxedo Control and these classes can be reused in the Java control that is replacing the Tuxedo Control instance.

With these pieces in place, what's required now is a method that creates the appropriate Tuxedo buffer type and places the method parameters values into the buffer. To create an FML32 buffer, one would use some code similar to:

```
    myData = new TypedFML32(new MyFieldTable());
```

Where `MyFieldTable` is the field table class that was created with the `mkfldclass32` utility. Now each of the parameters and their associated fields must be placed into the Tuxedo typed buffer. The Tuxedo Control used reflection to discover the field and property names at runtime.

A simpler approach is to simply hard code the population of the Tuxedo buffer based upon the known signature of the control. This is done using something similar to:

```
myData.Fchg(MyFieldTable.MyString, 0, toConvert);
```

Where `MyString` is the field name that is to be added to the buffer and `toConvert` is the parameter that was passed to the control. This needs to be done for each parameter and for any contained fields or JavaBean properties of the parameter. Once the buffer has been populated, then invoking the Tuxedo service can proceed as described above.

Web Service Control and Salt

Another method of accessing Tuxedo services from a WebLogic Workshop (WLW) application is via the Web Service control and the new Salt v1.1 Tuxedo option. Salt (Services Architecture Leveraging Tuxedo) is a native Web Services gateway that provides SOAP over HTTP(S) access to Tuxedo services. This approach for replacing the Tuxedo Control should be very straight forward for most applications and relatively simple to do.

The Web Service control provides a mechanism for accessing Web Services from within a Workshop application. All that is required is the WSDL of the Web Service to be invoked. Combined with Salt's totally configuration based model for exposing Tuxedo services as Web Services means that there is no Java coding to be performed. Using Salt or the Tuxedo Service Metadata Repository, one can easily generate the WSDL required for the Tuxedo service being called.

The steps for this approach are:

1) Install Salt.
2) Add the Tuxedo service definitions to the Tuxedo Service Metadata Repository.
3) Create the Salt configuration file that describes the services offered by each Web Service gateway (GWWS).
4) Add the metadata repository server to the Tuxedo configuration file, if that has not already been done.
5) Add the GWWS instances to the Tuxedo configuration file.
6) Restart the Tuxedo application.
7) Use the WSDL file generated by Salt to create an instance of the Web Service control.

The signature of the Web Service control generated from the Salt gateway should be nearly identical to the signature of the Tuxedo Control that is being replaced. If the signatures of the method are identical to the one defined in the Tuxedo Control, then the Web Service control can be used directly. If there are differences in the signatures, then the code using the Tuxedo Control would need to be refactored to match the new signature.

For more information about BEA Salt, refer to the Product Documentation web site at the following URL: http://e-docs.bea.com.

Web Service Control and AquaLogic Service Bus

This approach is nearly identical to the previous approach. The primary difference is that AquaLogic Service Bus (ALSB) is being used to provide the Web Service access to the Tuxedo service. In this scenario, ALSB is configured with a Tuxedo business service that accepts an any XML message. An HTTP proxy is defined that accepts a SOAP request defined by the WSDL file generated from the Tuxedo service metadata repository. This proxy is routed to the Tuxedo business service. Once the proxy and business service have been set up, the steps to use this Web Service with the Web Service Control are identical to the steps outlined in the previous approach.

For more information about BEA Salt, refer to the Product Documentation web site at the following URL: http://e-docs.bea.com.

Comparison of the Approaches

Each of the alternatives has advantages and disadvantages. The following sections compare and contrast the alternatives.

Factor	Tuxedo Control	Custom Java Control	BEA Salt	BEA ALSB
Coding Needed	Minimal	Yes	No	No
Service Invocation/Tuxedo Domain Decision	At WLS	At WLS	Tuxedo	ALSB
Need Additional Product	No	No	Yes	Yes
Complexity	Low	High	Low	Medium
Security Propagation	Yes	Yes	No*	No
Transaction Propagation	Yes	Yes	No*	No
Failover Capability	Domain Level	Domain Level	Depends upon client	Domain Level
Performance	Medium	Medium	High	Medium

* Planned for future release