Jsr181Requirements

Problem Analysis

In absence of a functional specification, this section formulates a few major assumptions and axioms all further discussions in this proposal are based upon.

Requirements Analysis

- Fully JSR-181-compliant implementation of “Web Service Metadata for the Java Platform”. More specifically, the implementation supports at least all mandatory operational modes, annotation types, error checks and processes specified in JSR-181.
- Extensibility: implementation is based on an open, modular architecture, which makes support for extensions and new runtimes easy.
- Platform-neutral option: the implementation can be integrated with any Web Services platform through open interfaces. Adequate abstractions are provided to encapsulate the implementation details for each platform.
- Supported platforms: Axis (on Tomcat), J2EE 1.4/JSR-109 (Geronimo?), others
- Deployment: the implementation provides (API) interfaces for runtimes to launch the processor programmatically, e.g. when files are dropped into monitored directories.

- Source code compatibility: in Java to Web Service mode, the processor accepts both Java Web Service implementation file formats .class and annotated Java Web Service source files. The preferred input format is the annotated Java Web Service source file. For a given annotated Java Web Services file, the processor produces equivalent Java Web Services for each supported target runtime. [Note that the produced Web Service artifacts need not be byte code compatible for all runtimes.]

- Ease of use: the processor simplifies development and deployment of Web Services as far as possible; adequate default values are used whenever possible.
- All requirements specified in JSR-181 apply. This includes the operational modes (Start with Java and Start with WSDL).

Use Case Analysis

Figure 1 gives an overview over the anticipated use cases for proposed JSR-181 implementation. The actors are:

- "WS Developer": Individual who develops a Web Service using command line tools, an IDE, or other tools.
- "Axis Runtime" (example): software such as Apache Axis that invokes the processor to manage (i.e. generate or deploy) Web Services.
- "IDE": Integrated development environment (software) that uses the processor e.g. to provide real-time feedback to its user, while the user edits files.

Figure 1: Use case analysis for the JSR-181 implementation.

Use Case: Generate Code

"WS Developer" and "IDE" invoke WSM to generate (Java) code from a service definition (.wsdl) file. The generated code consists of skeletons (annotated Java Web Service source file) and simplifies and accelerates the development of the Web Service when a service description exists. "Generate Code" corresponds to the JSR-181 Start with WSDL processor mode.

Use Case: Validate

"WS Developer" and "IDE" invoke WSM to validate a Web Service implementation against a service description (.wsdl) file. This is particularly useful to keep initially auto-generated but later modified code in sync with the original service description. For instance, "IDE" may choose to use WSM on the fly to signal any inconsistencies while a file is being edited.

Use Case: Generate Service Description

"WS Developer" invokes WSM to generate a service description from an existing implementation for a Web Service, e.g. to publish the Web Services's capabilities in a standard WSDL file.

Use Case: Generate Web Service

"WS Developer" invokes WSM to auto-generate the artifacts required to deploy a Web Service to a runtime from an annotated Java Web Service file. One of the artifacts can be a service description file (see above).

Depending on the chosen deployment model, a runtime may choose to automate the deployment process by automatically invoking WSM to generate a Web Service from an annotated Java Web Service file if such a file is dropped into a monitored directory. In other words, runtimes that support this automated deployment would take the burden of explicitly invoking WSM from "WS Developer", which in turn would only be required to author the original annotated Java Web Service file. [The Axis runtime currently supports a similar mechanism for un-annotated .jws files.]

The "Generated Web Service" use case corresponds to the JSR-181 Start with Java processor mode.

Use Case: Semantic Error Checking

WSM also check for errors at a semantic level, which cannot be caught by the Java compiler.

Use Case: Package Artifacts

In order to create a deployable Web Service, WSM needs to package the generated artifacts appropriately. For some runtimes, packaging may not be necessary.

Project Deliverables

Deliverables

The use case analysis suggests structuring the JSR-181 implementation into the following deliverables:

- JSR-181 annotation types
- JSR-181 processor with:
    - additional semantic error checking
    - WSDL validation
    - command line interface
    - API
- Processor plug-in(s) for select runtimes with packaging facilities and deployment strategy
- Interface specification for processor plug-ins
- Technology Compatibility Kit (TCK)
- Test Harness (unit tests)
- Documentation
- Feature requests for the Axis project (hook for Axis' monitored directory service)

Non-Deliverables

- SOAP stack: an existing JAX-RPC/SAAJ-based SOAP stack (e.g. Axis) will be used instead

Suggested Road Map

Phase 1

- JSR-181 annotation types.
- JSR-181 processor with:
    - Command line interface
    - Operational modes: Start with WSDL; Start with Java with support for both annotated Java Web Service source and byte-code (.class) input files
- Processor plug-in for Axis runtime
- Interface specification for processor plug-ins
- Technology Compatibility Kit (TCK)
- Test Harness
- Documentation
- Feature requests for the Axis project

Later Phases

- JSR-181 processor with:
    - additional semantic error checking
    - WSDL validation
    - API
- Processor plug-ins for J2EE (JSR-109, e.g. Geronimo), others

- Tight integration of Axis module and automatic deployment through support for monitored directories
- Tight integration with the other Beehive projects, particularly the controls subproject.

Supported JSR-181 Annotation Types

The required annotation types for Java Web Services as specified in JSR-181 are:

- javax.jws.WebService
- javax.jws.WebMethod
- javax.jws.Oneway
- javax.jws.WebParam
- javax.jws.WebResult
- javax.jws.soap.SOAPBinding
- javax.jws.HandlerChain
- javax.jws.soap.SOAPMessageHandlerChain
- javax.jws.SecurityRoles
- javax.jws.SecurityIdentity

Additional annotation types may be added later, for instance to support service controls or asynchronous Web Services.