LegSem
April 2007

LegStar

Presentation

I. Goals

LegStar provides development and runtime features for Java developers who need to integrate legacy functions, such as IBM COBOL-CICS programs.

With LegStar, legacy functions can be exposed as Web Services or as POJOs (Plain Old Java Objects.

Unlike other solutions available, LegStar is open-source and relies on recent standards in 3 crucial areas:

- XML Schema as a universal meta-data language
- Language to Language mapping, based on annotations
- API stacks for Web Services

LegStar takes the standpoint of Java developers, rather than COBOL developers, by making use of familiar patterns, frameworks (JAXB, JAXWS), tools (Ant, Eclipse) and targeting Java runtimes such as Apache Tomcat.

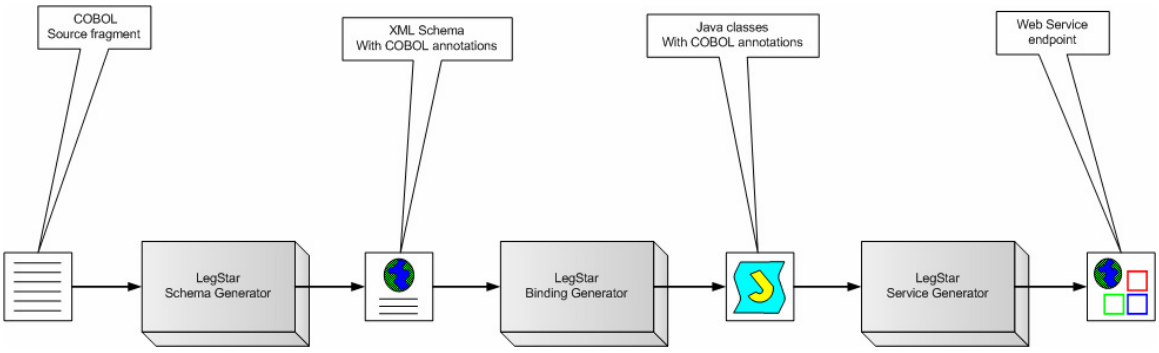
By using standard Web Services APIs, LegStar will work with the most advanced Web Services stack without having to provide a SOAP stack itself. Most other integration solutions have developed proprietary SOAP stacks they have trouble maintaining.

Because there is less infrastructure to worry about, LegStar concentrates on developers real issues (such as how to deal with COBOL "REDEFINES"). The accent is on XML schema to COBOL mapping, using XML schema annotations to account for COBOL idiosyncrasies.

It is believed that such "enriched" XML Schemas will find uses beyond legacy integration in the years to come.

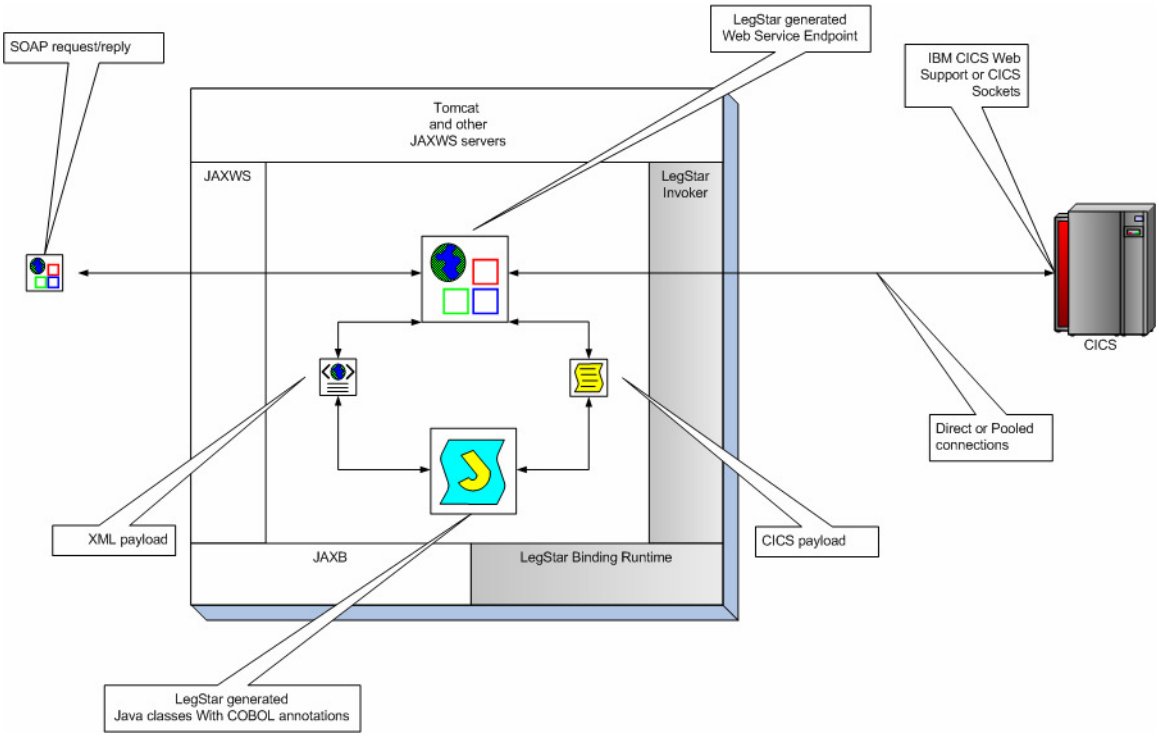
II. Architecture

With LegStar, developers would follow a 3 steps process in order to "Web Service-enable" a COBOL-CICS program:



Each one of these steps can be executed with an Ant script or an Eclipse plug-in.

From a runtime perspective, this is how a request/reply message exchange pattern would flow in an IBM CICS environment:



SOAP request/replies are first processed by a standard Web Service stack. The LegStar-generated endpoint uses the JAXWS standard API to communicate with the SOAP stack.

The XML payload extracted from SOAP requests is handed over to the LegStar-generated Java classes which use the standard JAXB binding framework to parse the XML. The Java classes use the LegStar COXB runtime to convert data into the format expected by CICS. This conversion includes Unicode to EBCDIC conversions, numeric conversions, etc…

Once data is in CICS format, the endpoint uses a transport independent layer called LegStar Invoker to invoke a remote CICS program. The actual transport is selected at runtime from a configuration file and offers the following options:

- Direct Socket connectivity to CICS
- Pooled Socket connections to CICS
- Direct HTTP connectivity to CICS
- Pooled HTTP connections to CICS

On the way back, CICS data is converted to XML and then wrapped in a SOAP reply.

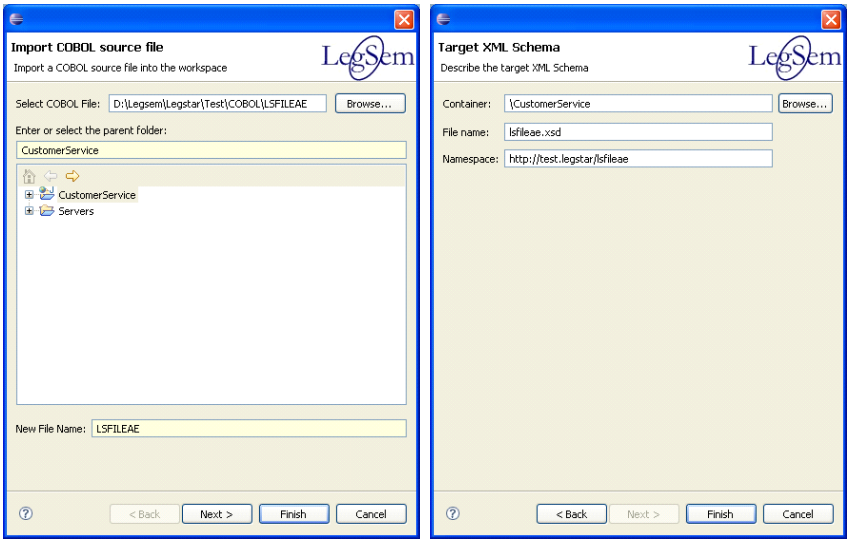
The CICS footprint of this architecture is minimal since all SOAP/XML processing occur off-host.

III. Eclipse integration

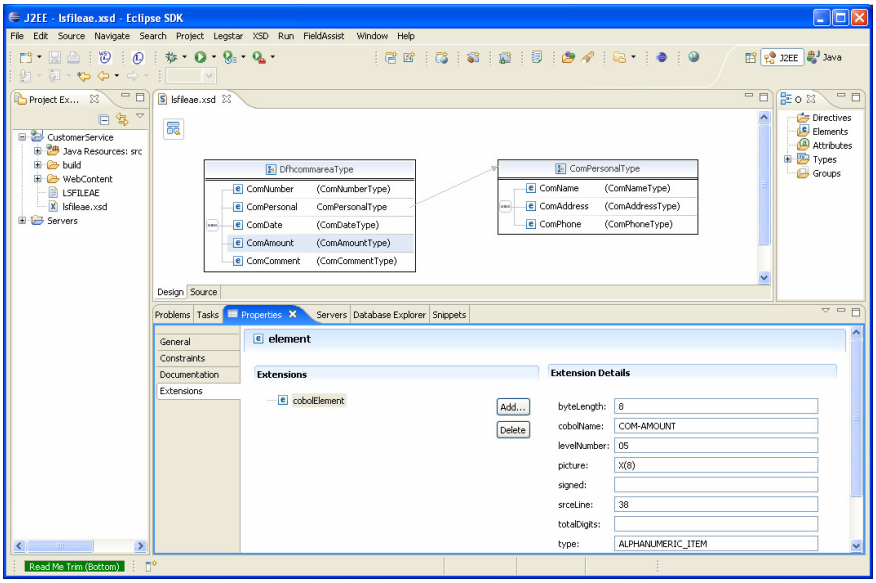
As already mentioned, the developers' tools are all available as Ant tasks. In addition, as a convenience for Eclipse developers, a set of plug-ins provide a user interface for each of the 3 development steps:

XML Schema generation:

The process starts by importing some COBOL source into the Eclipse workspace. Then a few parameters are needed to describe the generated XML Schema:



The generated XML schema is editable with standard Eclipse editors:

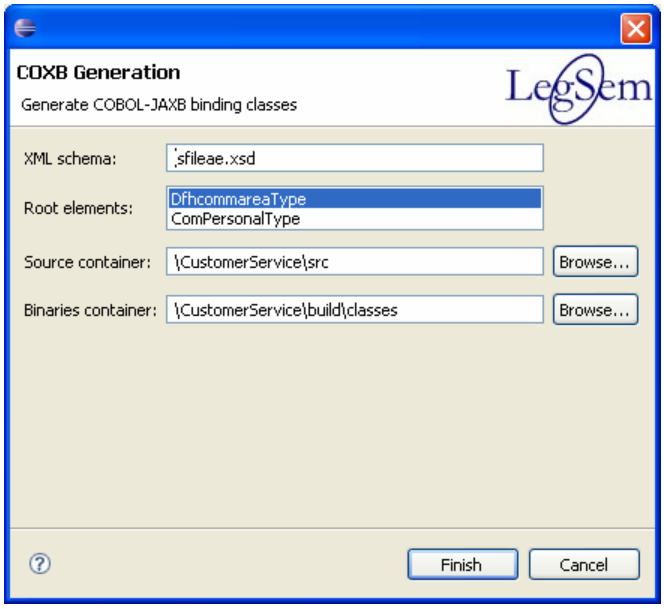


Observe the extensions used to annotate the XML Schema elements with COBOL meta-data.

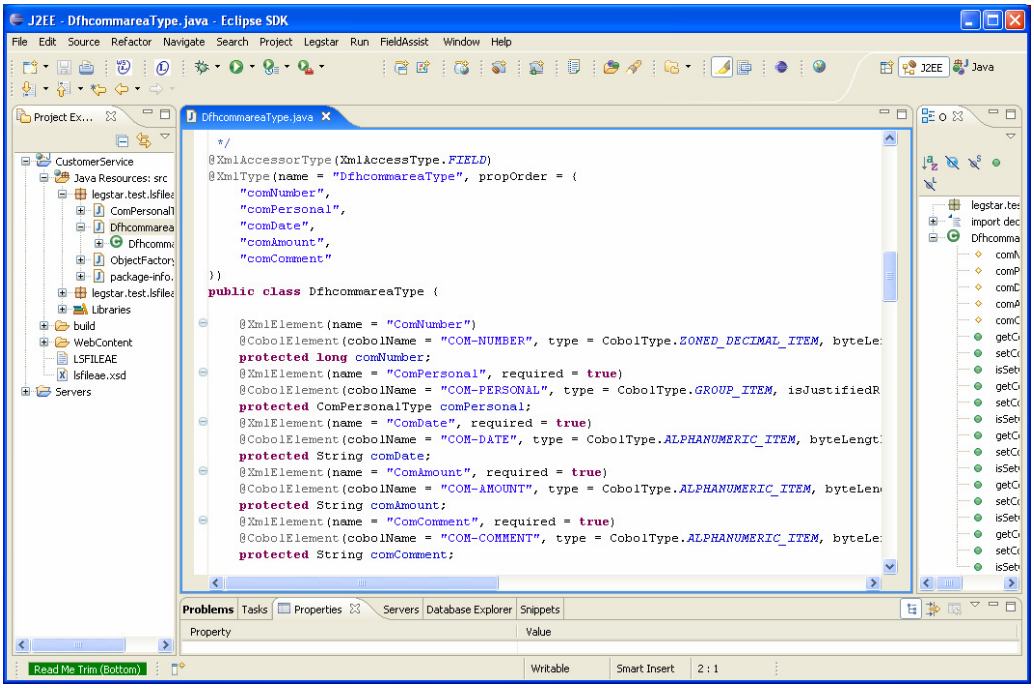
COBOL to XML binding generation:

The next step is to generate binding classes from the annotated XML Schema. These classes will be responsible for marshaling/un-marshaling XML into a z/OS data format as expected by CICS for instance.

The wizard is started by right-clicking on the previously generated XML Schema:

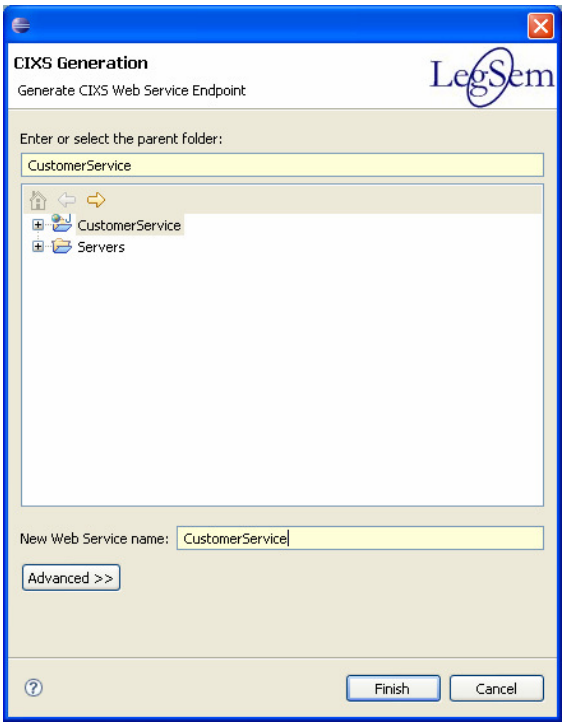


This dialog allows you to select multiple complex elements. Each complex element will be mapped to a standard JAXB class with additional COBOL annotations:

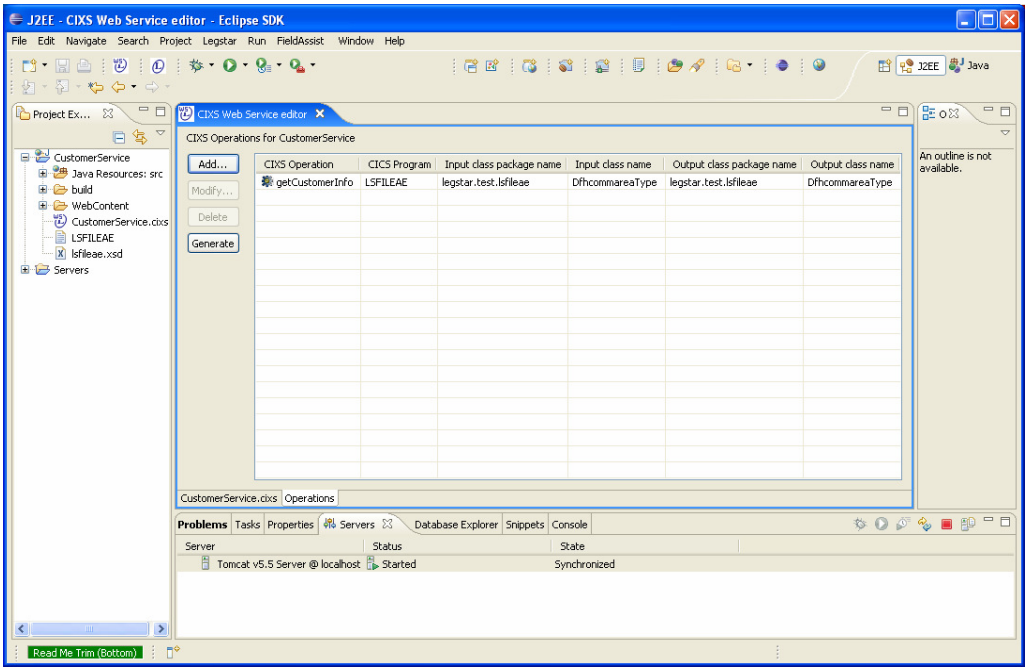


Web Service generation:

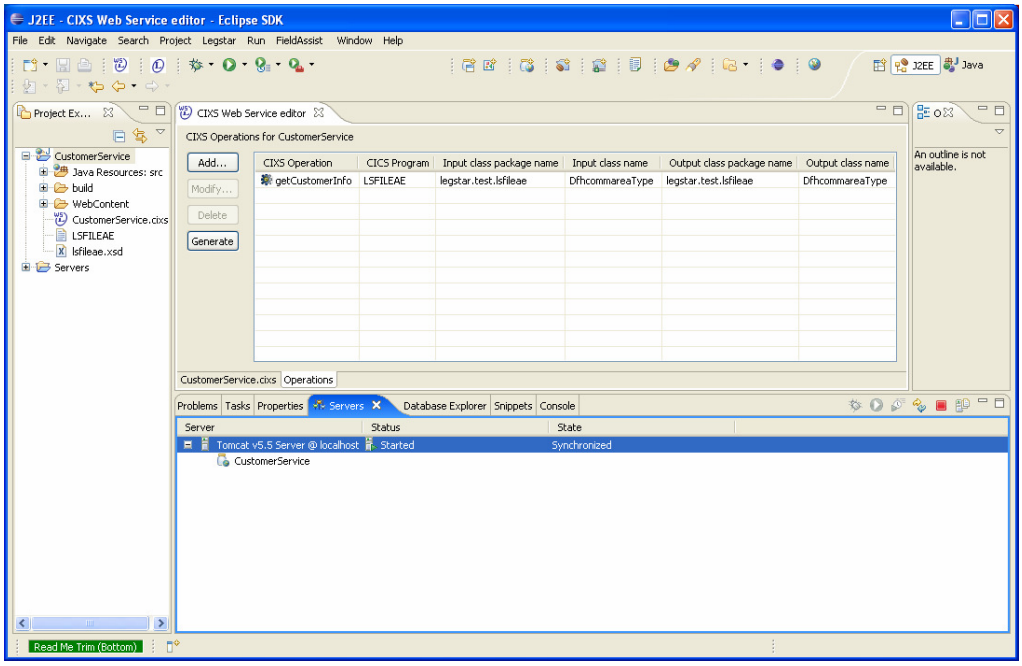
The final step in the process is to generate a JAXWS Endpoint where an operation maps to a remote CICS program.



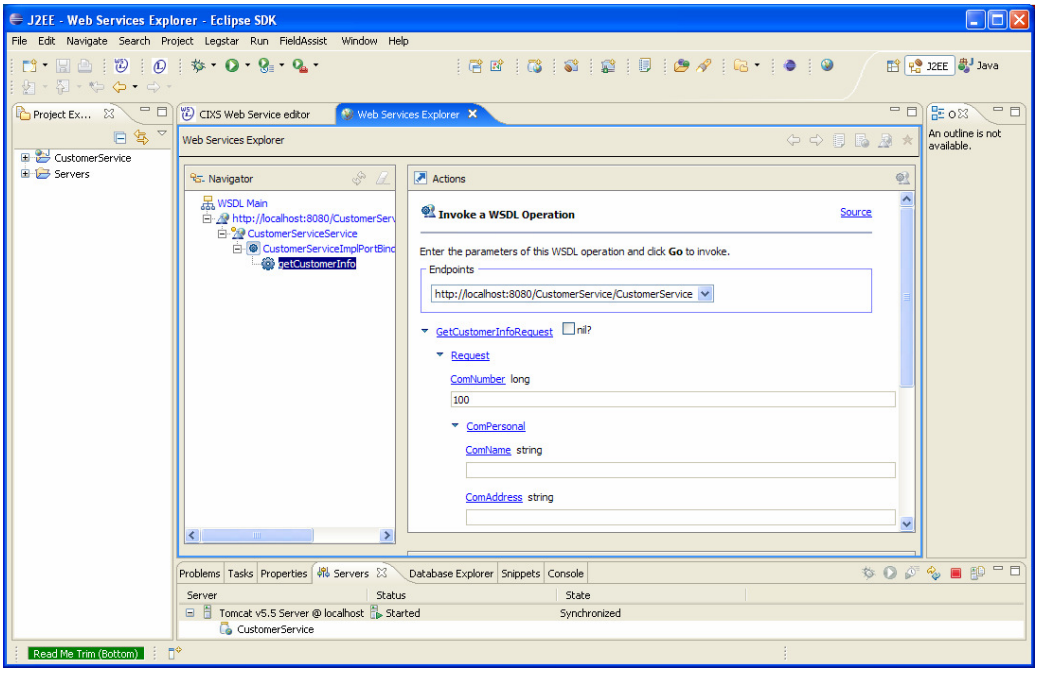
After naming the Web Service endpoint, an editor allows you to create operations. Each operation corresponds to a CICS program:



The "Generate" button will create all the necessary artifacts for a Web Service Endpoint. In the context of Eclipse with WTP, you can directly drop the project onto an instance of a Server (in this case, Apache Tomcat):



Once the service is deployed, you can test it using the WTP Web Service Explorer:



IV. Wrap up

Although this is not an in-depth presentation of LegStar, hopefully you now have a better understanding of the ideas behind its architecture.

By riding the Java wave, an integration solution such as this one can make it easier and faster for non-CICS programmers to reuse existing functions.

Naturally there are commercial products already providing integration solutions but most of them have been targeting COBOL developers and CICS system engineers.

As the mainframe population is shrinking and under more and more day to day pressure, it is important that new solutions, powered by open-source, come to the rescue of those developers whose ecosystem is built from Java, Eclipse, J2EE and advanced Web Services.